Print

COSINE Lab Book

Week 1 - Lab Books and Java Reprise

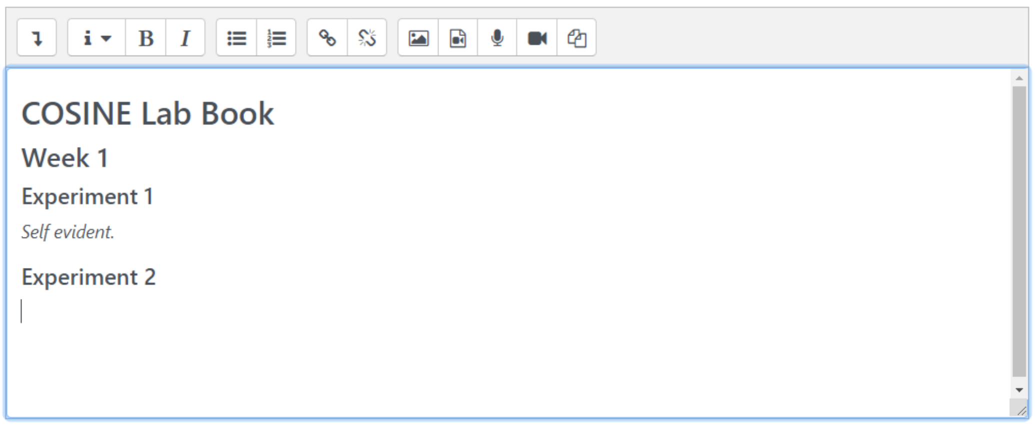
Experiment 1 - Start Wiki

"Create at least one section heading in such a way that it appears in the table of contents at the top of the Wiki page."

Self evident.

Experiment 2 - Upload Screenshot

"Create at least one screenshot of a window on your computer and insert it in your Wiki page (e.g. use the "Snipping Tool" in Windows accessories, save the image to a local file, then upload it to the Wiki)."



COSINE Lab Book
Week 1
Experiment 1
Self evident.
Experiment 2

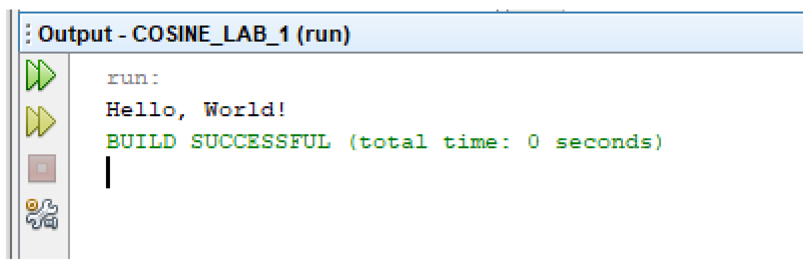
Experiment 3 - Attach File

"Upload and attach at least one file of some external type, e.g. Word, PowerPoint, etc, embedding a link to it in your Wiki."

Link to... '*Why's (Poignant) Guide to Ruby*'.

Experiment 4 - Run "Hello World" in Netbeans

Setting up of a project in Netbeans, to ensure a "Hello World" program can be run.



Output - COSINE_LAB_1 (run)

```
run:
Hello, World!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Reflective Question - Compare Netbeans

"Compare NetBeans with other Java deveopment environments you may have used in the past. What differences do you see?"

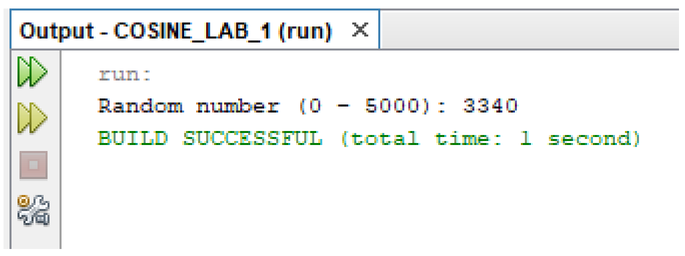
NetBeans takes longer to start than BlueJ, however NetBeans is equipped with far more tools that are likely to be helpful in developing more advanced programs. BlueJ takes a more visually driven approach to cater to new learners of Java, while Netbeans forgoes this in order to manage files in a way which more accurately represents a file system.

Wk 1 - Exercises

1. Method to generate random number.

```
public static void printRandNum()
   {
       Random rand = new Random();
       int  n = rand.nextInt(5000);
       System.out.println("Random number (0 - 5000): " + n);
   }
```

Output:



Output - COSINE_LAB_1 (run)

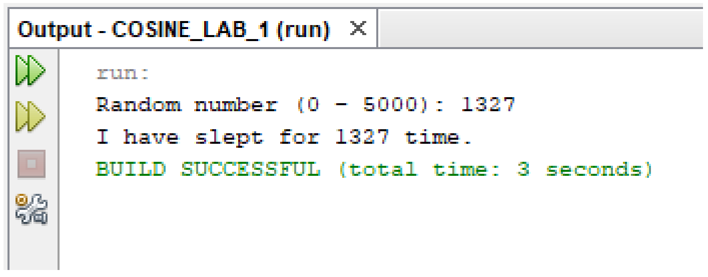
```
run:
Random number (0 - 5000): 3340
BUILD SUCCESSFUL (total time: 1 second)
```

2. Method to generate a random number and sleep for that amount of time.

```
public static void randWithSleep() throws InterruptedException
```

```
    {
       Random rand = new Random();
       int  n = rand.nextInt(5000);
       System.out.println("Random number (0 - 5000): " + n);
       Thread.sleep(n);
       System.out.println("I have slept for " + n + " time.");
   }
```

Output:

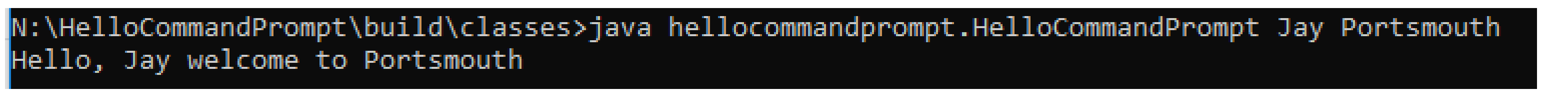


```
run:
Random number (0 - 5000): 1327
I have slept for 1327 time.
BUILD SUCCESSFUL (total time: 3 seconds)
```

3. Running a java class from the command line with arguments.

```
package hellocommandprompt;
/**
*
* @author up850844
*/
public class HelloCommandPrompt {
    /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {
       System.out.println("Hello, " + args[0] +" welcome to " + args[1]);
   }

}
```

Command line input and output:



```
N:\HelloCommandPrompt\build\classes>java hellocommandprompt.HelloCommandPrompt Jay Portsmouth
Hello, Jay welcome to Portsmouth
```

Week 2 - Objects and Fields In Java

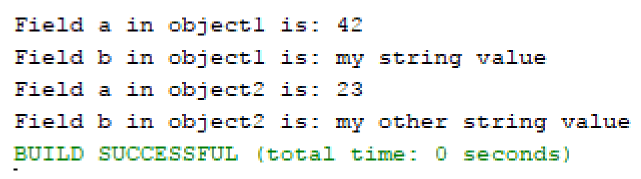
Experiment 1 - Create Two Objects and Print Their Field Data

Input code:

```
package wk2ex1;
```

```
public class Wk2ex1 {
    public static void main(String[] args) {
      MyClass object1 = new MyClass() ;
       // set a and b in object1
       object1.a = 42 ;
      object1.b = "my string value" ;
       MyClass object2 = new MyClass() ;
       // set a and b in object2
       object2.a = 23 ;
      object2.b = "my other string value" ;
       // print fields:
       System.out.println("Field a in object1 is: " + object1.a) ;
      System.out.println("Field b in object1 is: " + object1.b) ;
       System.out.println("Field a in object2 is: " + object2.a) ;
      System.out.println("Field b in object2 is: " + object2.b) ;
   }

}
class MyClass {
   int a;
   String b;
}
```

Output:



```
Field a in object1 is: 42
Field b in object1 is: my string value
Field a in object2 is: 23
Field b in object2 is: my other string value
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

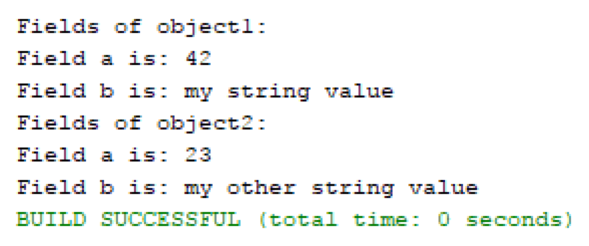
A class 'MyClass' is defined. Two objects of the type 'MyClass' are created. The objects have their field values assigned. The field values for the objects are printed with individual print statements.

Experiment 2 - Use method in Object Class to Print Field Data

Input code:

```
package wk2ex2;
public class Wk2ex2 {
    public static void main(String[] args) {
       MyNewClass object1 = new MyNewClass() ;
        object1.a = 42 ;
       object1.b = "my string value" ;
        MyNewClass object2 = new MyNewClass() ;
        object2.a = 23 ;
       object2.b = "my other string value" ;
        System.out.println("Fields of object1:") ;
       object1.printMyFields() ;
        System.out.println("Fields of object2:") ;
       object2.printMyFields() ;
   }

}
class MyNewClass {
   int a ;
   String b ;
    void printMyFields() {
        System.out.println("Field a is: " + a) ;
       System.out.println("Field b is: " + b) ;
   }
}
```

Output:



```
Fields of object1:
Field a is: 42
Field b is: my string value
Fields of object2:
Field a is: 23
Field b is: my other string value
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

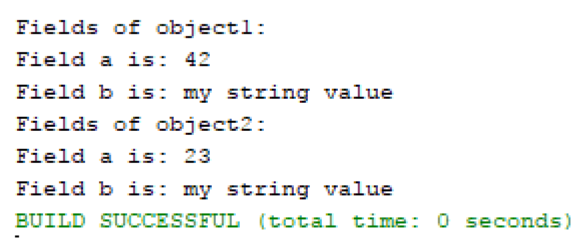
A class 'MyNewClass' is defined with instance variables and a method to print the field values. Two objects of the type 'MyNewClass' are created and their field values are set. The method 'printMyFields()' is used on each of the objects to print the field data for each object. The print statements in the main method are shorter than in the previous example.

Experiment 3 - Using Static Field Data

Input code:

```
package wk2ex3;
public class Wk2ex3 {

   public static void main(String[] args) {
       MyOtherClass.b = "my string value" ;
        MyOtherClass object1 = new MyOtherClass() ;
       object1.a = 42 ;
        MyOtherClass object2 = new MyOtherClass() ;
       object2.a = 23 ;
        System.out.println("Fields of object1:") ;
       object1.printMyFields() ;
        System.out.println("Fields of object2:") ;
       object2.printMyFields() ;
   }

}
class MyOtherClass {
   int a ;
   static String b ;
    void printMyFields() {
        System.out.println("Field a is: " + a) ;
       System.out.println("Field b is: " + b) ;
   }
}
```

Output:



```
Fields of object1:
Field a is: 42
Field b is: my string value
Fields of object2:
Field a is: 23
Field b is: my string value
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

A class 'MyOtherClass' is defined with an instance variable, 'a', and a static variable, 'b'. The static variable, once assigned, applies to all objects of the class. The static variable 'b' is assigned as "my string value", two objects of type 'MyOtherClass' are created, and the field data for both objects are printed using the 'printMyFields' method. The output demonstrates how the assignment of the static variable applies to both objects of the class.

Reflective Question - Considering Static and Instance Variables

If an instance method were to change the value 'a' for an object, the change would only apply to that object. This is because 'a' is an instance variable and is associated with the object created.

If an instance method were to change the value of 'b', the value for 'b' would be changed for all objects of the same type as 'b' is a static variable and is associated with the class rather than a specific object.

Wk 2 -Exercises

1. Write an instance method to change the instance variable for 'a'

Then call this method after the instance variables have been set, but before the 'printMtFields()' method is called.

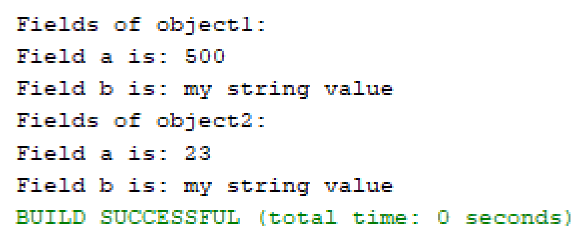
New instance method to change 'a':

```
void changeA(int newA) {
       a = newA;
}
```

Call to new method in main:

```
public static void main(String[] args) {
       MyOtherClass.b = "my string value" ;
        MyOtherClass object1 = new MyOtherClass() ;
       object1.a = 42 ;
        MyOtherClass object2 = new MyOtherClass() ;
       object2.a = 23 ;

       object1.changeA(500);

       System.out.println("Fields of object1:") ;
       object1.printMyFields() ;
        System.out.println("Fields of object2:") ;
       object2.printMyFields() ;
}
```

Console Output:



```
Fields of object1:
Field a is: 500
Field b is: my string value
Fields of object2:
Field a is: 23
Field b is: my string value
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Write an instance method to mutate the value of 'b'. Call this method in the main method.

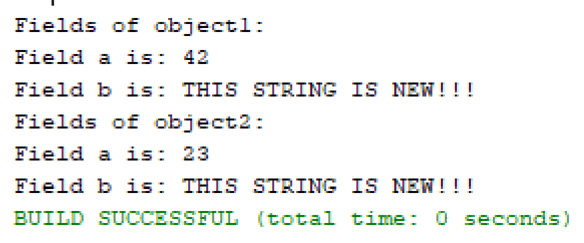
New instance method:

```
void changeB(String newB){
       b = newB;
}
```

Call to new method in main:

```
public static void main(String[] args) {
       MyOtherClass.b = "my string value" ;
        MyOtherClass object1 = new MyOtherClass() ;
       object1.a = 42 ;
        MyOtherClass object2 = new MyOtherClass() ;
       object2.a = 23 ;

       object1.changeB("THIS STRING IS NEW!!!");

       System.out.println("Fields of object1:") ;
       object1.printMyFields() ;
        System.out.println("Fields of object2:") ;
       object2.printMyFields() ;
}
```

Output:



```
Fields of object1:
Field a is: 42
Field b is: THIS STRING IS NEW!!!
Fields of object2:
Field a is: 23
Field b is: THIS STRING IS NEW!!!
BUILD SUCCESSFUL (total time: 0 seconds)
```

This confirms my expectations in the reflective question as by changing the static variable 'b' by calling the method on 'object1', the static variable 'b' for 'object2' is also changed.

Week 3 - Threads and Race Conditions

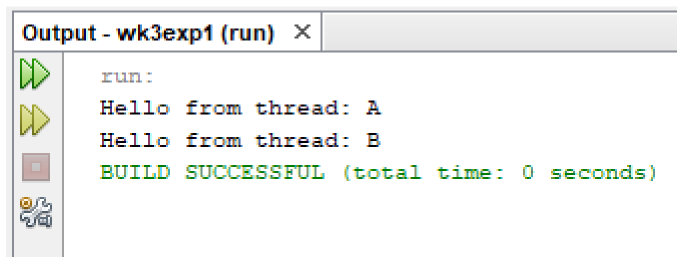
Note: a Race Condition is where two or more threads in a concurrent program try to update the same data structure.

Experiment 1 - Run Example Code

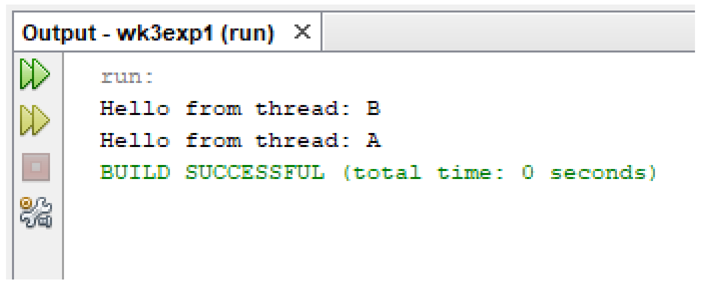
Input Code:

```
public class Wk3exp1 {
    public static void main(String[] args) throws Exception {
       MyThread thread1 = new MyThread() ;
       thread1.name = "A" ;
        MyThread thread2 = new MyThread() ;
       thread2.name = "B" ;
        thread1.start() ;
       thread2.start() ;
        thread2.join() ;
       thread1.join() ;
   }
}
class MyThread extends Thread {
   String name;

   public void run(){
       System.out.println("Hello from thread: " + name);
   }
}
```

Output:



```
Output - wk3exp1 (run)
run:
Hello from thread: A
Hello from thread: B
BUILD SUCCESSFUL (total time: 0 seconds)
```



```
Output - wk3exp1 (run)
run:
Hello from thread: B
Hello from thread: A
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

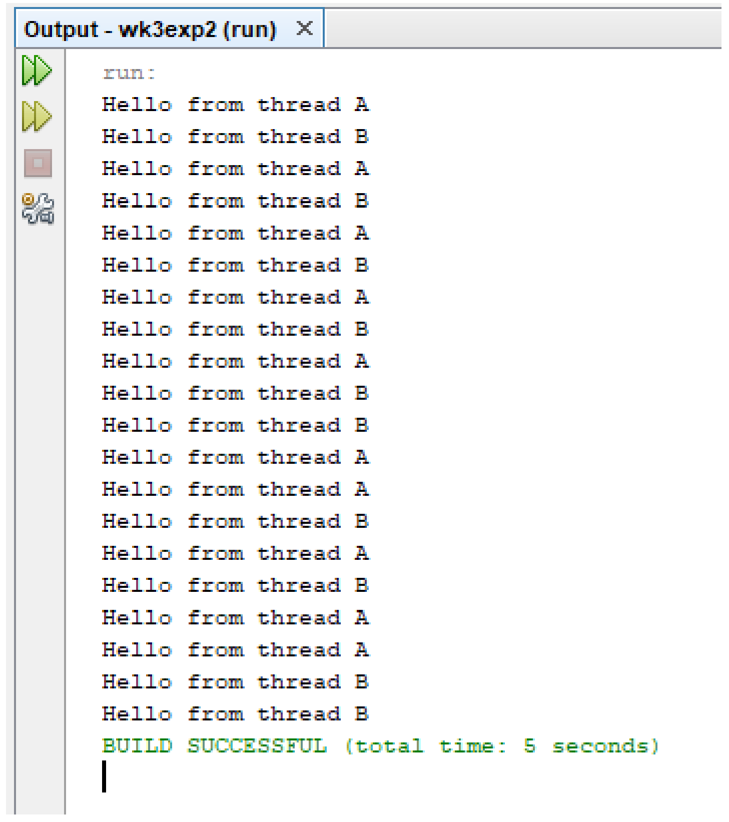
The class 'MyThread' extends the pre-existing class 'Thread'. The 'run' method is defined to print out a message. In the main method, two threads are created and their 'name' instance variables are set. Each thread is started (with the 'start' method), and then terminated (with the 'join' method) after the threads have ran.

Experiment 2 - Threads with loops and random delays

Input code:

```
public class Wk3exp2 {
    public static void main(String[] args) throws Exception {
      MyThread thread1 = new MyThread() ;
       thread1.name = "A" ;
        MyThread thread2 = new MyThread() ;
       thread2.name = "B" ;
        thread1.start() ;
       thread2.start() ;
        thread2.join() ;
       thread1.join() ;
   }

}
class MyThread extends Thread {
   String name ;
      public void run() {
          for(int i = 0 ; i < 10 ; i++) {
             delay() ;
             System.out.println("Hello from thread " + name) ;
         }
     }
      void delay() {
          int delay = (int) (1000 * Math.random()) ;
         try {
             Thread.sleep(delay) ;
         }
         catch(Exception e) {
         }
     }
}
```

Output:



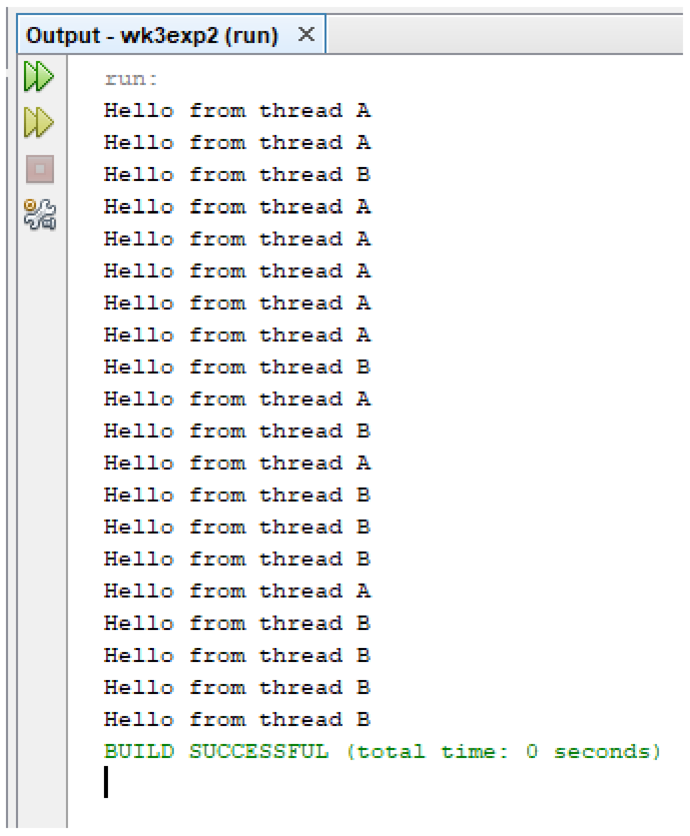
Output - wk3exp2 (run)

```
run:
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread B
Hello from thread A
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread A
Hello from thread B
Hello from thread B
BUILD SUCCESSFUL (total time: 5 seconds)
```

Explanation:

The second experiment works similarly to the first experiment, however, the run method prints ten times for each thread with a for loop that features a delay each iteration. This better illustrates the nature of threads as it shows how the printed messages do not follow a repeating pattern. If the calls to 'delay()' are removed, the threads run faster and the effects of concurrent threads are exaggerated as there is more interference.

Output without delay()



Output - wk3exp2 (run)

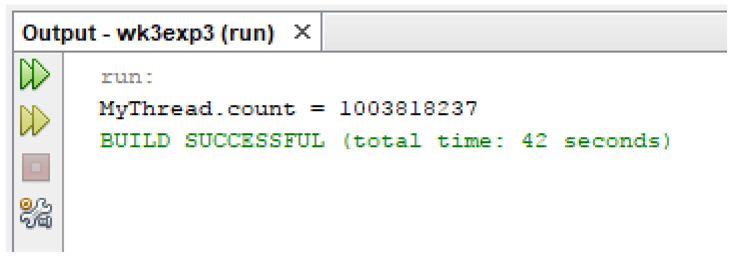
```
run:
Hello from thread A
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread A
Hello from thread A
Hello from thread A
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread B
Hello from thread B
Hello from thread A
Hello from thread B
Hello from thread B
Hello from thread B
Hello from thread B
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment 3 - Further Illustration of Race Conditions

Input code:

```
public class Wk3exp3 {
    public static void main(String[] args) throws Exception {
       MyThread.count = 0 ;
        MyThread thread1 = new MyThread();
       MyThread thread2 = new MyThread();
        thread1.start() ;
       thread2.start() ;
        thread2.join() ;
       thread1.join() ;
        System.out.println("MyThread.count = " + MyThread.count) ;
   }

}
class MyThread extends Thread {
    volatile static int count ;
    public void run() {
        for(int i = 0 ; i < 1000000000 ; i++) {
           int x = count ;
           count = x + 1 ;
     }
 }
}
```

Output:



```
run:
MyThread.count = 1003818237
BUILD SUCCESSFUL (total time: 42 seconds)
```

Explanation:

The output for the two threads maintaining the 'count' variable is significantly less than 2000000000. This is because both threads are trying to iterate the value until the count variable reaches 1000000000. This means that the value will never reach 2000000000, but due to thread interference, the counter will be incremented an incorrect amount of times. Thus producing the result: 1003818237.

Experiment 4 - Attempt to Resolve Race Condition with Unary Operation

Input code:

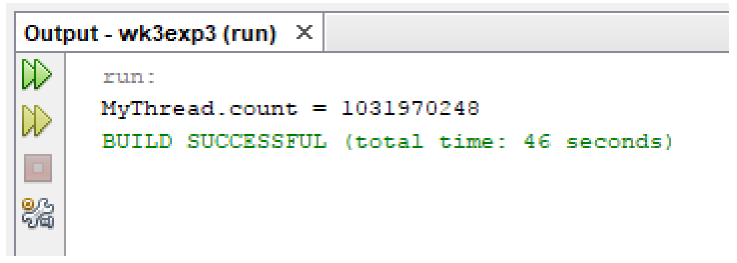
Same as from Experiment 3 but in MyThread class

```
int x = count ;
count = x + 1 ;
```

is replaced with

```
count++;
```

Output:



```
run:
MyThread.count = 1031970248
BUILD SUCCESSFUL (total time: 46 seconds)
```

Explanation:

This produces results similar to the results from Experiment 4 as the change in the code will not prevent interference between threads.

Reflective Question 1 - Atomicism of Java Statements

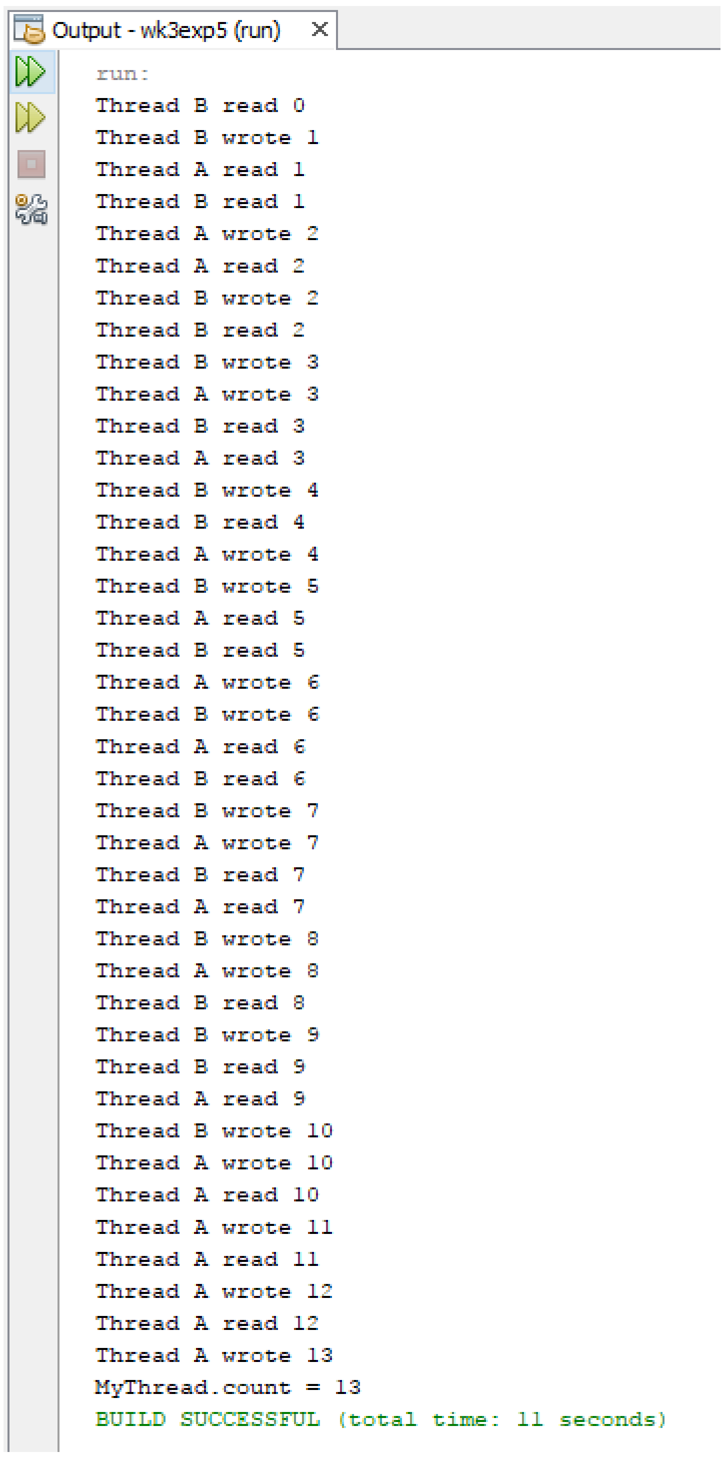
The change in experiment 4 does not resolve the Race Condition. This tells us that some statements in Java involving multiple threads are not totally atomic. This means that there is not the guarantee that operations within threads are isolated, allowing for interference between threads.

Experiment 5 - Investigating Race Conditions with Random Delays and Print Statements

Input code:

```
public class Wk3exp5 {
    public static void main(String[] args) throws Exception {
       MyThread.count = 0 ;
        MyThread thread1 = new MyThread() ;
       thread1.name = "A" ;
        MyThread thread2 = new MyThread() ;
       thread2.name = "B" ;
        thread1.start() ;
       thread2.start() ;
        thread2.join() ;
       thread1.join() ;
        System.out.println("MyThread.count = " + MyThread.count) ;
    }

}
class MyThread extends Thread {
    volatile static int count ;
    String name ;
    public void run() {
        for(int i = 0 ; i < 10 ; i++) {
           delay() ;
           int x = count ;
           System.out.println("Thread " + name + " read " + x) ;
           delay() ;
           count = x + 1 ;
           System.out.println("Thread " + name + " wrote " + (x + 1)) ;
       }
    }
    void delay() {
        int delay = (int) (1000 * Math.random()) ;
       try {
           Thread.sleep(delay) ;
       }
       catch(Exception e) {
       }
    }
}
```

Output:



```
run:
Thread B read 0
Thread B wrote 1
Thread A read 1
Thread B read 1
Thread A wrote 2
Thread A read 2
Thread B wrote 2
Thread B read 2
Thread B wrote 3
Thread A wrote 3
Thread B read 3
Thread A read 3
Thread B wrote 4
Thread B read 4
Thread A wrote 4
Thread B wrote 5
Thread A read 5
Thread B read 5
Thread A wrote 6
Thread B wrote 6
Thread A read 6
Thread B read 6
Thread B wrote 7
Thread A wrote 7
Thread B read 7
Thread A read 7
Thread B wrote 8
Thread A wrote 8
Thread B read 8
Thread B wrote 9
Thread B read 9
Thread A read 9
Thread B wrote 10
Thread A wrote 10
Thread A read 10
Thread A wrote 11
Thread A read 11
Thread A wrote 12
Thread A read 12
Thread A wrote 13
MyThread.count = 13
BUILD SUCCESSFUL (total time: 11 seconds)
```

Explanation:

In this experiment, similar to the previous experiment, the counter does not reach the estimated value of 20, due to the race condition. Both threads try to increment the 'counter' value 10 times, but fails to due to interference, despite the use of both a random delay and print statements.

Reflective Question 2 - Random Delays in Sequential Programs

Random delays in a sequential program would never effect the correctness of behaviour. This is because the random delay will not interfere with how the rest of the program runs as sequential programs cannot exhibit race conditions.

Wk 3 - Exercises

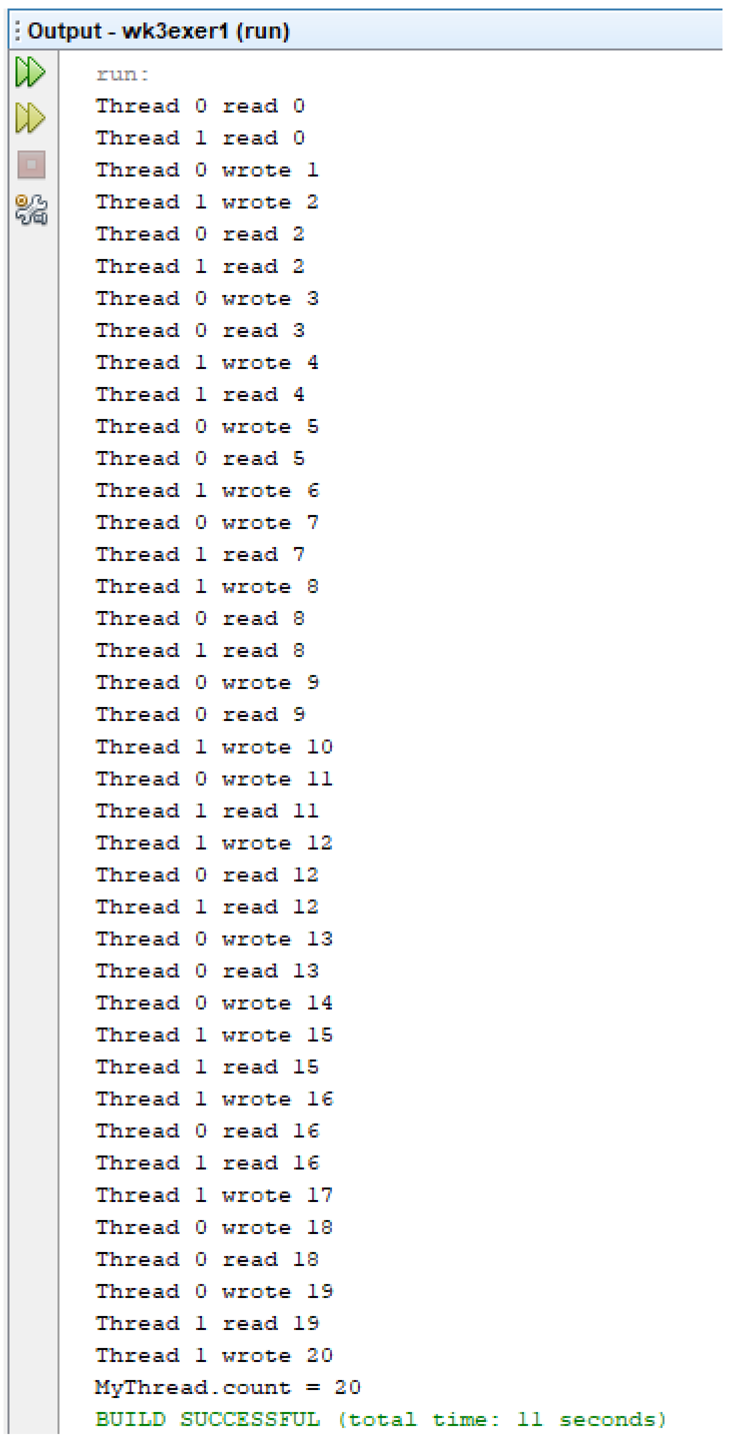
1. Race Condition Elimination via Peterson's Algorithm

The code for the Main method is largely the same as in Experiment 5 but uses the setThreadID() method that I wrote for the MyThread class to keep the thread ID instance variables private. Through using the protections provided by implementing Peterson's Algorithm, the Race Condition is able to be eliminated, so that a count value of 20 can be achieved.

Modified MyThread Class:

```
class MyThread extends Thread {
    volatile static int count;
   private int threadID;

   //peterson variables
   volatile static boolean flag[] = {false, false};
   volatile static int turn;
    public void run() {
        for(int i = 0 ; i < 10 ; i++) {
           delay() ;

           //critical preamble
           flag[threadID] = true;

           int otherTID;
           if(threadID == 0){
               otherTID = 1;
           }else{
               otherTID = 0;
           }

           turn = otherTID;

           while(flag[otherTID] && (turn == otherTID)) {}

           //critical section start
           int x = count ;
           System.out.println("Thread " + Integer.toString(threadID) + " read " + x);
           delay();
           count = x + 1;
           System.out.println("Thread " + Integer.toString(threadID) + " wrote " + count);
           //critical section end

           //critical postamble
           flag[threadID] = false;

       }
   }
    void delay() {
        int delay = (int) (1000 * Math.random()) ;
       try {
           Thread.sleep(delay) ;
       }
       catch(Exception e) {
       }
   }

   public void setThreadID(int id){
       threadID = id;
   }
}
```

Output:


Output - wk3exer1 (run)

```
run:
Thread 0 read 0
Thread 1 read 0
Thread 0 wrote 1
Thread 1 wrote 2
Thread 0 read 2
Thread 1 read 2
Thread 0 wrote 3
Thread 0 read 3
Thread 1 wrote 4
Thread 1 read 4
Thread 0 wrote 5
Thread 0 read 5
Thread 1 wrote 6
Thread 0 wrote 7
Thread 1 read 7
Thread 1 wrote 8
Thread 0 read 8
Thread 1 read 8
Thread 0 wrote 9
Thread 0 read 9
Thread 1 wrote 10
Thread 0 wrote 11
Thread 1 read 11
Thread 1 wrote 12
Thread 0 read 12
Thread 1 read 12
Thread 0 wrote 13
Thread 0 read 13
Thread 0 wrote 14
Thread 1 wrote 15
Thread 1 read 15
Thread 1 wrote 16
Thread 0 read 16
Thread 1 read 16
Thread 1 wrote 17
Thread 0 wrote 18
Thread 0 read 18
Thread 0 wrote 19
Thread 1 read 19
Thread 1 wrote 20
MyThread.count = 20
BUILD SUCCESSFUL (total time: 11 seconds)
```

Explanation:

The Main method creates two objects of class MyThread, and sets the static MyTread variable variable 'count' to 0. Each thread increments the static variable 10 times, thus - due to mutual exclusion protections - the 'count' variable ends with a value of 20.

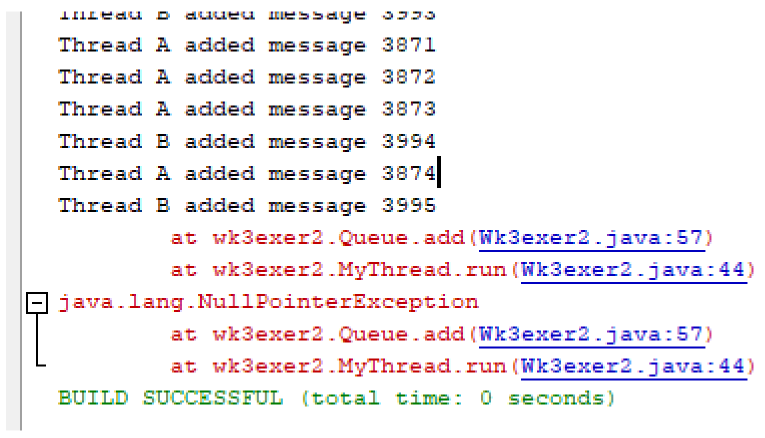
2. Inspect and run codes

The code creates two threads, each of which attempts to add 1,000,00 messages to the static object 'q' of the class 'Queue'. The 'Queue' class instances a static 'Node' object which, in tandem with the 'Queue' object, maintains the index of the first and last values of the queue.

The failure mode is where the threads interfere such that the Node values point to indexes in the queue that do not exist, producing a

```
java.lang.NullPointerException
```

error.



```
Thread B added message 3993
Thread A added message 3871
Thread A added message 3872
Thread A added message 3873
Thread B added message 3994
Thread A added message 3874
Thread B added message 3995
        at wk3exer2.Queue.add(Wk3exer2.java:57)
        at wk3exer2.MyThread.run(Wk3exer2.java:44)
java.lang.NullPointerException
        at wk3exer2.Queue.add(Wk3exer2.java:57)
        at wk3exer2.MyThread.run(Wk3exer2.java:44)
BUILD SUCCESSFUL (total time: 0 seconds)
```

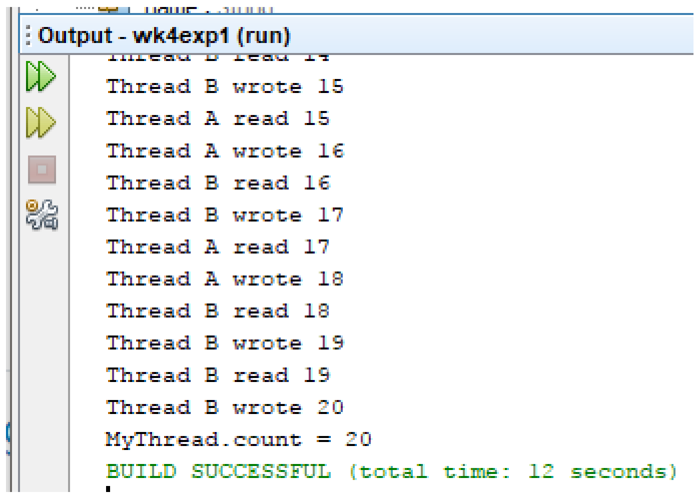
Week 4 - Mutual Exclusion

Experiment 1 - Possible Mutual Exclusion Implementation

Input code:

```
package wk4exp1;
public class Wk4exp1 {
    public static void main(String[] args) throws Exception {
       MyThread.count = 0 ;
        MyThread thread1 = new MyThread() ;
       thread1.name = "A" ;
        MyThread thread2 = new MyThread() ;
       thread2.name = "B" ;
        thread1.start() ;
       thread2.start() ;
        thread2.join() ;
       thread1.join() ;
        System.out.println("MyThread.count = " + MyThread.count) ;
   }

}
class MyThread extends Thread {
    volatile static int count ;
    volatile static boolean lock = false ;
    String name ;
    public void run() {
        for(int i = 0 ; i < 10 ; i++) {
           delay() ;
            while(lock) {}  // wait until lock is false
           lock = true ;   // claim access to critical region
            // start of critical section
           int x = count ;
           System.out.println("Thread " + name + " read " + x) ;
           delay() ;
           count = x + 1 ;
           System.out.println("Thread " + name + " wrote " + (x + 1)) ;
           // end of critical section
            lock = false ;  // release access to critical region
       }
   }
    void delay() {
        int delay = (int) (1000 * Math.random()) ;
       try {
           Thread.sleep(delay) ;
       }
       catch(Exception e) {
       }
   }
}
```

Output: reaches a count value of 20.



Output - wk4exp1 (run)

```
Thread B wrote 15
Thread A read 15
Thread A wrote 16
Thread B read 16
Thread B wrote 17
Thread A read 17
Thread A wrote 18
Thread B read 18
Thread B wrote 19
Thread B read 19
Thread B wrote 20
MyThread.count = 20
BUILD SUCCESSFUL (total time: 12 seconds)
```

Explanation:

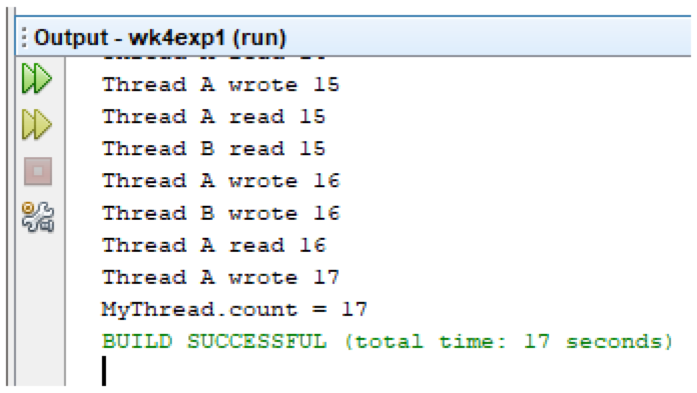
Despite reaching a count value of 20, the Race Condition has not been resolved, and Mutual Exclusion has not been achieved. This is because it still allows for both threads to enter their critical sections at the same time. One thread can test for the 'lock' value in between the setting of the 'lock' value and the end of the while wait in the other thread. Therefore safety is not guaranteed.

Experiment 2 - Boolean Lock With Random Delay

Input code:

Same as previous experiment except with a call to 'delay()' in between the while wait and setting of the 'lock' value as true in the MyThread class.

Output:



Output - wk4exp1 (run)

```
Thread A wrote 15
Thread A read 15
Thread B read 15
Thread A wrote 16
Thread B wrote 16
Thread A read 16
Thread A wrote 17
MyThread.count = 17
BUILD SUCCESSFUL (total time: 17 seconds)
```

Explanation:

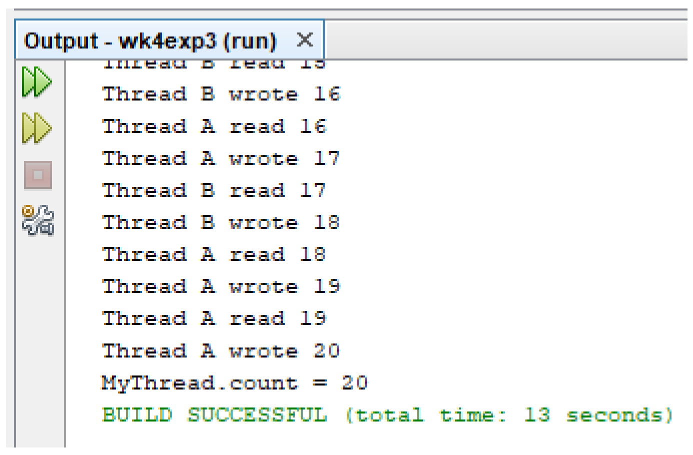
The call to delay exaggerates the theoretical mutual accessing of critical sections by both threads as there is a longer time between the wait loop and the setting of 'lock'. This is demonstrated by the fact that the count value only reaches 17.

Experiment 3 - Run Given Peterson's Algorithm Implementation

Input code:

```
ackage wk4exp3;
public class Wk4exp3 {
    public static void main(String[] args) throws Exception {
       MyThread.count = 0 ;
        MyThread thread1 = new MyThread() ;
       thread1.name = "A" ;
       thread1.id = 0 ;
        MyThread thread2 = new MyThread() ;
       thread2.name = "B" ;
       thread2.id = 1 ;
        thread1.start() ;
       thread2.start() ;
        thread1.join() ;
       thread2.join() ;
        System.out.println("MyThread.count = " + MyThread.count) ;
   }

}
class MyThread extends Thread {
    volatile static int count ;
    String name ;
   int id ;
    public void run() {
        for(int i = 0 ; i < 10 ; i++) {
           delay() ;
            beginCriticalSection() ;
            int x = count ;
           System.out.println("Thread " + name + " read " + x) ;
           delay() ;
           count = x + 1 ;
           System.out.println("Thread " + name + " wrote " + (x + 1)) ;
            endCriticalSection() ;
       }
   }
    // Peterson's algorithm
    volatile static int turn ;
   volatile static boolean [] interested = new boolean [2] ;
    void beginCriticalSection() {
       interested [id] = true ;
       int jd = 1 - id ;
       turn = jd ;
       while(interested [jd] & turn == jd) {}
   }
    void endCriticalSection() {
       interested [id] = false ;
   }
    void delay() {
        int delay = (int) (1000 * Math.random()) ;
       try {
           Thread.sleep(delay) ;
       }
       catch(Exception e) {
       }
   }
}
```

Output:


```
Thread B read 15
Thread B wrote 16
Thread A read 16
Thread A wrote 17
Thread B read 17
Thread B wrote 18
Thread A read 18
Thread A wrote 19
Thread A read 19
Thread A wrote 20
MyThread.count = 20
BUILD SUCCESSFUL (total time: 13 seconds)
```

Explanation:

The code in this experiment implements Peterson's Algorithm, similarly to the implementation from Exercise 1 in Week 3, however this implementation separates the Peterson pre/postamble into their own methods.

Experiment 4 - Semaphore Implementation

Note: to use the native Java Semaphores, the concurrency package must be declared with

```
import java.util.concurrent.*;
```

and Semaphores can be instantiated with

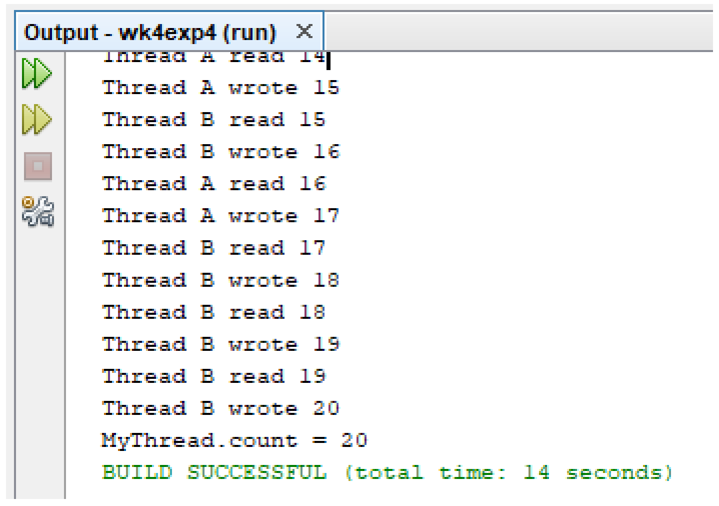
```
Semaphore mySemaphore = new Semaphore(N);
```

where N is the initial value.

Input code same as in the previous 'slow race' examples, but the critical section is protected with Semaphore operations, as such:

```
mySemaphore.acquire() ;  // P - decrease semaphore
int x = count ;
System.out.println("Thread " + name + " read " + x) ;
delay() ;
count = x + 1 ;
System.out.println("Thread " + name + " wrote " + (x + 1)) ;
mySemaphore.release() ;  // V - increase semaphore
```

Output:


Output - wk4exp4 (run)

```
Thread A read 14
Thread A wrote 15
Thread B read 15
Thread B wrote 16
Thread A read 16
Thread A wrote 17
Thread B read 17
Thread B wrote 18
Thread B read 18
Thread B wrote 19
Thread B read 19
Thread B wrote 20
MyThread.count = 20
BUILD SUCCESSFUL (total time: 14 seconds)
```

Explanation:

Through using the semaphores, we are able to implement mutual exclusion. The 'acquire' method decreases the semaphore on entry to the critical section, preventing other threads from accessing their critical sections, and the 'release' method increases the semaphore after the execution of the critical section, enabling another thread to enter its critical sections.

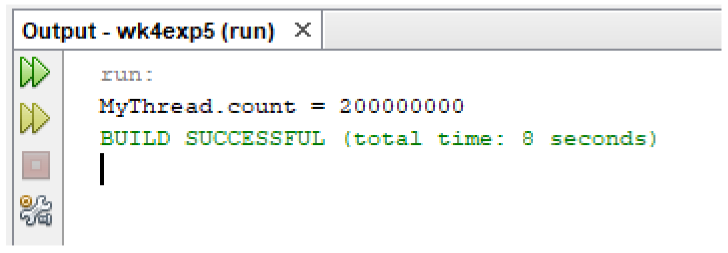
Experiment 5 - Using Synchronised Methods

Native implementation of mutual exclusion in Java easily used through using the 'synchronized' method modifier, which prevents simultaneous access to objects via methods. The same can be applied to static methods.

Input Code:

```
package wk4exp5;
public class Wk4exp5 {
   public static void main(String[] args) throws Exception{
       MyThread.count = 0 ;
       MyThread thread1 = new MyThread();
       MyThread thread2 = new MyThread();
       thread1.start() ;
       thread2.start() ;
       thread2.join() ;
       thread1.join() ;
       System.out.println("MyThread.count = " + MyThread.count) ;
 }
}
class MyThread extends Thread {
   static volatile int count ;
   String name ;
    public void run() {
       for(int i = 0 ; i < 100000000 ; i++) {
           increment() ;
       }
   }
   synchronized static void increment() {
       int x = count ;
       count = x + 1 ;
   }
}
```

Output:


Output - wk4exp5 (run)

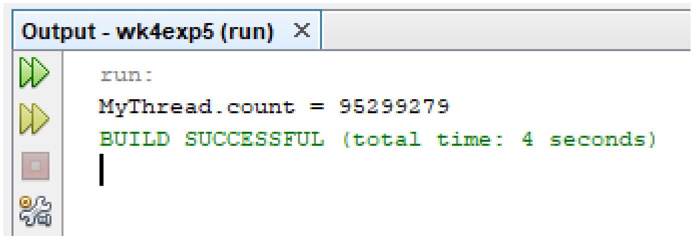
```
run:
MyThread.count = 200000000
BUILD SUCCESSFUL (total time: 8 seconds)
```

Explanation:

The 'sychronized' method modifier applied to the critical section, which is pulled out into its own method, implements mutual exclusion, as no two threads can are able to run 'increment()' at the same time, producing a count value of 200,000,000.

Experiment 6 - Testing Previous, without 'synchronized' Modifier

Output:


Output - wk4exp5 (run)

```
run:
MyThread.count = 95299279
BUILD SUCCESSFUL (total time: 4 seconds)
```

Explanation:

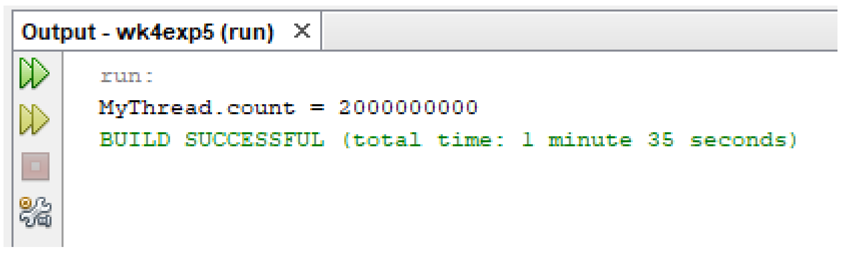
Removing the 'synchronized' modifier made the 'increment()' method not mutually exclusive with allowed for interference, thus producing a result of 95,299,279.

Reflective Questions - Effect of Synchronised Methods On Behaviour

The unsychronised version of the method had a positive effect on overall run-time, in that, with the 'sychronized' modifier, the program took half as long to run. However, without the 'sychronized' modifier we gain an inaccurate result.

The loop count was reduced by a factor 10 as with 1,000,000,000 iterations per loop, the program would take much longer to run and would be impractical to test.

Output with loop count as 1,000,000,000:


Output - wk4exp5 (run)

```
run:
MyThread.count = 2000000000
BUILD SUCCESSFUL (total time: 1 minute 35 seconds)
```

Wk 4 -Exercises

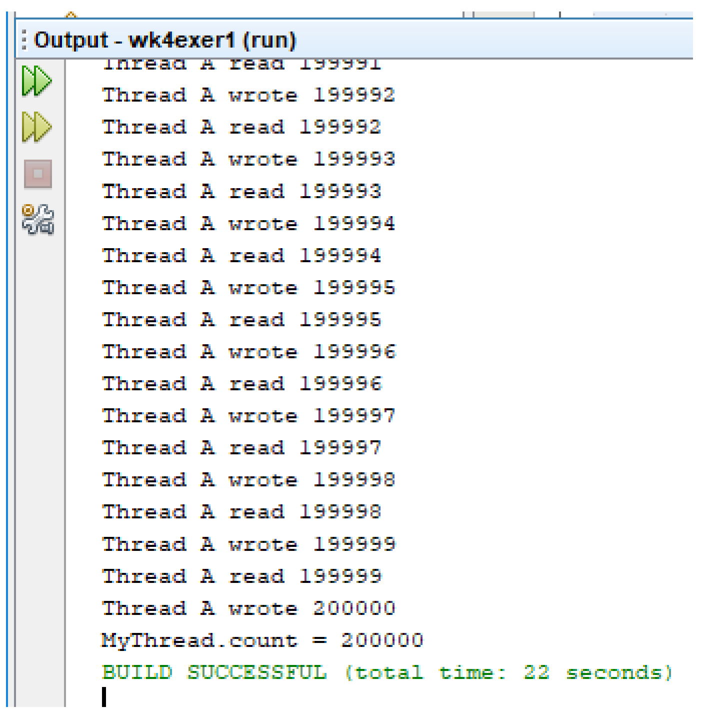
1. Attempt to Break Peterson's Algorithm Implementation

Despite adding random delays in many places throughout the code and increasing the number of count updates to 100,000 time per thread, I have not been able to break this implementation of Peterson's Algorithm.

Input MyThread Class:

```
class MyThread extends Thread {
   volatile static int count ;
   String name ;
   int id ;
   public void run() {
       for(int i = 0 ; i < 100000 ; i++) {
           delay();
           beginCriticalSection() ;
           delay();
           int x = count ;
           delay();
           System.out.println("Thread " + name + " read " + x) ;
           delay();
           count = x + 1 ;
           delay();
           System.out.println("Thread " + name + " wrote " + (x + 1)) ;
           delay();
           endCriticalSection() ;
           delay();
       }
   }
   volatile static int turn ;
   volatile static boolean [] interested = new boolean [2] ;
   void beginCriticalSection() {
       delay();
       interested [id] = true ;
       delay();
       int jd = 1 - id ;
       delay();
       turn = jd ;
       delay();
       while(interested [jd] & turn == jd) {}
       delay();
   }
   void endCriticalSection() {
       delay();
       interested [id] = false ;
       delay();
   }
   void delay() {
     int delay = (int) (1 * Math.random()) ;
     try {
         Thread.sleep(delay) ;
     }
     catch(Exception e) {}
   }
}
```

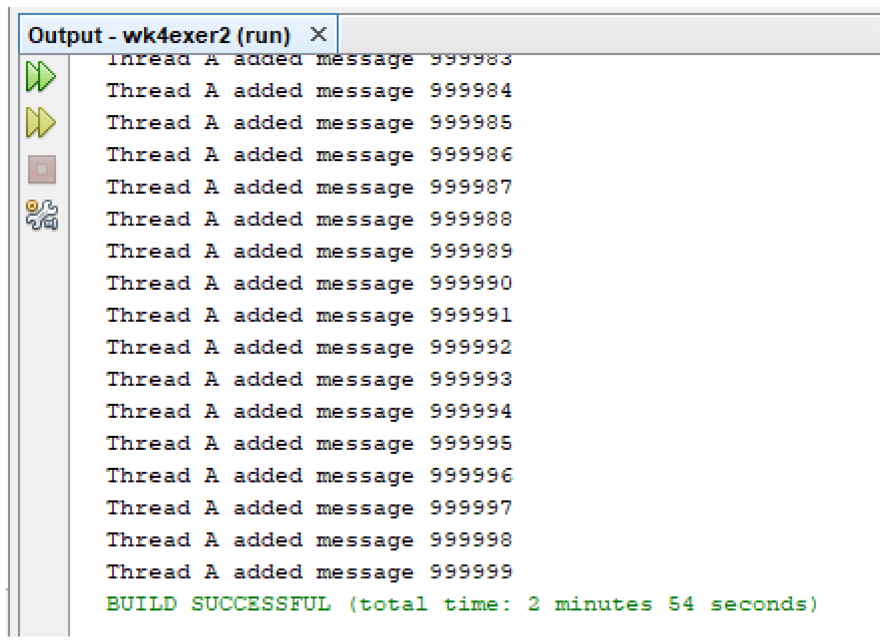
Output:


Output - wk4exer1 (run)

```
Thread A read 199991
Thread A wrote 199992
Thread A read 199992
Thread A wrote 199993
Thread A read 199993
Thread A wrote 199994
Thread A read 199994
Thread A wrote 199995
Thread A read 199995
Thread A wrote 199996
Thread A read 199996
Thread A wrote 199997
Thread A read 199997
Thread A wrote 199998
Thread A read 199998
Thread A wrote 199999
Thread A read 199999
Thread A wrote 200000
MyThread.count = 200000
BUILD SUCCESSFUL (total time: 22 seconds)
```

2. Fix Race Condition from Last Week Exercise

The race condition from the last exercise from last week was easily rectified through changing the method modifier for the 'add()' method in the Queue class to 'synchronized'.


Output - wk4exer2 (run)

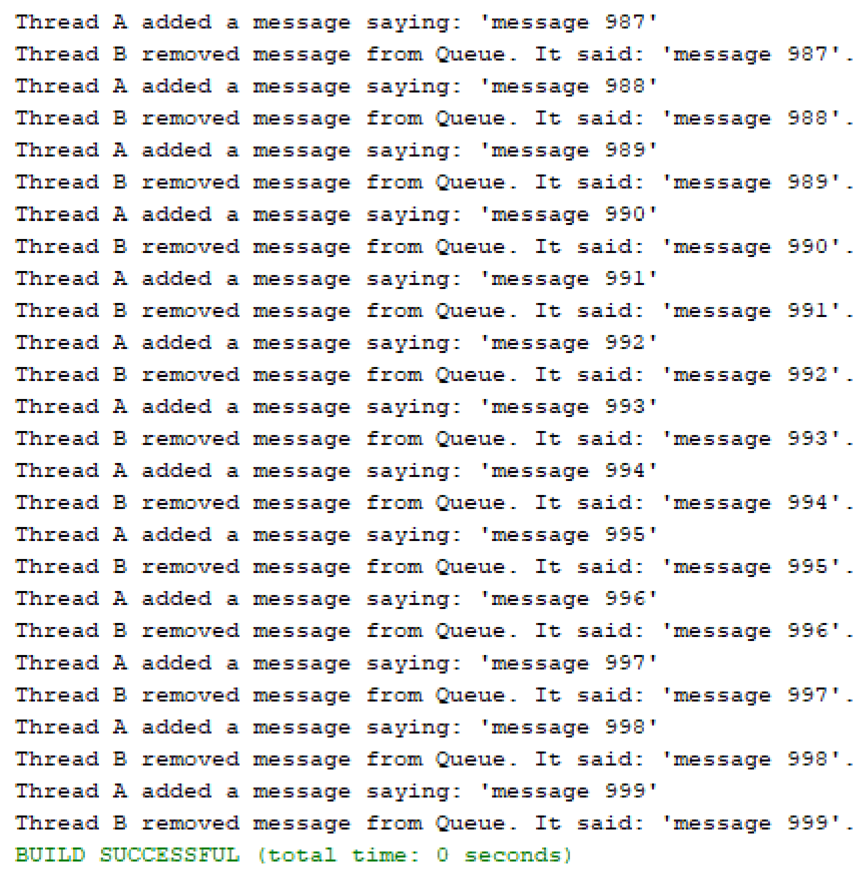
```
Thread A added message 999983
Thread A added message 999984
Thread A added message 999985
Thread A added message 999986
Thread A added message 999987
Thread A added message 999988
Thread A added message 999989
Thread A added message 999990
Thread A added message 999991
Thread A added message 999992
Thread A added message 999993
Thread A added message 999994
Thread A added message 999995
Thread A added message 999996
Thread A added message 999997
Thread A added message 999998
Thread A added message 999999
BUILD SUCCESSFUL (total time: 2 minutes 54 seconds)
```

3. Use Distinct Threads to Add to and Remove From a Queue //Partial Answer

Input Code:

```
package wk4exer3;
public class Wk4exer3 {
    public static void main(String[] args) throws Exception{
       MyThread.q = new Queue() ;
        MyThread thread1 = new MyThread() ;
       thread1.name = "A" ;
       thread1.id = 0;
        MyThread removal = new RemovalThread() ;
       removal.name = "B" ;
       removal.id = 1;
        thread1.start() ;
       removal.start() ;
        removal.join() ;
       thread1.join() ;
   }

}
class MyThread extends Thread {
    volatile static Queue q ;
   String name ;
   int id;

   volatile static int turn;
   volatile static boolean [] interested = new boolean [2];
    public void run() {
        for(int i = 0 ; i < 1000 ; i++) {
           beginCritSec();
           String mess = "message " + i + "";
           q.add(mess) ;
           System.out.println("Thread " + name + " added a message saying: '" + mess + "'") ;
           endCritSec();
       }
   }

   void beginCritSec(){
      interested [id] = true ;
      int jd = 1 - id ;
      turn = jd ;
      while(interested [jd] & turn == jd) {}
   }
   void endCritSec(){
       interested[id] = false;
   }
}
class RemovalThread extends MyThread {

   @Override
   public void run() {

       for (int i = 0; i < 1000; i++){
           beginCritSec();
           Node removedNode = q.remove();
           if (removedNode != null){
               System.out.println("Thread " + name + " removed message from Queue. It said: '" + removedNode.data + "'.");
           }else{
               System.out.println("Message cannot be popped.");
           }
           endCritSec();
       }
   }
}
class Queue {
    volatile Node front, back ;
    synchronized void add(String data) {
       if (front != null) {
           Node newNode = new Node(data);
           newNode.next = back.next;
           back = newNode;
       }
       else {
           front = new Node(data) ;
           back = front ;
       }
   }

   synchronized Node remove(){
       Node popped = null;
       if (rem() != null){
           popped = back;
           back = back.next;
       }
       return popped;


   }
    public String rem() {
       // returns null if queue empty
       String result = null ;
       if (front != null)  {
         result = front.data ;
         front = front.next ;
       }
       return result ;
   }
}
class Node {
   Node(String data) {
     this.data = data ;
   }
    String data ;
   Node next ;
}
```

Output:



```
Thread A added a message saying: 'message 987'
Thread B removed message from Queue. It said: 'message 987'.
Thread A added a message saying: 'message 988'
Thread B removed message from Queue. It said: 'message 988'.
Thread A added a message saying: 'message 989'
Thread B removed message from Queue. It said: 'message 989'.
Thread A added a message saying: 'message 990'
Thread B removed message from Queue. It said: 'message 990'.
Thread A added a message saying: 'message 991'
Thread B removed message from Queue. It said: 'message 991'.
Thread A added a message saying: 'message 992'
Thread B removed message from Queue. It said: 'message 992'.
Thread A added a message saying: 'message 993'
Thread B removed message from Queue. It said: 'message 993'.
Thread A added a message saying: 'message 994'
Thread B removed message from Queue. It said: 'message 994'.
Thread A added a message saying: 'message 995'
Thread B removed message from Queue. It said: 'message 995'.
Thread A added a message saying: 'message 996'
Thread B removed message from Queue. It said: 'message 996'.
Thread A added a message saying: 'message 997'
Thread B removed message from Queue. It said: 'message 997'.
Thread A added a message saying: 'message 998'
Thread B removed message from Queue. It said: 'message 998'.
Thread A added a message saying: 'message 999'
Thread B removed message from Queue. It said: 'message 999'.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

The output shows the program working to a large extent. For simplicity, I operated on the Queue like a stack by adding and removing from the back. Through combing techniques of using volatile variables, synchronised methods and an implementation of Peterson's algorithm. More work should be done with this so that is operates more like a queue.

The shortcoming in sending and receiving messages between two threads is that one operation (send/receive) must be completed before the other can begin.

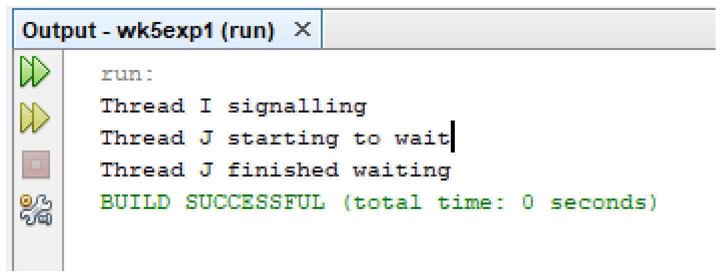
Week 5 - Synchronisation and Deadlock

Experiment 1 - Run Program to Illustrate Thread Notification Thread Synchronisation.

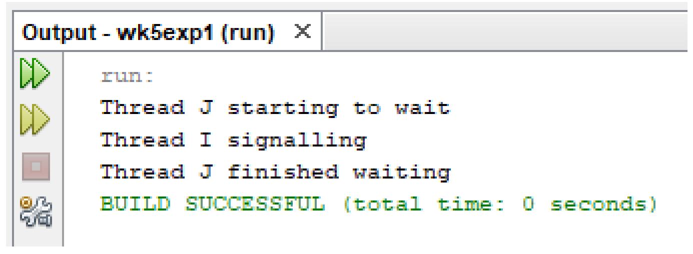
Input Code:

```
package wk5exp1;
import java.util.concurrent.*;
public class Wk5exp1 {

   public static void main(String[] args) throws Exception {
       MyThreadI thread1 = new MyThreadI() ;
        MyThreadJ thread2 = new MyThreadJ() ;
        thread1.start() ;
       thread2.start() ;
        thread2.join() ;
       thread1.join() ;
   }

}
class MyThreadI extends Thread {
    static Semaphore flag = new Semaphore(0);  // initialize to 0
    void delay() {
        int delay = (int) (1000 * Math.random()) ;
       try {
           Thread.sleep(delay) ;
       }
       catch(Exception e) {
       }
   }
   public void run() {
       try {
           delay() ;
            System.out.println("Thread I signalling") ;
            flag.release() ;  // V(flag)
       }
       catch(Exception e) {}
   }
}
class MyThreadJ extends MyThreadI {

   public void run() {
       try {
         delay() ;
          System.out.println("Thread J starting to wait") ;
          flag.acquire() ;  // P(flag)
          System.out.println("Thread J finished waiting") ;
       }
       catch(Exception e) {}
   }
}
```

Outputs:


Output - wk5exp1 (run)

```
run:
Thread I signalling
Thread J starting to wait
Thread J finished waiting
BUILD SUCCESSFUL (total time: 0 seconds)
```


Output - wk5exp1 (run)

```
run:
Thread J starting to wait
Thread I signalling
Thread J finished waiting
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

The program creates two threads: 'I' and 'J'. As both threads access the same semaphore, such that Thread J is not allowed to finish waiting until Thread I has started signalling. This is because the run method for Thread J acquires the semaphore before Thread J can finish, therefore, thread J must wait until Thread I's run method releases the semaphore so that Thread J can finish waiting.

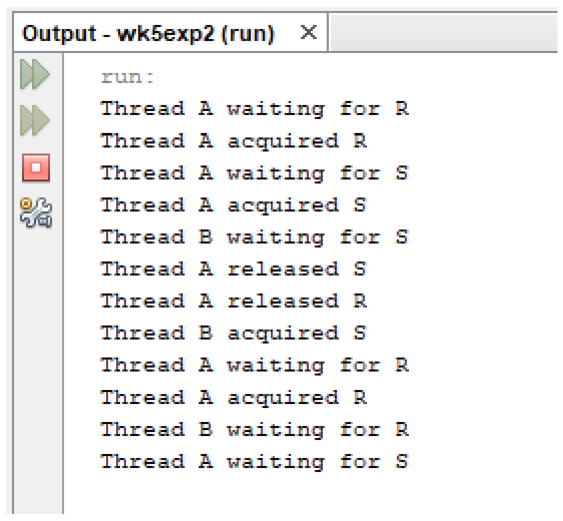
Experiment 2 - Run Program to Illustrate Deadlock

Input code:

```
package wk5exp2;
import java.util.concurrent.*;
public class Wk5exp2 {
   public static void main(String[] args) throws Exception{

       MyThreadA thread1 = new MyThreadA();
        MyThreadB thread2 = new MyThreadB();
        thread1.start();
       thread2.start();
        thread2.join();
       thread1.join();

   }
}
class MyThreadA extends Thread {
    static Semaphore semR = new Semaphore(1);
    static Semaphore semS = new Semaphore(1);
    void delay() {
        int delay = (int) (1000 * Math.random());
       try {
           Thread.sleep(delay);
       }
       catch(Exception e) {
       }
   }
   public void run() {
       try {
           while(true) {
               delay();
                System.out.println("Thread A waiting for R");
               semR.acquire();
               System.out.println("Thread A acquired R");
                delay() ;
                System.out.println("Thread A waiting for S") ;
               semS.acquire() ;
               System.out.println("Thread A acquired S") ;
                delay() ;
                semS.release() ;
               System.out.println("Thread A released S") ;
                semR.release() ;
               System.out.println("Thread A released R") ;
           }
       }
       catch(Exception e) {}
   }
}
class MyThreadB extends MyThreadA {
   public void run() {
       try {
           while(true) {
               delay() ;
                System.out.println("Thread B waiting for S") ;
               semS.acquire() ;
               System.out.println("Thread B acquired S") ;
                delay() ;
                System.out.println("Thread B waiting for R") ;
               semR.acquire() ;
               System.out.println("Thread B acquired R") ;
                delay() ;
                semR.release() ;
               System.out.println("Thread B released R") ;
                semS.release() ;
               System.out.println("Thread B released S") ;
           }
       }
       catch(Exception e) {}
   }
}
```

Output:

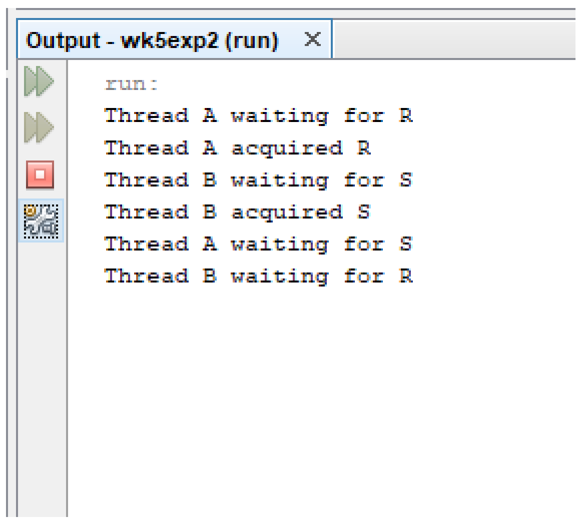
Example 1:



Output - wk5exp2 (run)

```
run:
Thread A waiting for R
Thread A acquired R
Thread A waiting for S
Thread A acquired S
Thread B waiting for S
Thread A released S
Thread A released R
Thread B acquired S
Thread A waiting for R
Thread A acquired R
Thread B waiting for R
Thread A waiting for S
```

Example 2:



Output - wk5exp2 (run)

```
run:
Thread A waiting for R
Thread A acquired R
Thread B waiting for S
Thread B acquired S
Thread A waiting for S
Thread B waiting for R
```

Explanation:

The experiment here demonstrates how a deadlock can be achieved. This is illustrated by the fact that the program becomes idle when each thread has acquired one resource causing both threads to wait infinitely as neither resource can become available again. The resources are represented by the semaphores as it restricts the access to one thread at a time. This is why we can get different outputs when running the program as the concurrent nature of the program will allow for deadlock to occur after a varied number of operations.

Reflective Question - Compare Race Conditions and Deadlocks

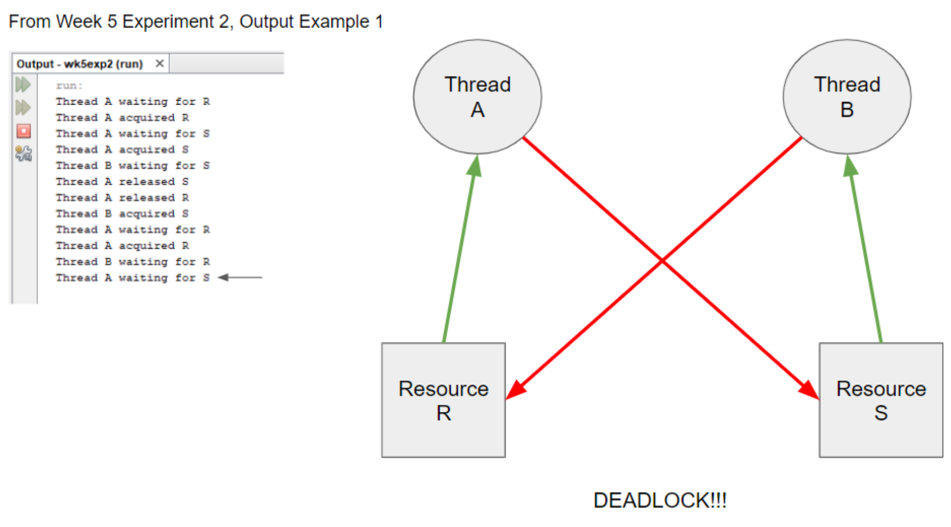
Both Race Conditions and Deadlocks are similar in that they have to ability to disrupt the intended function of a concurrently running program due to the issues that can arise when attempting to run a program in such a way that both data and operational integrity is preserved.

Deadlock does this by a logical flaw in that threads can request access to resources such that any resources needed for any thread to complete operation are occupied. The issues raised by deadlock occurrences can be mitigated by looking for cycles within a Resource Allocation Graph to illustrate where deadlock can occur. When deadlock occurs, however, it can be remedied by using periodic detection algorithms and a recovery scheme to resume normal operation. One way in which a recovery scheme could work is by killing processes and then allocating resources to the surviving processes.

Race conditions differ in this regard as unexpected behaviour comes from the shared data structures between threads. This causes interference with how the data structure is operated on by the threads leading to corrupt, or incorrect data structures. The issues surrounding race conditions can be resolved through using mutually exclusive practices such as implementing Peterson's Algorithm, using Semaphores, or by utilising language-level method synchronisation.

Wk 5 Exercises

1. Create a Resource Allocation Graph animation to demonstrate deadlock.



Output - wk5exp2 (run)

```
run:
Thread A waiting for R
Thread A acquired R
Thread A waiting for S
Thread A acquired S
Thread B waiting for S
Thread A released S
Thread A released R
Thread B acquired S
Thread A waiting for R
Thread A acquired R
Thread B waiting for R
Thread A waiting for S
```

View Resource Allocation Graph example animation ppt.

or download the same Resource Allocation Graph example.

2. Extend the Deadlock2 class to a Deadlock3 class to involve three threads and three resources.

With three resources and three threads that each only require two resources, the program can run for longer before a deadlock is reached.

Program works to the extent that three threads can wait for, acquire, and release from three resources. Program does not properly illustrate deadlock as it allows for threads acquiring the same resource.

Input:

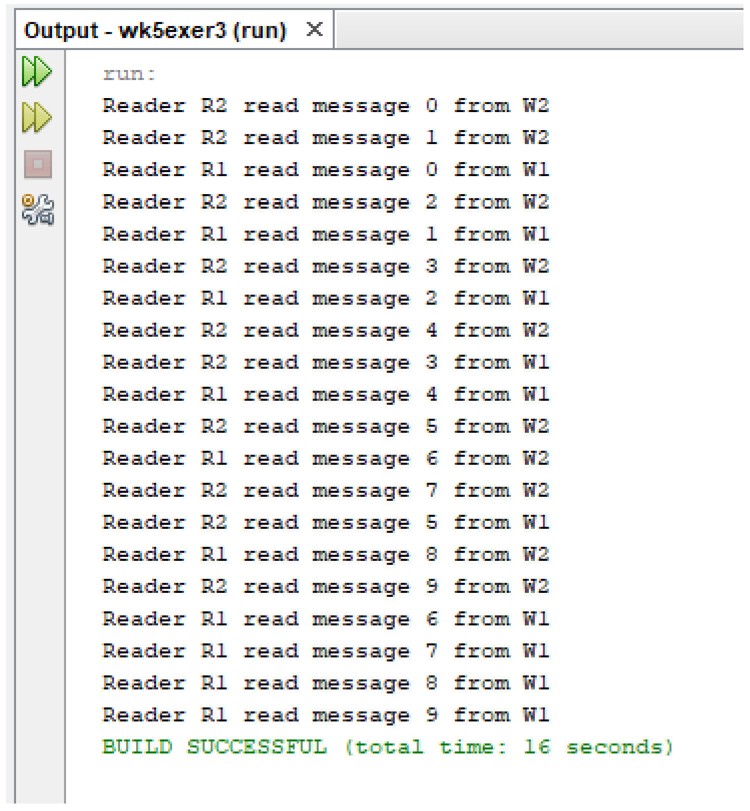
Output:

3. Run and investigate message passing code.

Input:

Code from here is run.

Output:



Output - wk5exer3 (run)

```
run:
Reader R2 read message 0 from W2
Reader R2 read message 1 from W2
Reader R1 read message 0 from W1
Reader R2 read message 2 from W2
Reader R1 read message 1 from W1
Reader R2 read message 3 from W2
Reader R1 read message 2 from W1
Reader R2 read message 4 from W2
Reader R2 read message 3 from W1
Reader R1 read message 4 from W1
Reader R2 read message 5 from W2
Reader R1 read message 6 from W2
Reader R2 read message 7 from W2
Reader R2 read message 5 from W1
Reader R1 read message 8 from W2
Reader R2 read message 9 from W2
Reader R1 read message 6 from W1
Reader R1 read message 7 from W1
Reader R1 read message 8 from W1
Reader R1 read message 9 from W1
BUILD SUCCESSFUL (total time: 16 seconds)
```

Explanation:

How many messages are sent and received in total?

Ten messages are sent in total.

How might you adapt this implementation so that senders can send messages to specifically chosen receivers?

The senders could send messages to specific receivers by including an additional variable in the Node class to determine a recipient. When one of the reader objects attempts to read from a Node, the access to the Node data must be validated by checking the recipient which could match with a thread ID.

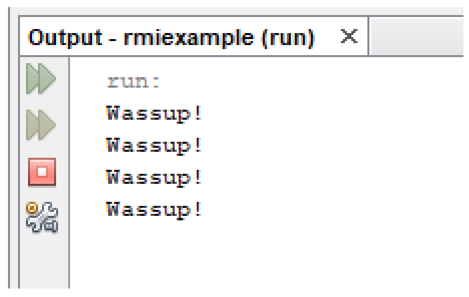
Week 7 - Remote Method Invocation

Experiment 1 - Using Java RMI

Server Code

```
package rmiexample;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class MyServer extends UnicastRemoteObject
       implements MyRemoteInterface {
    public void printMessage(String message) throws RemoteException {
       System.out.println(message);
   }
    public static void main(String args []) throws Exception {
        MyServer server = new MyServer();
        Registry reg = LocateRegistry.createRegistry(1234);
       reg.bind("myrmiserver", server);
   }
    public MyServer() throws RemoteException {}
}
```

Output after client program is run four times:



```
run:
Wassup!
Wassup!
Wassup!
Wassup!
```

Explanation:

The server code extends the UnicastRemoteObject class and implements the remote interface to print a message to the console. The main method in the MyServer class instantiates a new instance of the MyServer class and creates a registry at port 1234". The server then binds the instance of MyServer to the registry on port 1234 and names server bound to the registry "myrmiserver".

Client Code

```
package rmiexample;
import java.rmi.registry.*;
public class MyClient {

   public static void main(String[] args) throws Exception {
       Registry reg = LocateRegistry.getRegistry("148.197.235.241", 1234) ;
       MyRemoteInterface handle = (MyRemoteInterface) reg.lookup("myrmiserver");
       handle.printMessage("Wassup!");
 }

}
```

Output:

Client program has no console output, apart from confirming a successful build, as message delivered from the client is output on the server's console.

Explanation:

The client code initialises a registry value by getting a registry at a specific port and IP address. The port should be the port determined by the server code and the IP address should be the address where the server code is running. The client program then initialises a MyRemoteInterface value, "handle", by looking up the server "myrmiserver" at the registry the client gained from the server. The remote interface's method is then called by passing the string "Wassup!" as a parameter and the method is executed on the server as implemented by the server.

Remote Interface Code

```
package rmiexample;
import java.rmi.*;
public interface MyRemoteInterface extends Remote {
   void printMessage(String message) throws RemoteException ;
}
```

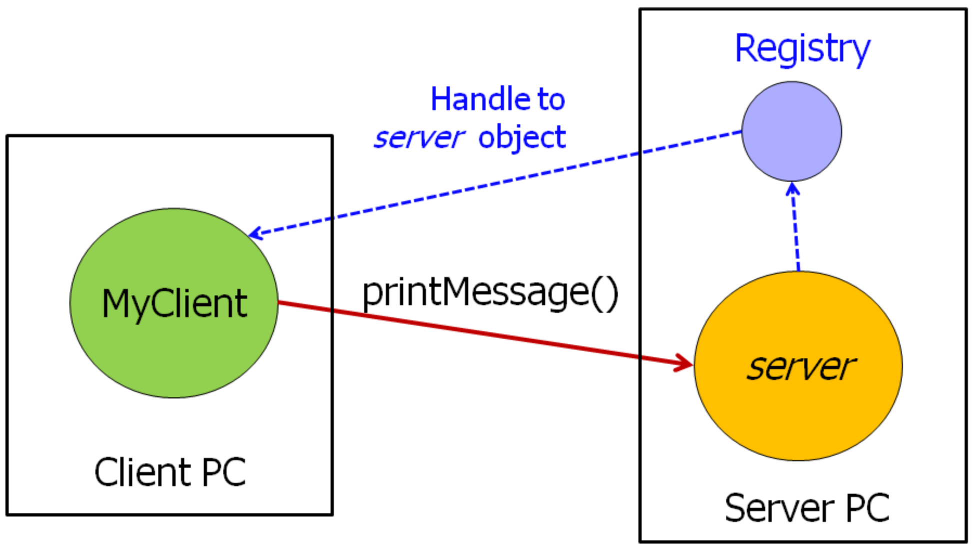
Explanation:

The remote interface is common between both the server and the client programs. A remote interface is similar to a class however it can only contain method signatures and data fields, but not method implementations. The interface MyRemoteInterface is implemented in the MyServer class to print the message from the MyClient class. The MyClient class creates an instance of the MyRemoteInterface and looks for a certain server name at a certain IP address to call the remote interface method on.

Reflective Question - Discuss what happened in Experiment 1, relating to Remote Procedure Calls

Experiment 1 illustrates the use of Remote Procedure Calls through sending a message from a client, "MyClient", to a server, "MyServer". The remote interface, "MyRemoteInterface", consists of a method signature for a method to print a message which is implemented by the server code, but called within the client code which allows the use (or call) of a procedure by the client, despite not knowing the implementation.

Method arguments are handled by the server creating a registry for the client to access which allows the the client to invoke the method implemented by the server as the method interface is shared by both the client and the server. Once the client 'gets' the registry from the server at the servers IP, on the port the server created the registry on, the client can invoke the remote method at the servers side.



Week 7 Exercises:

1. Adapt MyClient to Take a Typed Message

Input:

In the MyClient class

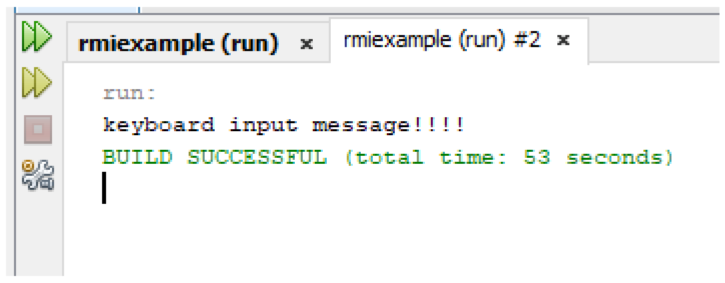
```
handle.printMessage("Wassup!");
```

is replaced with

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
String message = in.readLine() ;
handle.printMessage(message);
```

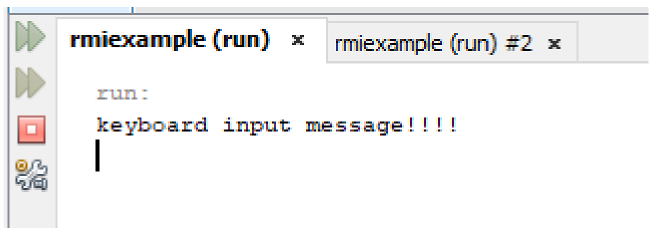
Output:

MyClient



```
run:
keyboard input message!!!!
BUILD SUCCESSFUL (total time: 53 seconds)
```

MyServer



```
run:
keyboard input message!!!!
```

Explanation:

Through using the BufferedReader and InputStreamReader classes, I was able to easily take input from the console and passed the entered string through the handle.printMessage() call. The string is treated as in Experiment 1 and output on the console at the client.

2. Adapt MyClient to Use a While Loop to Take series of Messages

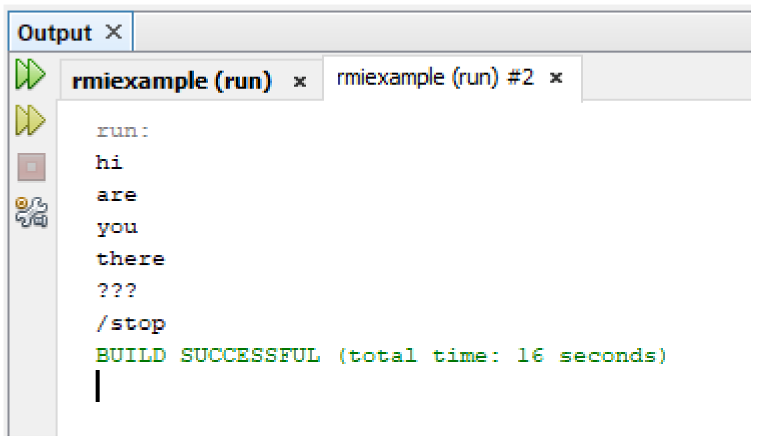
Input:

Previous input routine is replaced with:

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
boolean another = true;
while(another){
   String message = in.readLine();
   if(message.equalsIgnoreCase("/stop")){
       another = false;
   }else{
       handle.printMessage(message);
   }
}
```

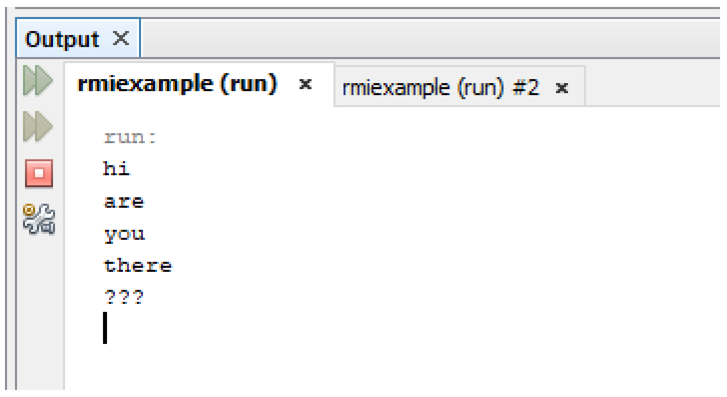
Output:

MyClient



```
run:
hi
are
you
there
???
/stop
BUILD SUCCESSFUL (total time: 16 seconds)
```

MyServer



```
run:
hi
are
you
there
???
```

Explanation:

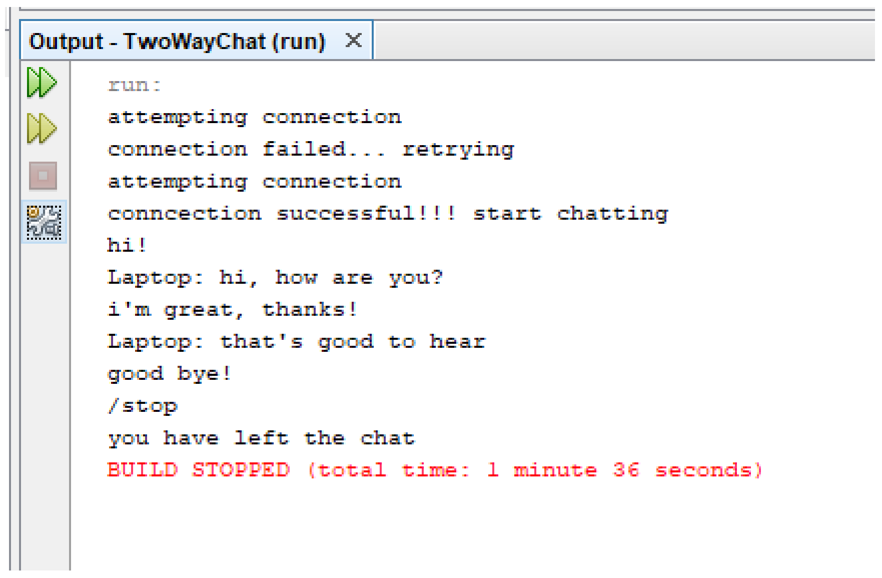
The code section that takes in console input is modified to use a Boolean variable and a while loop to continue taking messages from the console and printing them to the server console until "/stop" is entered.

3. Adapt the Program to create a Two Way chat program

Input:

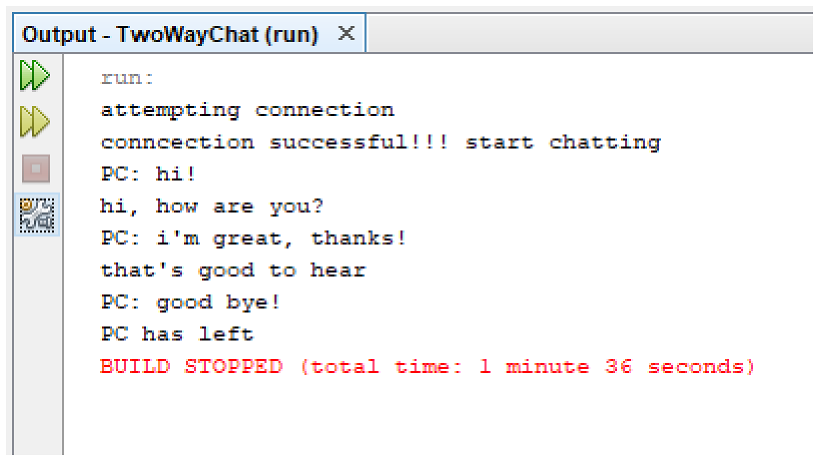
```
package twowaychat;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class TwoWayChat extends UnicastRemoteObject
                       implements MyRemoteInterface {
    public void printMessage(String message) throws RemoteException {
       System.out.println(message);
   }
   public static void main(String args []) throws Exception {
       //config
       String uname = "PC";
       String partnerIP = "10.128.159.170";
       //server code
       TwoWayChat chatServer = new TwoWayChat();
       Registry reg = LocateRegistry.createRegistry(1234);
       System.setProperty("java.rmi.server.hostname", "148.197.96.63");
       reg.rebind("mychatserver", chatServer);
       //clientcode
       MyRemoteInterface handle;
while(true){
    try{
        System.out.println("attempting connection");
        Registry partnerReg = LocateRegistry.getRegistry(partnerIP, 1234);
        handle = (MyRemoteInterface) partnerReg.lookup("mychatserver");
        System.out.println("conncection successful!!! start chatting");
        break;
    } catch (Exception e) {
          System.out.println("connection failed... retrying");
          Thread.currentThread().sleep(5*1000);
      }
}

boolean another = true;
       while(another){
           BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
           String message = in.readLine();
           if(message.equalsIgnoreCase("/stop")){
               System.out.println("you have left the chat");
               handle.printMessage(uname + " has left");
               another = false;
           }else{
               String totalMess = uname + ": " + message;
               handle.printMessage(totalMess);
           }
       }
   }
   public TwoWayChat() throws RemoteException {}
}
```

Output from the perspective of a user named PC:


Output - TwoWayChat (run)

```
run:
attempting connection
connection failed... retrying
attempting connection
conncection successful!!! start chatting
hi!
Laptop: hi, how are you?
i'm great, thanks!
Laptop: that's good to hear
good bye!
/stop
you have left the chat
BUILD STOPPED (total time: 1 minute 36 seconds)
```

Output from the perspective of a user named Laptop in the same session as the previous output:


Output - TwoWayChat (run)

```
run:
attempting connection
conncection successful!!! start chatting
PC: hi!
hi, how are you?
PC: i'm great, thanks!
that's good to hear
PC: good bye!
PC has left
BUILD STOPPED (total time: 1 minute 36 seconds)
```

Explanation:
The program expands upon the functionality of experiment 1 by first initialising the server, and then initialising the client after a delay, within the same file. This allows for a two way chat system that enables each user to enter messages at their IDE console and receive messages from their partner.

Additional modifications that I made include allowing the user to determine a username to be presented with their message, and for the console to stop taking input if the user types "/stop" as their message. Doing this will also message their partner that they have stopped entering messages.

Due to issues presented by the Oracle VM installed on university computers, we needed to include

```
-Djava.rmi.server.hostname=<localIPaddess>
```

in the VM Options in the Run Properties for the NetBeans IDE.

Week 8 - File Primitives

Experiment 1 - Creating Files

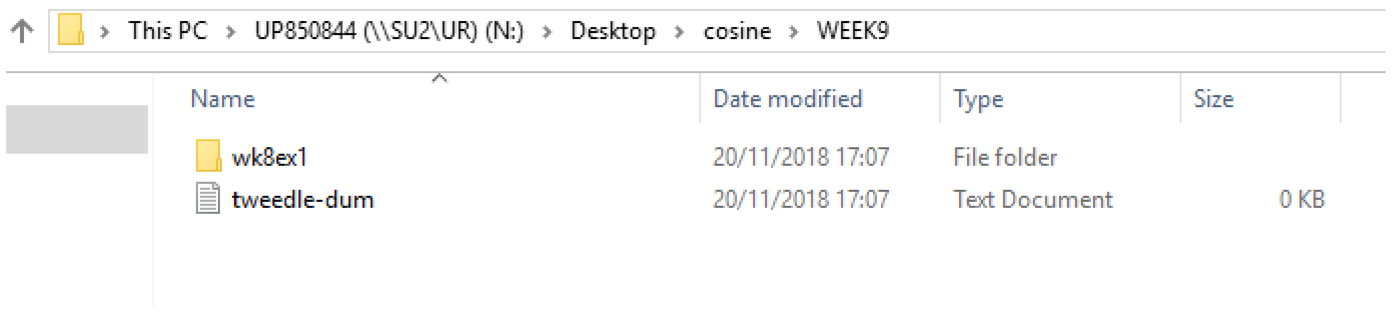
Input:

```
import java.io.*;
```

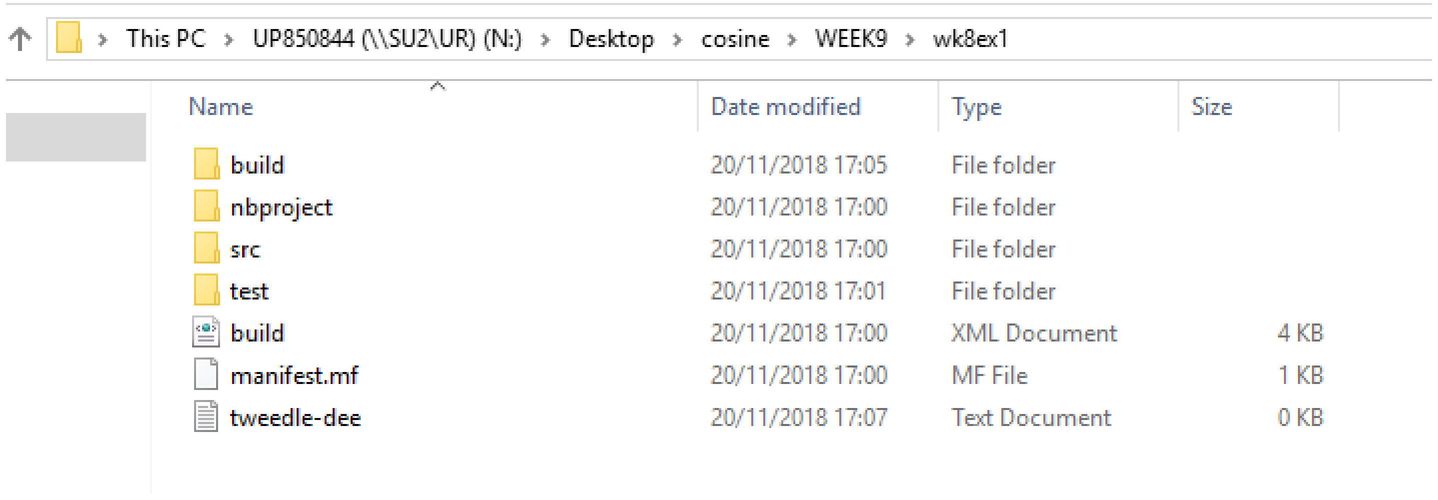
```
public class CreateFiles {

   public static void main(String[] args) throws Exception {
       File dum = new File("N:\\Desktop\\cosine\\WEEK9\\tweedle-dum.txt");
       File dee = new File("tweedle-dee.txt");
       dum.createNewFile();
       dee.createNewFile();
   }
}
```

Output:

Creation of two empty .txt files an absolute file path and in the current working directory.


This PC > UP850844 (\\SU2\UR) (N:) > Desktop > cosine > WEEK9

Name	Date modified	Type	Size
wk8ex1	20/11/2018 17:07	File folder	
tweedle-dum	20/11/2018 17:07	Text Document	0 KB



```
This PC > UP850844 (\\SU2\UR) (N:) > Desktop > cosine > WEEK9 > wk8ex1
```

Name	Date modified	Type	Size
build	20/11/2018 17:05	File folder	
nbproject	20/11/2018 17:00	File folder	
src	20/11/2018 17:00	File folder	
test	20/11/2018 17:01	File folder	
build	20/11/2018 17:00	XML Document	4 KB
manifest.mf	20/11/2018 17:00	MF File	1 KB
tweedle-dee	20/11/2018 17:07	Text Document	0 KB

Explanation:

The program creates two instances of the Java class File: one at an absolute path, named 'tweedle-dum.txt', and another in the path of the current working directory, named 'tweedle-dee.txt'. The program then calls a method to create files on both of the File objects.

Experiment 2 - Deleting Files

Input:

```
import java.io.*;
```

```
public class DeleteFiles {
   public static void main(String args []) throws Exception {
       File dum = new File("N:\\Desktop\\cosine\\WEEK9\\tweedle-dum.txt");
       File dee = new File("tweedle-dee.txt");
       dum.delete();
       dee.delete();
   }
}
```

Output:

Files created in the previous experiment are deleted.

Explanation:

The program creates two instances of the Java class File, as the files have already been created, the files that were created in the previous experiment are deleted as the method to delete the files is called on the two File objects with the same name and location as the created files from the previous example.

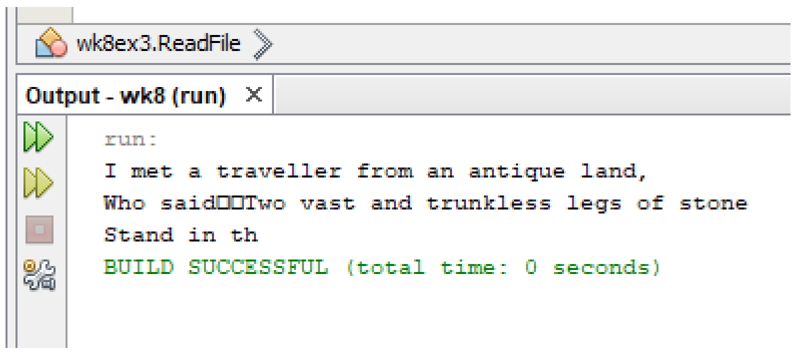
The files may not be deleted because...

Experiment 3 - Opening and Reading from a File

Input:

```
public class ReadFile {
   public static void main(String [] args) throws Exception {

       FileInputStream in = new FileInputStream("N:\\Desktop\\cosine\\WEEK9\\tweedle-dum.txt");
       byte buffer [] = new byte [100] ;
       int numBytesRead = in.read(buffer) ;
       System.out.println(new String(buffer)) ;
   }
}
```

Output:



```
run:
I met a traveller from an antique land,
Who said□□Two vast and trunkless legs of stone
Stand in th
BUILD SUCCESSFUL (total time: 0 seconds)
```

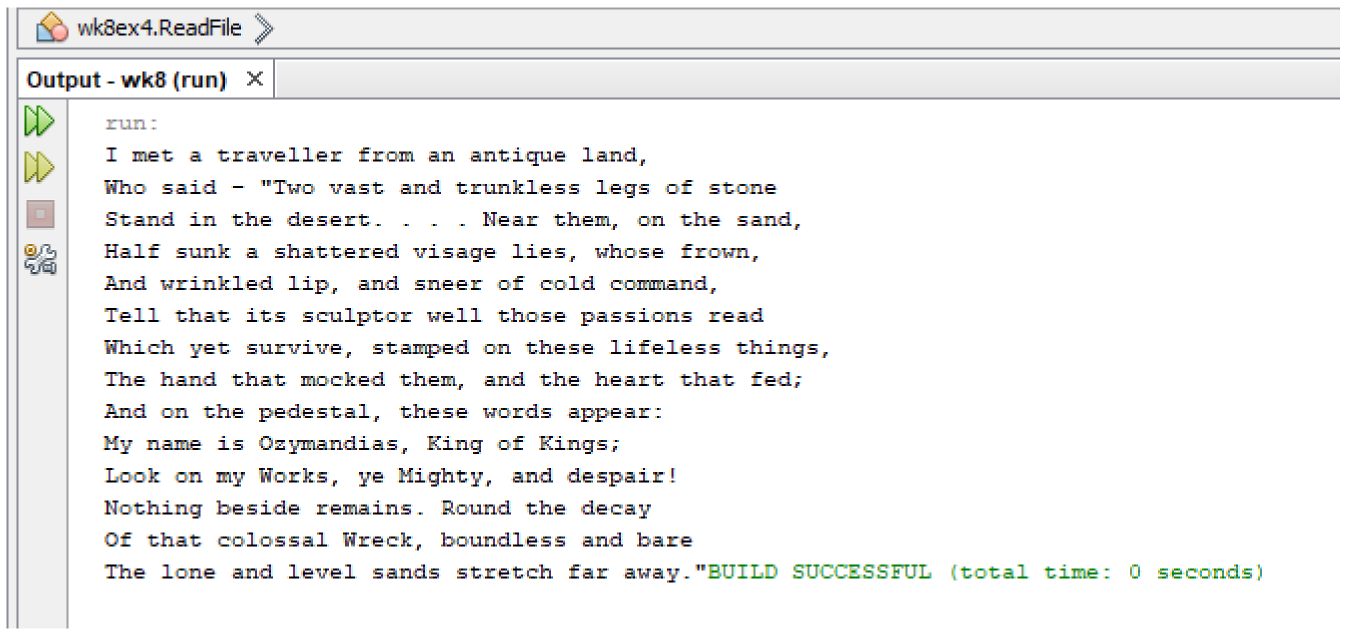
Explanation:

Experiment 4 - Reading Whole Files

Input:

```
public class ReadFile {
   public static void main(String [] args) throws Exception {

       FileInputStream in = new FileInputStream("N:\\Desktop\\cosine\\WEEK9\\tweedle-dum.txt");
       byte buffer [] = new byte [100] ;
       int numBytesRead = in.read(buffer) ;

       while(numBytesRead > 0){
           System.out.print(new String(buffer, 0, numBytesRead));
           numBytesRead = in.read(buffer);
       }
   }
}
```

Output:



wk8ex4.ReadFile

Output - wk8 (run)

```
run:
I met a traveller from an antique land,
Who said - "Two vast and trunkless legs of stone
Stand in the desert. . . . Near them, on the sand,
Half sunk a shattered visage lies, whose frown,
And wrinkled lip, and sneer of cold command,
Tell that its sculptor well those passions read
Which yet survive, stamped on these lifeless things,
The hand that mocked them, and the heart that fed;
And on the pedestal, these words appear:
My name is Ozymandias, King of Kings;
Look on my Works, ye Mighty, and despair!
Nothing beside remains. Round the decay
Of that colossal Wreck, boundless and bare
The lone and level sands stretch far away."BUILD SUCCESSFUL (total time: 0 seconds)
```

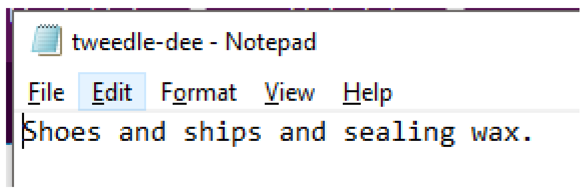
Explanation:

Experiment 5 - Writing a File

Input:

```
public class WriteFile {
   public static void main(String [] args) throws Exception {

       FileOutputStream out = new FileOutputStream("N:\\Desktop\\cosine\\WEEK9\\tweedle-dee.txt");
       String text = "Shoes and ships and sealing wax." ;
       byte buffer [] = text.getBytes();
       out.write(buffer) ;
       out.close() ;
   }
}
```

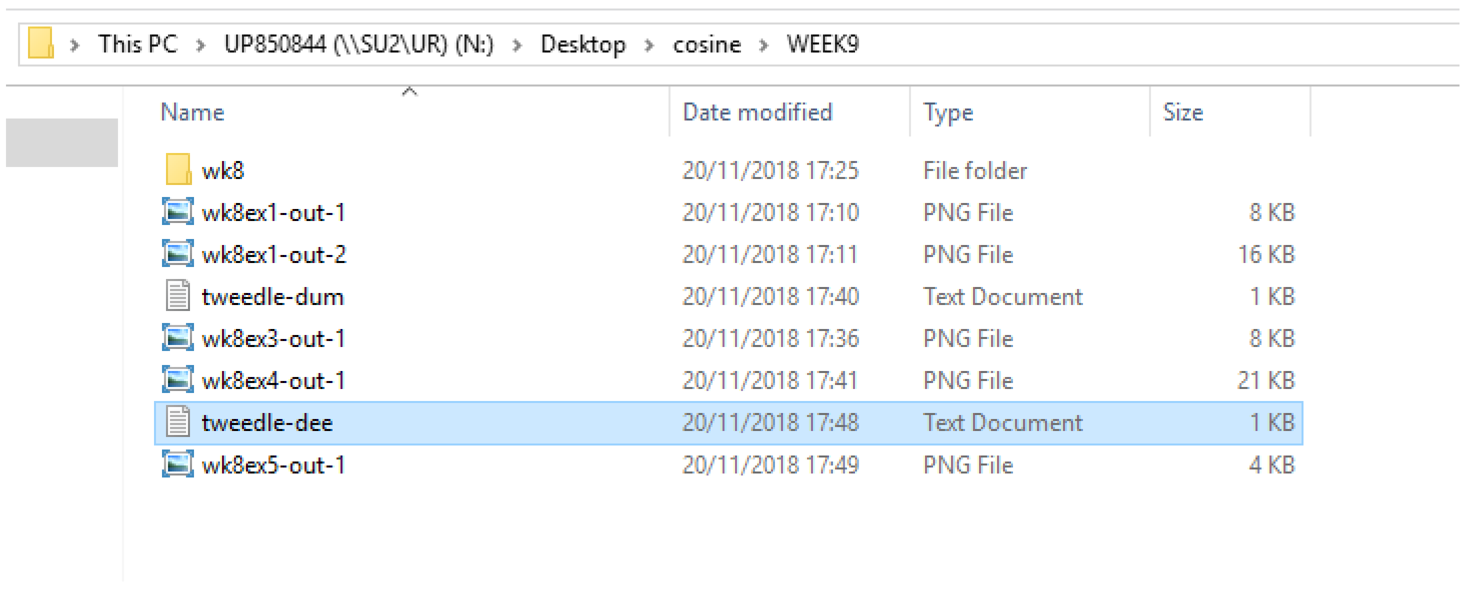
Output:



tweedle-dee - Notepad

File Edit Format View Help

```
Shoes and ships and sealing wax.
```



This PC › UP850844 (\\SU2\UR) (N:) › Desktop › cosine › WEEK9

Name	Date modified	Type	Size
wk8	20/11/2018 17:25	File folder	
wk8ex1-out-1	20/11/2018 17:10	PNG File	8 KB
wk8ex1-out-2	20/11/2018 17:11	PNG File	16 KB
tweedle-dum	20/11/2018 17:40	Text Document	1 KB
wk8ex3-out-1	20/11/2018 17:36	PNG File	8 KB
wk8ex4-out-1	20/11/2018 17:41	PNG File	21 KB
tweedle-dee	20/11/2018 17:48	Text Document	1 KB
wk8ex5-out-1	20/11/2018 17:49	PNG File	4 KB

Explanation:

Experiment 6 - Reading and Writing Files

Input:

Output:

Explanation:

Experiment 7 - Seeking Files

Input:

Output:

Explanation:

Experiment 8 - Random Seeking Files

Input:

Output:

Explanation:

Experiment 9 - File Attributes

Input:

Output:

Explanation:

Reflective Question - POSIX Systems calls for Primitive File Operations

Week 8 Exercises

1. Experiment with directory manipulation methods

2. Write a program to copy files from a directory

3. Use RMI to create a simple "network file server"

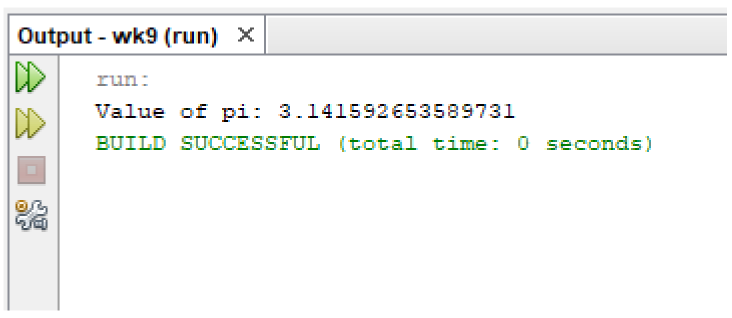
Week 9 - Parallel Processing/Programming

Calculating π

Input

```
public class SequentialPI {
   public static void main(String[] args) {
       int numSteps = 10000000;
       double step = 1.0 / (double) numSteps;
       double sum = 0.0;
       for(int i = 0 ; i < numSteps ; i++){
           double x = (i + 0.5) * step ;
           sum += 4.0 / (1.0 + x * x);
       }
       double pi = step * sum ;
       System.out.println("Value of pi: " + pi);
   }
}
```

Output:



```
run:
Value of pi: 3.141592653589731
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

The value found from the program would seem accurate to 14 significant figures compared to this estimation found here.

A Parallel Program

Benchmarking

Exercises

1. Make a "quad-core" version of ParallelPi, which divides the range of the sum into four equal parts, and uses four threads.

Does this version go faster than the two-threaded version?

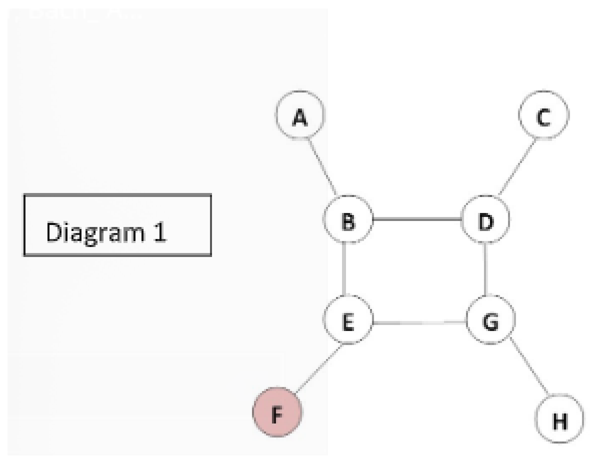
Week 13 - Networking Review

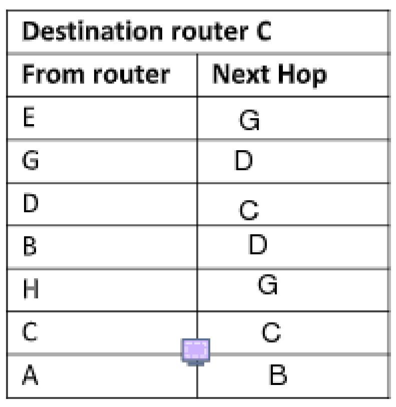
document

Week 14 - Static Routing

Task A - Detail Hops Across a Network

Diagram 1 is a design of a network of interconnected routers. Starting form router F, complete the table below. Here you need to detail the next hop from each router to get to router C.





Destination router C	
From router	**Next Hop**
E	G
G	D
D	C
B	D
H	G
C	C
A	B

If you start at router F and wish to send a packet to C. Which route would you take?

F -> E -> G -> D -> C

or

F -> E -> B -> D -> C

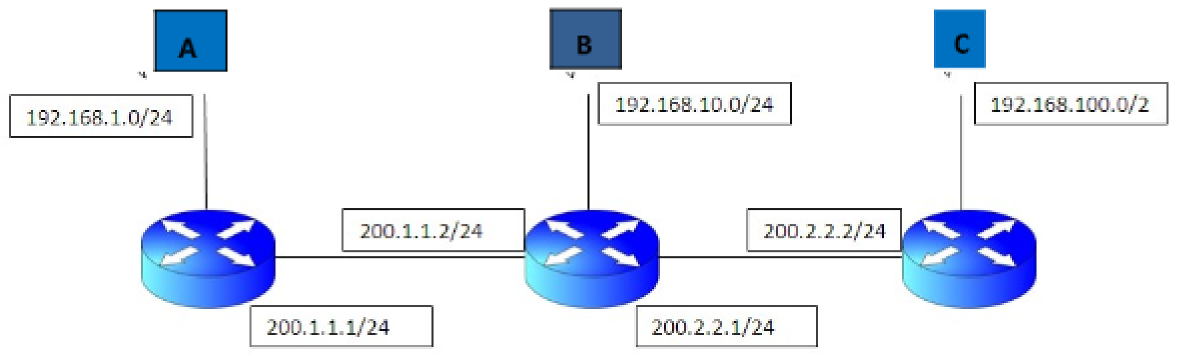
Will there be any impact which route you will take?

From the two routes suggested there would be no difference in which route is selected, so long as the interconnections between the nodes and the nodes themselves are functionally identical. This is because both routes require the same number of hops.

Other concerns will include the cost and the bandwidth of transfer across the interconnections and whether hop count is a priority.

Task B - Develop a Static Routing Table

Develop a static routing table for router B below to include all route connected to them.



Task C - Provide a Best route

Starting from R MI with your destination R VC . Provide the best route to achieve this.

Note: The first diagram has the IP addresses for each connected port on the router. The second diagram has the cost of each connection between the routers.

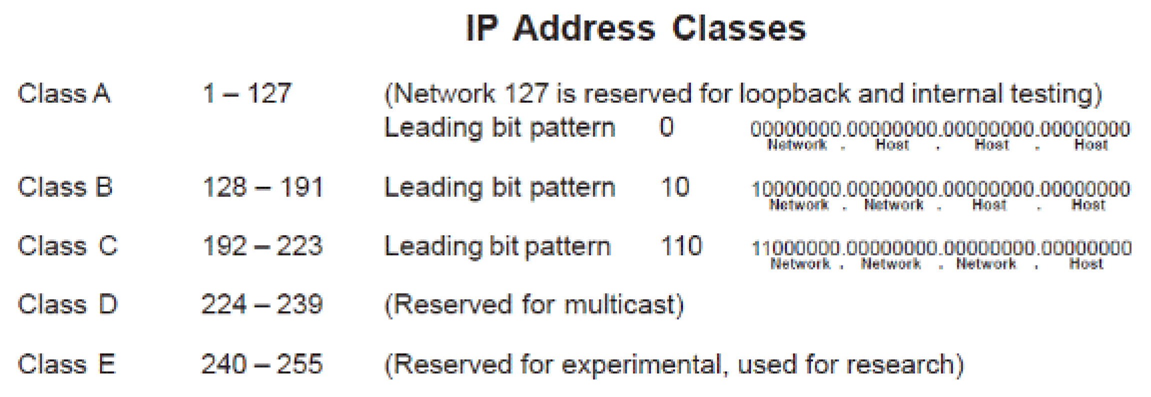
Question on Static Routing Drawbacks

What are the 4 main drawbacks of static routing?

- Near impossible to maintain with larger networks

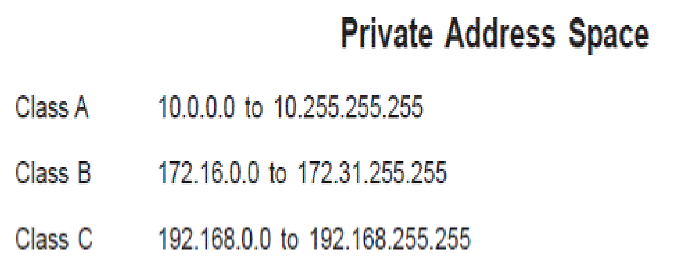
Week 15 - Addresses and Sub-netting

Given notes from worksheet:



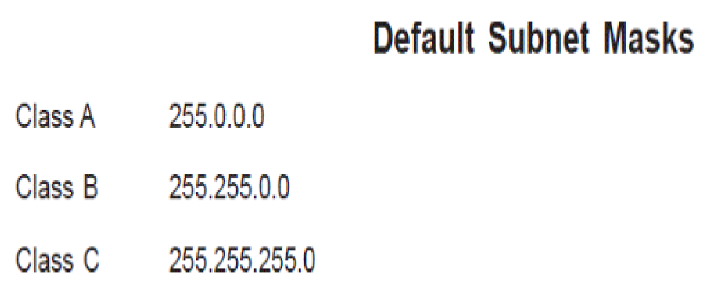
IP Address Classes

Class A	1 – 127	(Network 127 is reserved for loopback and internal testing) Leading bit pattern	0	00000000.00000000.00000000.00000000 Network . Host . Host . Host
Class B	128 – 191	Leading bit pattern	10	10000000.00000000.00000000.00000000 Network . Network . Host . Host
Class C	192 – 223	Leading bit pattern	110	11000000.00000000.00000000.00000000 Network . Network . Network . Host
Class D	224 – 239	(Reserved for multicast)		
Class E	240 – 255	(Reserved for experimental, used for research)		



Private Address Space

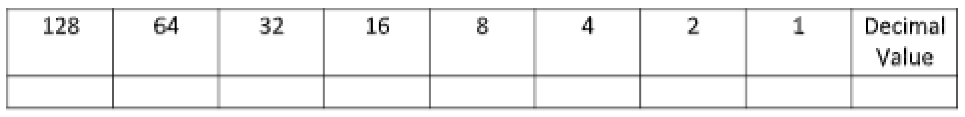
Class A	10.0.0.0 to 10.255.255.255
Class B	172.16.0.0 to 172.31.255.255
Class C	192.168.0.0 to 192.168.255.255



Default Subnet Masks

Class A	255.0.0.0
Class B	255.255.0.0
Class C	255.255.255.0

Exercise 1 - Convert the following binary numbers to decimal by using the provided table.



128	64	32	16	8	4	2	1	Decimal Value

00011011 == 16 + 8 + 2 + 1 == 27

10101010 == 128 + 32 + 8 + 2 == 170

01101111 == 64 + 32 + 16 + 8 + 4 + 2 + 1 == 127

11111000 == 128 + 64 +32 + 16 + 8 == 248

00100000 == 32

01010101 == 64 + 16 + 4 + 1 == 85

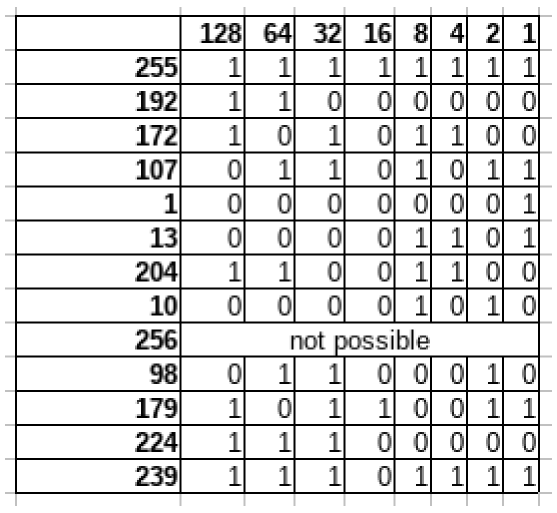
00111110 ==32 + 16 + 8 + 4 + 2 == 62

00000011 == 2 + 1 == 3

11101101 == 128 + 64 + 32 + 8 + 4 + 1 == 237

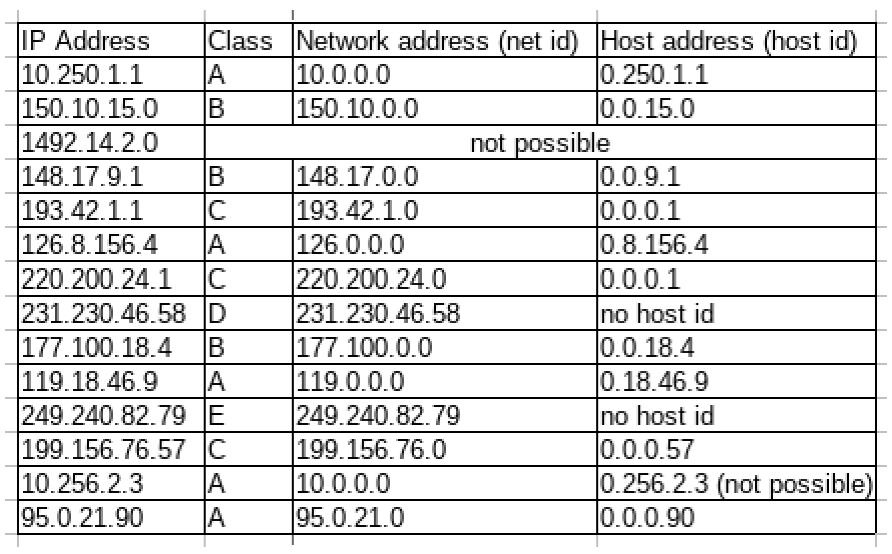
11000000 == 128 + 64 == 192

Exercise 2 - Convert the following decimal numbers to binary by using the provided table.



	128	64	32	16	8	4	2	1
255	1	1	1	1	1	1	1	1
192	1	1	0	0	0	0	0	0
172	1	0	1	0	1	1	0	0
107	0	1	1	0	1	0	1	1
1	0	0	0	0	0	0	0	1
13	0	0	0	0	1	1	0	1
204	1	1	0	0	1	1	0	0
10	0	0	0	0	1	0	1	0
256	not possible							
98	0	1	1	0	0	0	1	0
179	1	0	1	1	0	0	1	1
224	1	1	1	0	0	0	0	0
239	1	1	1	0	1	1	1	1

Exercise 3/4 - Identify class, net id, and host id for the following IP addresses.

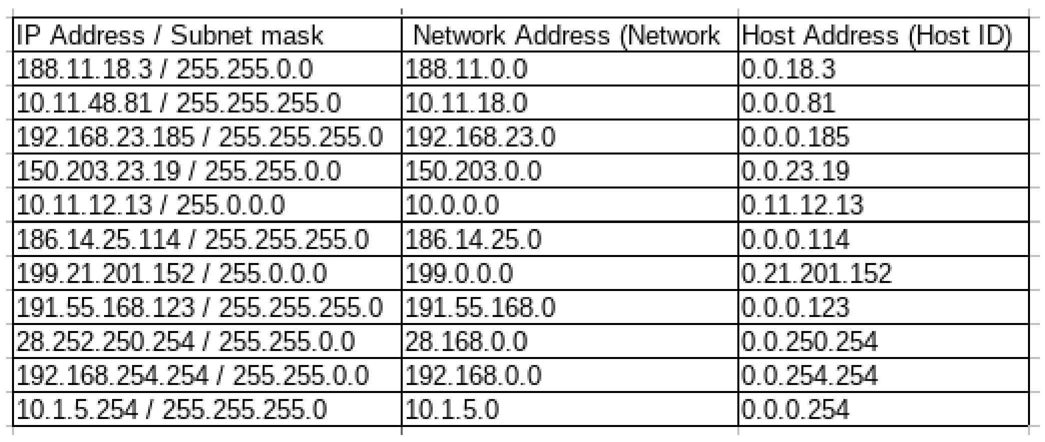


IP Address	Class	Network address (net id)	Host address (host id)
10.250.1.1	A	10.0.0.0	0.250.1.1
150.10.15.0	B	150.10.0.0	0.0.15.0
1492.14.2.0	not possible		
148.17.9.1	B	148.17.0.0	0.0.9.1
193.42.1.1	C	193.42.1.0	0.0.0.1
126.8.156.4	A	126.0.0.0	0.8.156.4
220.200.24.1	C	220.200.24.0	0.0.0.1
231.230.46.58	D	231.230.46.58	no host id
177.100.18.4	B	177.100.0.0	0.0.18.4
119.18.46.9	A	119.0.0.0	0.18.46.9
249.240.82.79	E	249.240.82.79	no host id
199.156.76.57	C	199.156.76.0	0.0.0.57
10.256.2.3	A	10.0.0.0	0.256.2.3 (not possible)
95.0.21.90	A	95.0.21.0	0.0.0.90

Exercise 5 - Calculate the network address and host address for the following IP addresses as defined by the provided custom sub-net mask.

For example:

156.222.32.27 / 255.255.255.0 -> 156.222.32.0 (network) and 0.0.0.27 (host)



IP Address / Subnet mask	Network Address (Network	Host Address (Host ID)
188.11.18.3 / 255.255.0.0	188.11.0.0	0.0.18.3
10.11.48.81 / 255.255.255.0	10.11.18.0	0.0.0.81
192.168.23.185 / 255.255.255.0	192.168.23.0	0.0.0.185
150.203.23.19 / 255.255.0.0	150.203.0.0	0.0.23.19
10.11.12.13 / 255.0.0.0	10.0.0.0	0.11.12.13
186.14.25.114 / 255.255.255.0	186.14.25.0	0.0.0.114
199.21.201.152 / 255.0.0.0	199.0.0.0	0.21.201.152
191.55.168.123 / 255.255.255.0	191.55.168.0	0.0.0.123
28.252.250.254 / 255.255.0.0	28.168.0.0	0.0.250.254
192.168.254.254 / 255.255.0.0	192.168.0.0	0.0.254.254
10.1.5.254 / 255.255.255.0	10.1.5.0	0.0.0.254

Exercise 6 - For the following IP address and the provided sub-net mask calculate the number of sub-nets and hosts you have for each sub-net.

Scenario 1 - Default sub-net mask

IP address: 192.100.10.0

Number of sub-nets = 24

Number of hosts per sub-net = 254

Scenario 2 – Custom sub-net mask

IP address: 192.100.10.0

Custom Sub-net mask: 255.255.255.240

Number of sub-nets = 28

Number of hosts per sub-net = 14

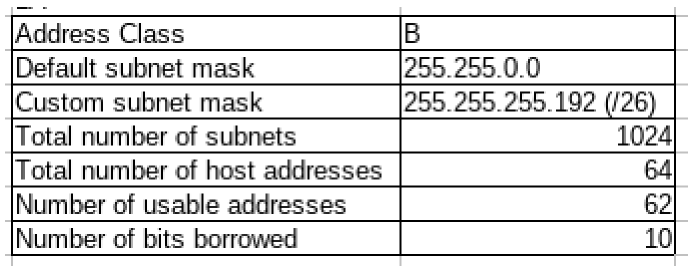
Week 16 - CIDR, Subnetting and Supernetting (Marking Deadline Week 20)

Important: *You will be assessed on* ***Exercise 1-c****,* ***Exercise 2-b*** *and* ***Exercise 3-c****.*

Task 1 - Complete Following Tables Based on Specification

1A - Class B Network Subnetting

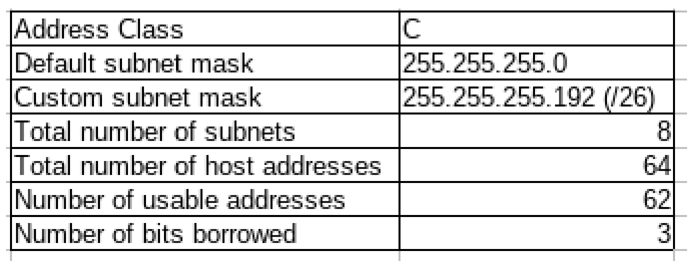
Number of subnets in need: **1000**
Number of usable hosts in need: **60**
Network address: **166.102.0.0**



Address Class	B
Default subnet mask	255.255.0.0
Custom subnet mask	255.255.255.192 (/26)
Total number of subnets	1024
Total number of host addresses	64
Number of usable addresses	62
Number of bits borrowed	10

1B - Class C Network Subnetting

Number of subnets in need: **6**
Number of usable hosts in need: **30**
Network address: **195.86.12.0**



Address Class	C
Default subnet mask	255.255.255.0
Custom subnet mask	255.255.255.192 (/26)
Total number of subnets	8
Total number of host addresses	64
Number of usable addresses	62
Number of bits borrowed	3

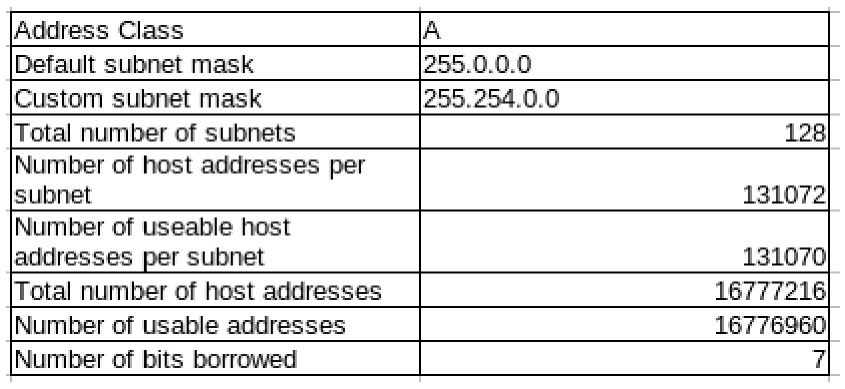
1C - Class A Network Subnetting *

Number of subnets in need: **126**
Number of usable hosts in need: **131,070**
Network address: **117.0.0.0**

Custom subnet mask = 11111111.**1111111**0.00000000.00000000

As 7 bits need to be borrowed from the net id (as 2^7 = 128, which can accommodate 126 subnets), there are 17 bits remaining for host addresses per subnet = 2^17 = 131072. The number of useable addresses per subnet = 131072 - 2 = 131070.

For the network overall, the total number of host addresses will be **16777216** (128*131072), but there will exist **16776960** (128 * 131070) potential useable network addresses or **16514820** (126*131070) useable addresses if only the required 126 subnets are used.



Address Class	A
Default subnet mask	255.0.0.0
Custom subnet mask	255.254.0.0
Total number of subnets	128
Number of host addresses per subnet	131072
Number of useable host addresses per subnet	131070
Total number of host addresses	16777216
Number of usable addresses	16776960
Number of bits borrowed	7

Task 2 - Complete the Following Tables Based on Their Specifications

2A - Class C Subnetting

Number of subnets in need: **2**
Network address: **195.223.50.0**

Showing Working:

As 2 subnets are needed, 1 bit will need to be borrowed from the host ID as 2^1 = 2.

11111111.11111111.11111111.1**000000**

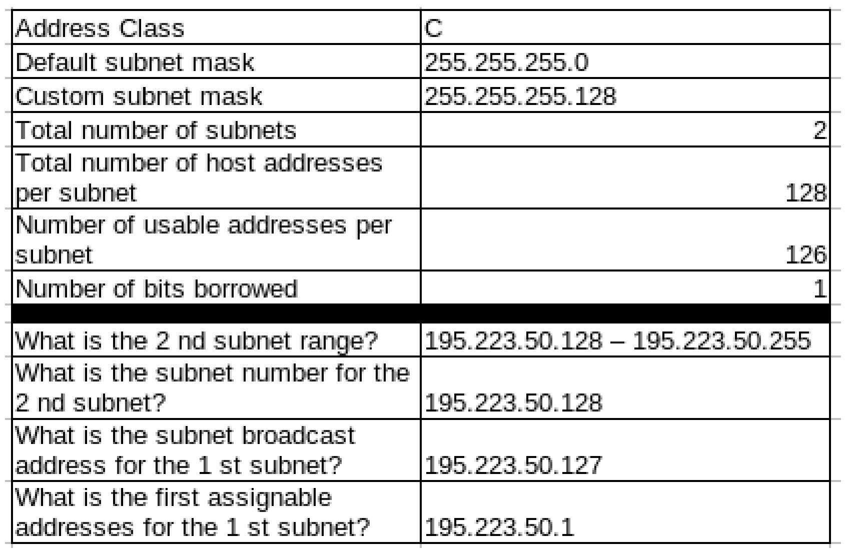
7 bits remain for host IDs, 2^7 = 128 total host addresses.

128 - 2 = 126 usable host addresses

The second address will start after 128 addresses so the subnet number for the 2nd subnet is 195.233.500.128. Then, as each subnet has 128 addresses, the last value in upper bound of the 2nd subnet address range will be 128 + 127 (as the address range starts at 128) = 255

The broadcast address for the first subnet will be the last address in the range which is 193.223.50.127

As the zeroth address is reserved to denote the subnet, the first useable address is 195.223.50.1



Address Class	C
Default subnet mask	255.255.255.0
Custom subnet mask	255.255.255.128
Total number of subnets	2
Total number of host addresses per subnet	128
Number of usable addresses per subnet	126
Number of bits borrowed	1
What is the 2 nd subnet range?	195.223.50.128 – 195.223.50.255
What is the subnet number for the 2 nd subnet?	195.223.50.128
What is the subnet broadcast address for the 1 st subnet?	195.223.50.127
What is the first assignable addresses for the 1 st subnet?	195.223.50.1

2B - Class A Subnetting *

Number of usable hosts per subnet in need: **6**
Network address: **126.0.0.0**

Showing Working:

As 6 hosts are needed, only 3 bits are needed as part of the network ID as 2^3 = 8 addresses, which can accommodate 6 usable networks as the number of useable networks with 3 bits reserved for the host id = 2^3 - 3 = 6.

As only 3 bits are reserved for the network ID, and the network address for the network needs 6 bits:

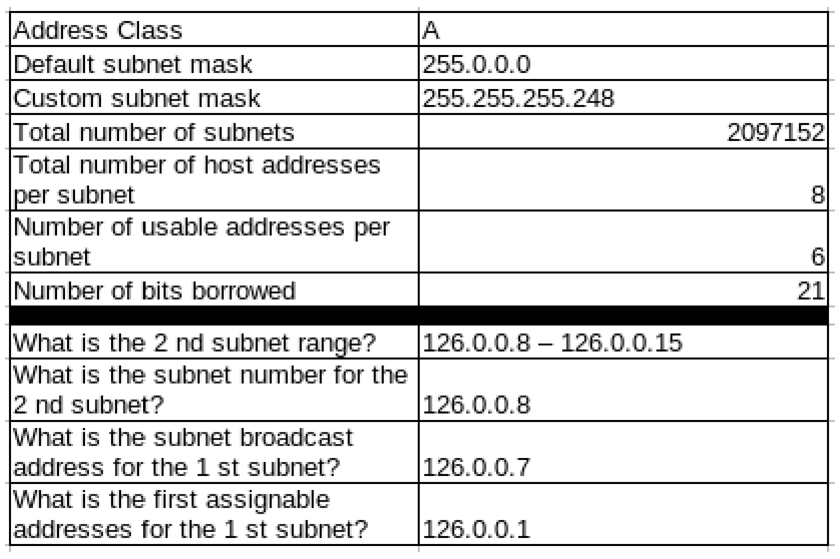
32 - 6 - 3 = 21 bits which are borrowed for subnetting

2^21 = 2,097,152 subnets

Second subnet address range is from 126.0.0.8 - 126.0.0.15 as 8 addresses are counted up from 126.0.0.7 which is the last address in the previous address.

The broadcast address for the first subnet is 126.0.0.7 as it is the last address in the subnet

The first assignable address in the first subnet is 126.0.0.1 as 126.0.0.0 is reserved to denote the subnet address.



Address Class	A
Default subnet mask	255.0.0.0
Custom subnet mask	255.255.255.248
Total number of subnets	2097152
Total number of host addresses per subnet	8
Number of usable addresses per subnet	6
Number of bits borrowed	21
What is the 2 nd subnet range?	126.0.0.8 – 126.0.0.15
What is the subnet number for the 2 nd subnet?	126.0.0.8
What is the subnet broadcast address for the 1 st subnet?	126.0.0.7
What is the first assignable addresses for the 1 st subnet?	126.0.0.1

Task 3 - Supernetting

3A - Identify the Supernet address and Supernet Mask for the Following Networks

(Hint: You might want to convert them into binary and calculate the common bits)

151.12.0.0 /16, 151.13.0.0 /16, 151.14.0.0 /16, 151.15.0.0 /16

Supernet Address - 151.0.0.0 /16

Supernet Mask - 11111111.11111111.00000000.00000000 = 255.255.0.0

3B - Calculate the Supernet Address and Mask For the Following Networks

211.7.0.0 /24, 211.7.1.0 /24, 211.7.2.0 /24, 211.7.3.0 /24

Supernet Address - 211.7.0.0/24

Supernet Mask - 11111111.11111111.11111111.00000000 = 255.255.255.0

3C - A company has been allocated the address range **193.160.0.0** to **193.175.255.255** *

```
What is the network (supernet) address and what is the (supernet) mask?
```

```
    193.160.0.0 = 11000001.10100000.00000000.00000000
```

```
193.175.255.255 = 11000001.10101111.11111111.11111111
```

```
supernet mask   = 11111111.11110000.00000000.00000000 = 255.240.0.0
```

Supernet Address - 193.160.0.0

Supernet Mask - 255.240.0.0

3D - A company has been allocated the network address **210.84.224.0 /22**.

Calculate the network (supernet) address and what are the first and last host addresses in the range?

11111111.11111111.1111*11*00.00000000 = SNM

11010010.01010100.11100000.00000000 = network address

Supernet Address - 210.84.224.0/22

First Host Address - 210.84.224.1

Last Host Address - 210.84.252.254

How many host addresses can be allocated (usable)?

32 - 22 = 10

2 ^ 10 = 1024 useable hosts

Week 17 - Dynamic Routing

Question 1 - Work out the Link State Algorithm for Network Shown

Determine the next hops

Question 2 - Using the Distance Vector algorithm find the shortest path from node D to other destination nodes.

Compare it against the cost of the Link State algorithm.

Lab Question - Compare the dynamic and static routing protocols and highlight the strength and weakness of them both.

Link State improves over distance vector in the sense that updated packets can be sent via multicast rather than broadcast and the algorithm can reduce the processing on non-router devices. As well as this, when using Link State, unnecessary traffic can be reduced and it [what?] can be configured in a hierarchical fashion.

Week 19 - RIP, Routing Information Protocol

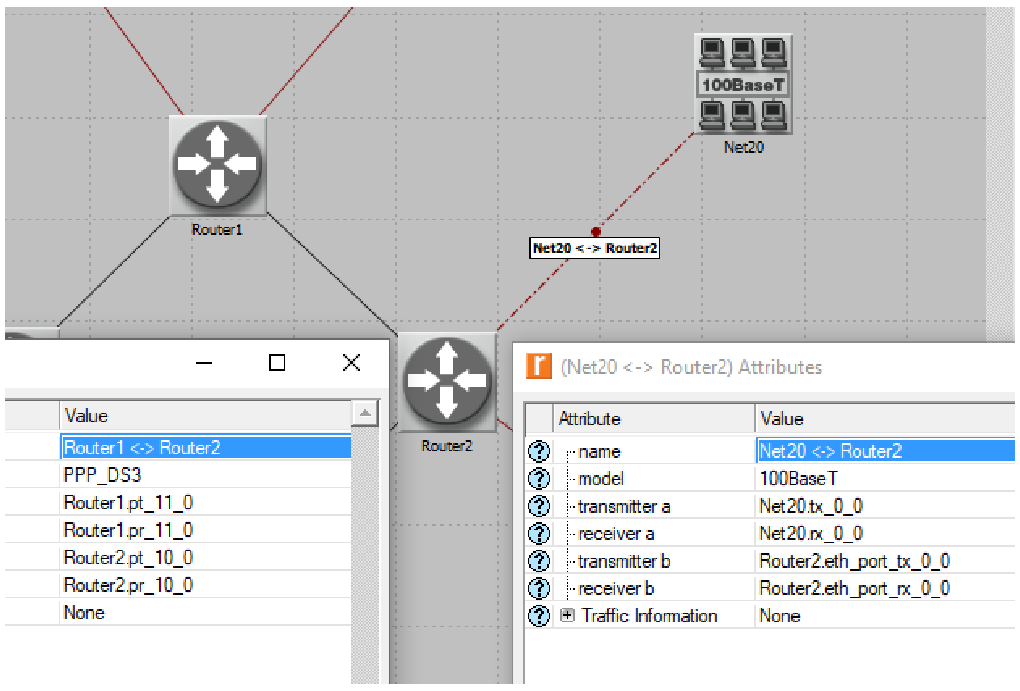
"The fundamental problem of routing is: How do routers acquire the information in their forwarding tables?"

W19 Summary

In this practical we simulated a network using RIP. We evaluated the performance of RIP by comparing network statistics in two scenarios, where one introduced a failed link after a period of time. We also compared the routing tables for each scenario to understand how RIP works. We then looked at implementing a simulation of a network that uses RIP which exhibits a failed link then recovers that link.

W19 Implementation

In the Riverbed Academic Modeler, we simulated a network with 4 routers (connected in a ring with PPP DS3 links), each connected to two 100BaseT LANs with bidirectional 100BaseT links.

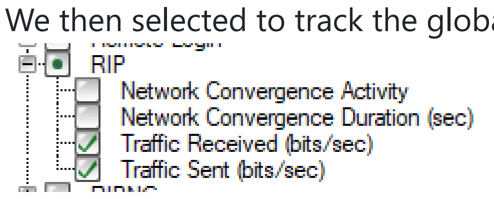


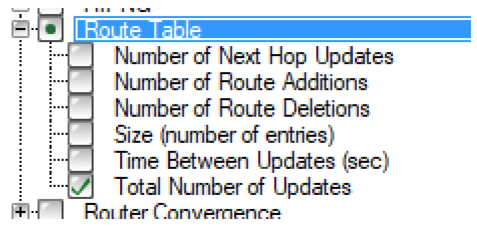
Value
Router1 <-> Router2
PPP_DS3
Router1.pt_11_0
Router1.pr_11_0
Router2.pt_10_0
Router2.pr_10_0
None

(Net20 <-> Router2) Attributes

Attribute	Value
name	Net20 <-> Router2
model	100BaseT
transmitter a	Net20.tx_0_0
receiver a	Net20.rx_0_0
transmitter b	Router2.eth_port_tx_0_0
receiver b	Router2.eth_port_rx_0_0
⊞ Traffic Information	None

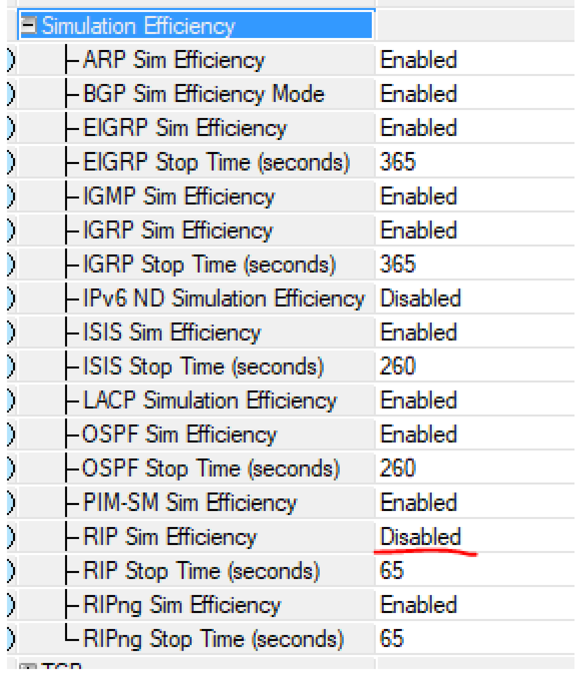
We then selected to track the global RIP traffic sent and received statistics as well as the number of updates on the route table.


RIP
- [ ] Network Convergence Activity
- [ ] Network Convergence Duration (sec)
- [✓] Traffic Received (bits/sec)
- [✓] Traffic Sent (bits/sec)

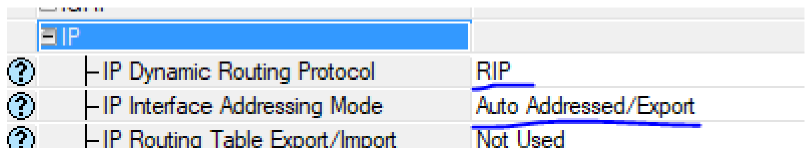

Route Table
- [ ] Number of Next Hop Updates
- [ ] Number of Route Additions
- [ ] Number of Route Deletions
- [ ] Size (number of entries)
- [ ] Time Between Updates (sec)
- [✓] Total Number of Updates

Router Convergence

We then configured the simulation to run for 10 minutes, to use RIP as the IP Dynamic routing protocol, and to disable the the RIP Sim efficiency. We also selected the Auto Addresses/Export option for the IP Interface Addressing mode setting.

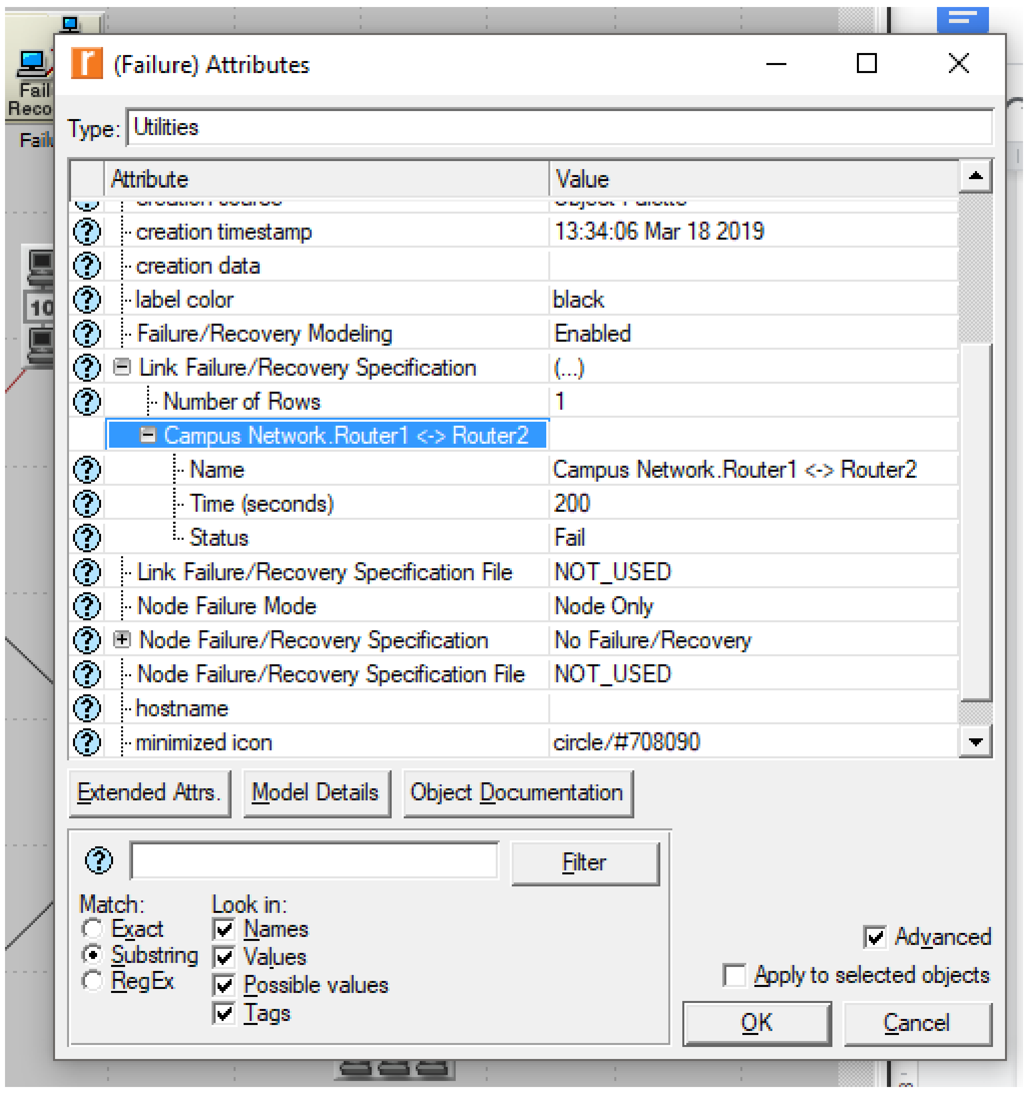


Simulation Efficiency	
ARP Sim Efficiency	Enabled
BGP Sim Efficiency Mode	Enabled
EIGRP Sim Efficiency	Enabled
EIGRP Stop Time (seconds)	365
IGMP Sim Efficiency	Enabled
IGRP Sim Efficiency	Enabled
IGRP Stop Time (seconds)	365
IPv6 ND Simulation Efficiency	Disabled
ISIS Sim Efficiency	Enabled
ISIS Stop Time (seconds)	260
LACP Simulation Efficiency	Enabled
OSPF Sim Efficiency	Enabled
OSPF Stop Time (seconds)	260
PIM-SM Sim Efficiency	Enabled
RIP Sim Efficiency	Disabled
RIP Stop Time (seconds)	65
RIPng Sim Efficiency	Enabled
RIPng Stop Time (seconds)	65



IP	
IP Dynamic Routing Protocol	RIP
IP Interface Addressing Mode	Auto Addressed/Export
IP Routing Table Export/Import	Not Used

We then duplicated this scenario and introduced a Failure Recovery object to simulate a link failure between Router1 and Router2 after 200 seconds.



(Failure) Attributes

Type: Utilities

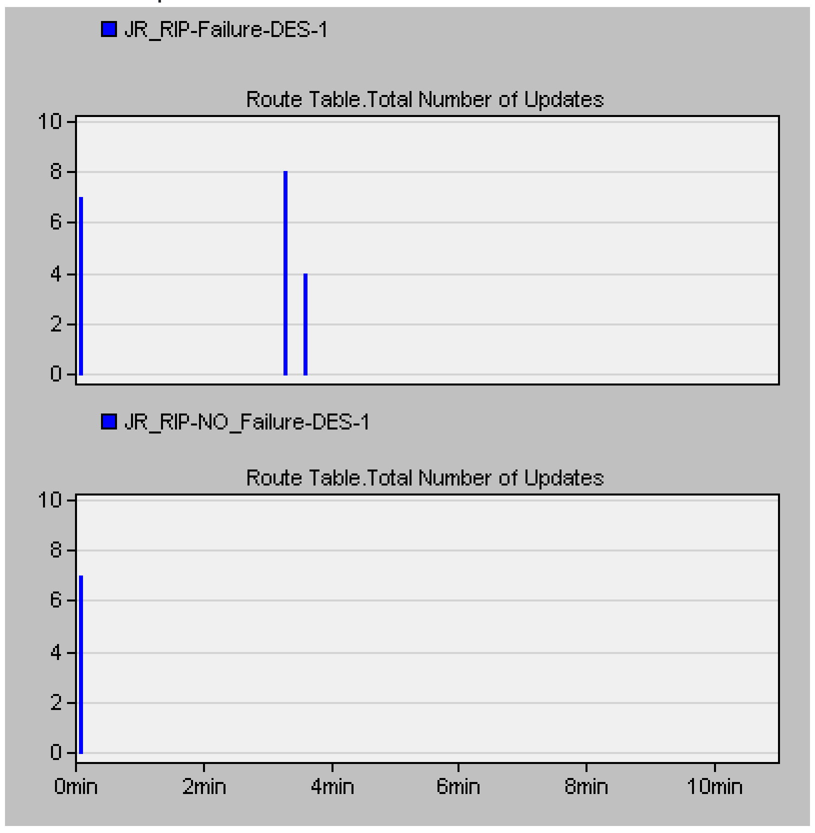
Attribute	Value
creation timestamp	13:34:06 Mar 18 2019
creation data	
label color	black
Failure/Recovery Modeling	Enabled
Link Failure/Recovery Specification	(...)
Number of Rows	1
Campus Network.Router1 <-> Router2	
Name	Campus Network.Router1 <-> Router2
Time (seconds)	200
Status	Fail
Link Failure/Recovery Specification File	NOT_USED
Node Failure Mode	Node Only
Node Failure/Recovery Specification	No Failure/Recovery
Node Failure/Recovery Specification File	NOT_USED
hostname	
minimized icon	circle/#708090

Extended Attrs.
Model Details
Object Documentation
Filter
Match: Exact [✓ Substring] RegEx
Look in: ✓ Names ✓ Values ✓ Possible values ✓ Tags
✓ Advanced
Apply to selected objects
OK
Cancel

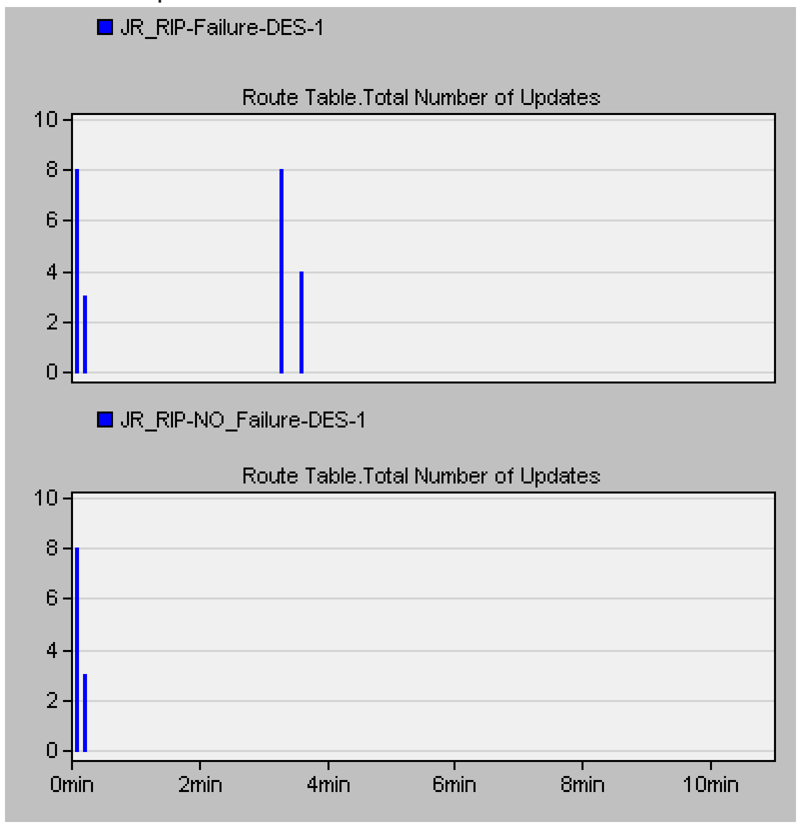
The simulation was then run and results we collected for both scenarios.

W19 Results

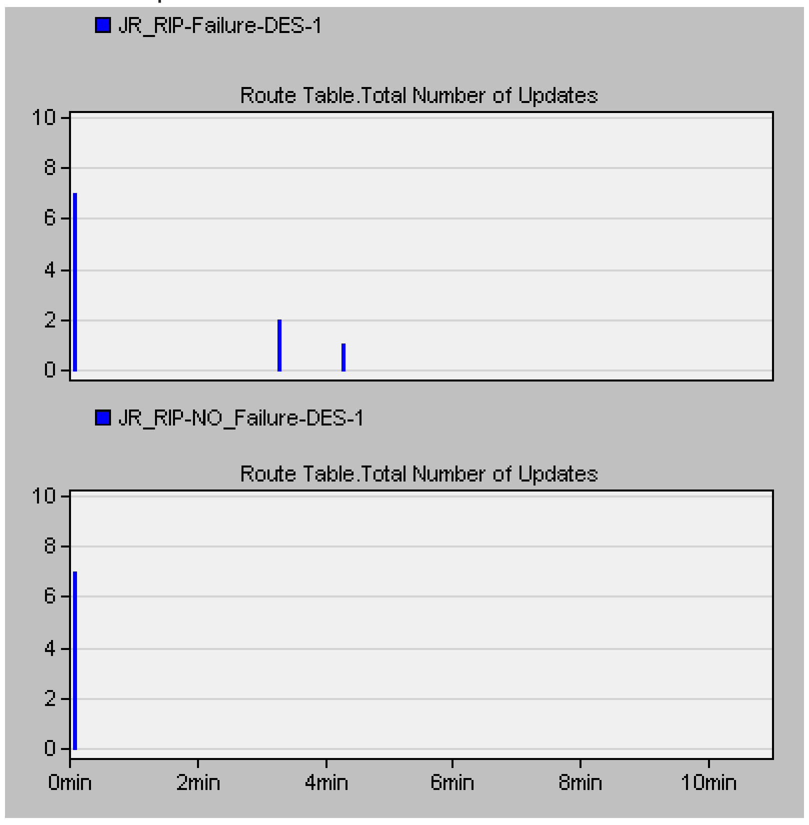
Router 1 Updates



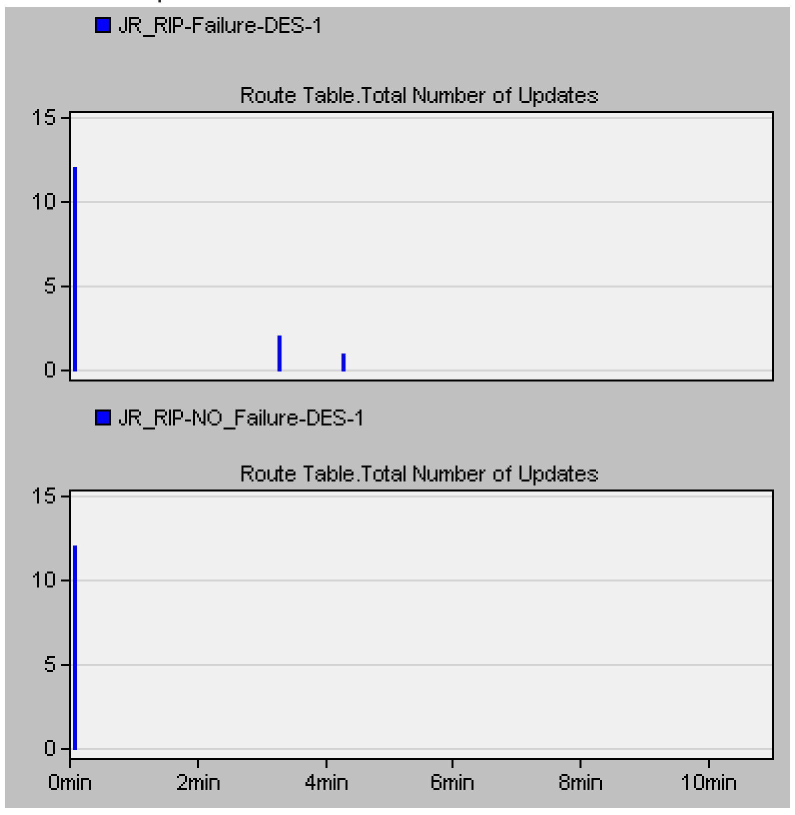
Router 2 Updates:



Router 3 Updates:



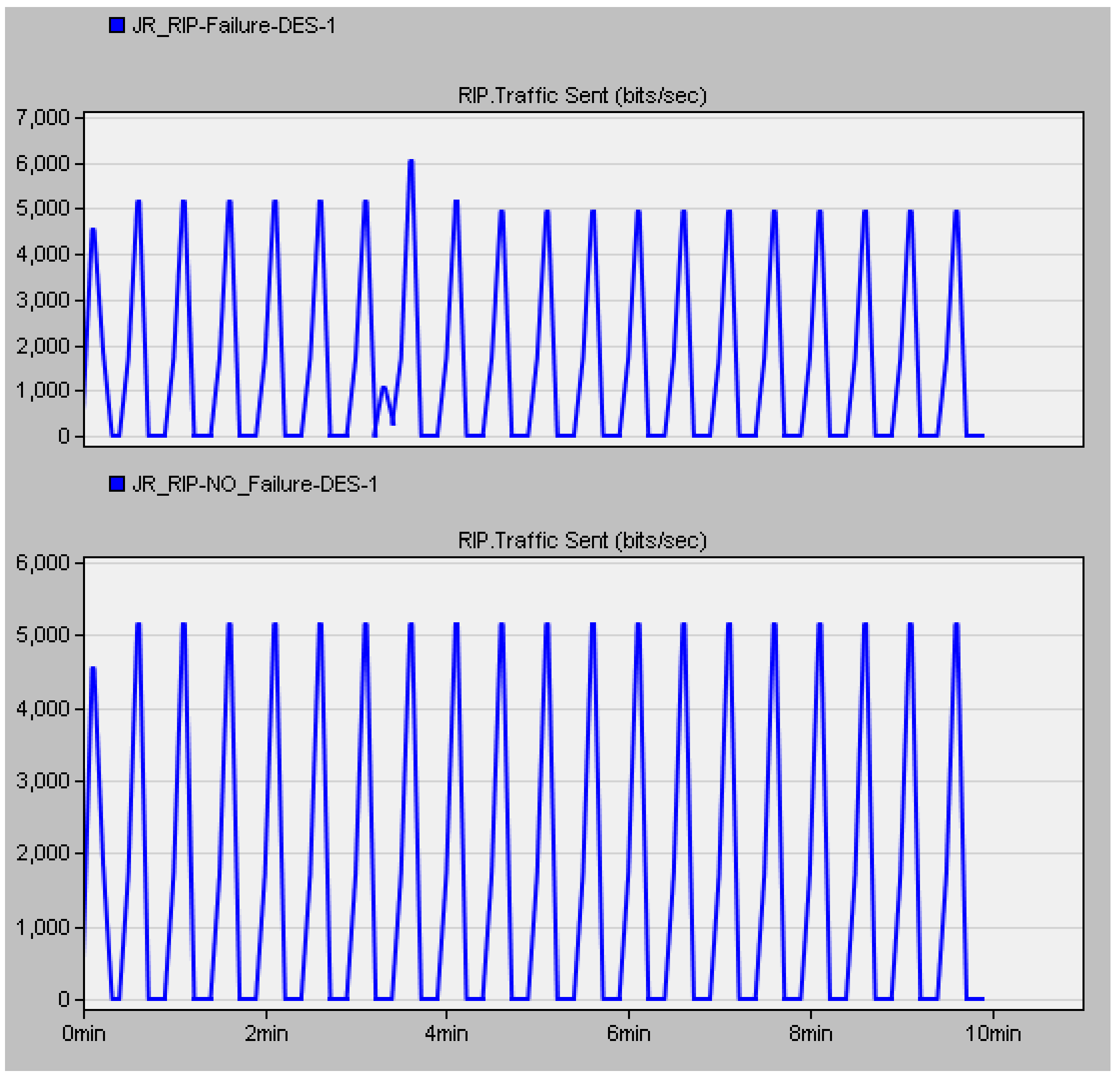
Router 4 Updates



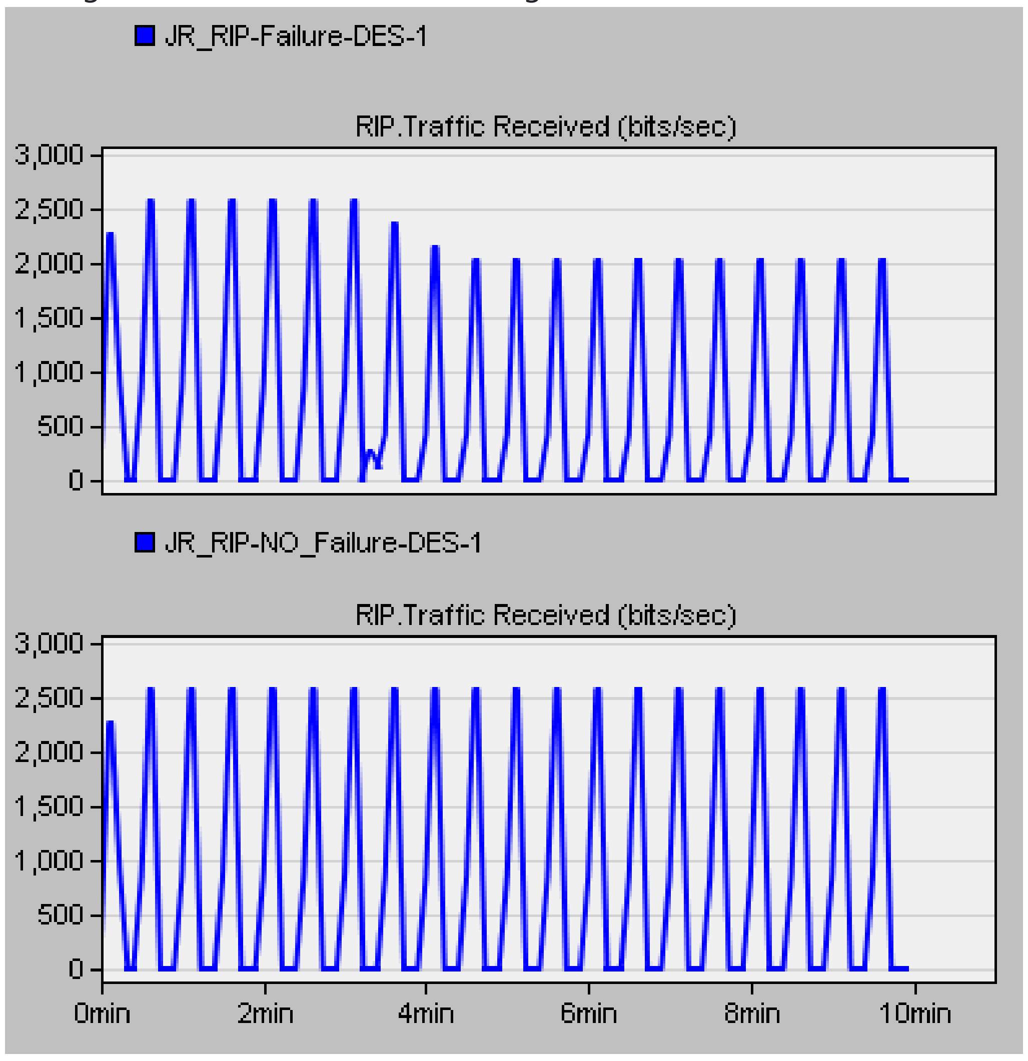
In the Failure scenario each of the routers send out Router Table Updates when the connection between Router1 and Router2 fails. Router1 and Router2 update the routing tables the same amount of times, at the same time as they are the routers directly involved with the connection that fails. Router3 and Router4 exhibit similar routing table updates as they are topologically similar with respect to the link that fails. The two bars which indicate routing table updates are for when (1) the routes using the failed connection are deemed invalid as the route timeout limit is reached and for when (2) the routes using the failed link are flushed/removed from the routing table. The period between the two updates is the holddown period used to ensure that neighbours are notified of the route becoming invalid.

Traffic Sent:

In the Failure scenario Traffic Sent experiences a short increase when the connection between Router1 and Router2 fails as the routing tables are updated to indicated to the routers on the network that there is no longer a direct connection between Router1 and Router2. This causes the spike in RIP traffic as the routing table update is propagated through the network.

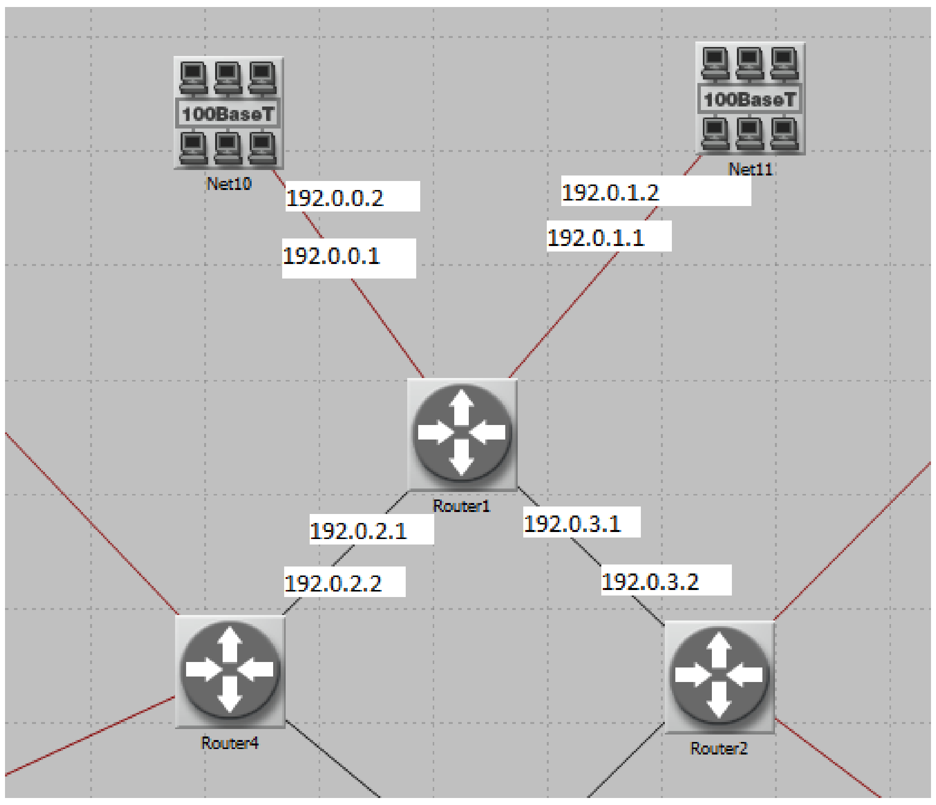


Traffic Received:
In the Failure scenario, the rate of traffic received experiences a permanent reduction as the network has one fewer line at which it can communicate. This means that due to the failure of the link between Router1 and Router2, data between those routers have to travel further, and through more routers before reaching its destination. This then reduces the overall rate at which RIP traffic can be received.



IP Addressing:

Interface IP Addresses Associated with Router1



Addresses Assigned to Each subnet
Subnet address found by doing the logical AND of the interface address with the subnet mask.

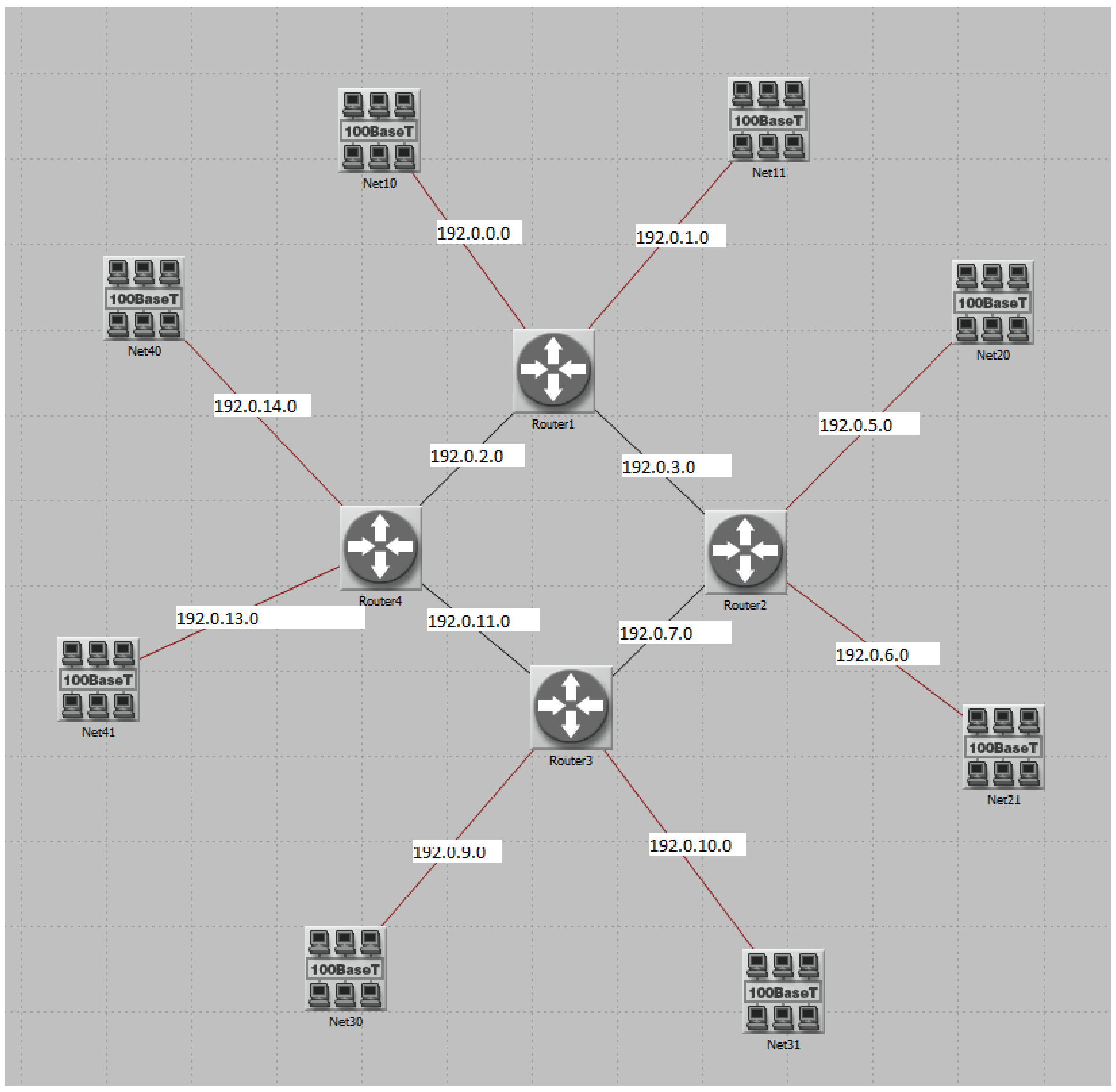
eg:

---

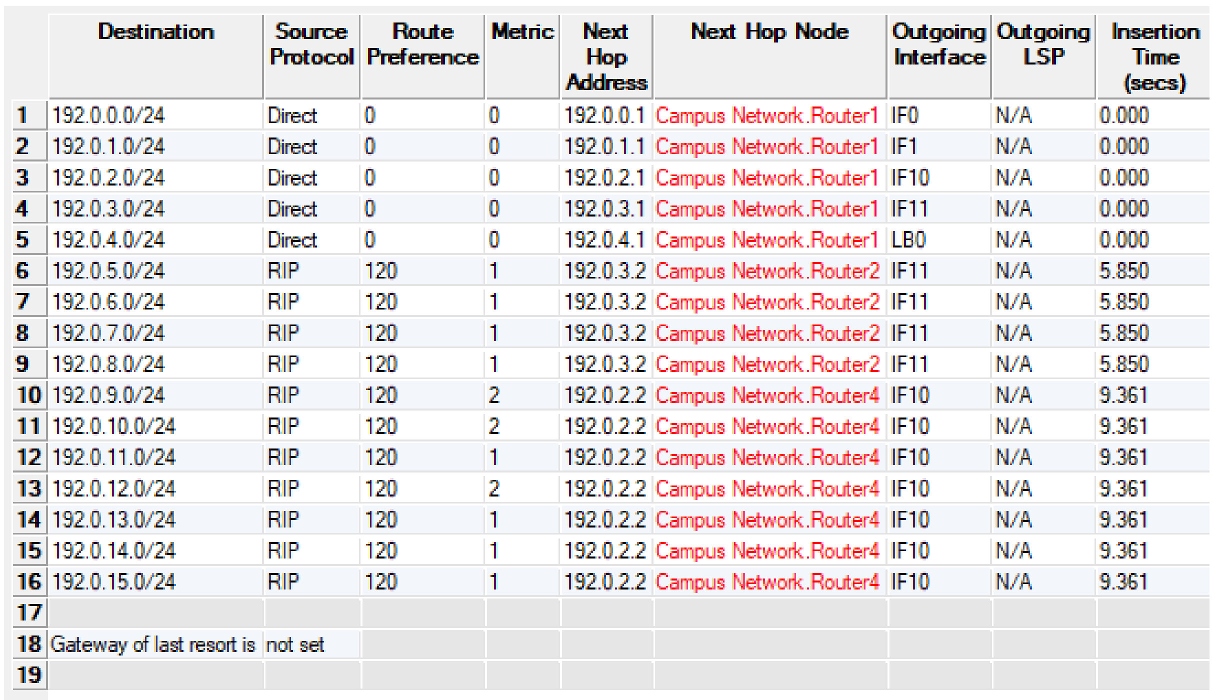
interface add = 192.0.14.1 = 11000000.00000000.00001110.00000001

subnet mask = 255.255.255.0 = 11111111.11111111.11111111.00000000

so, subnet add= 192.0.14.0 = 11000000.00000000.00001110.00000000

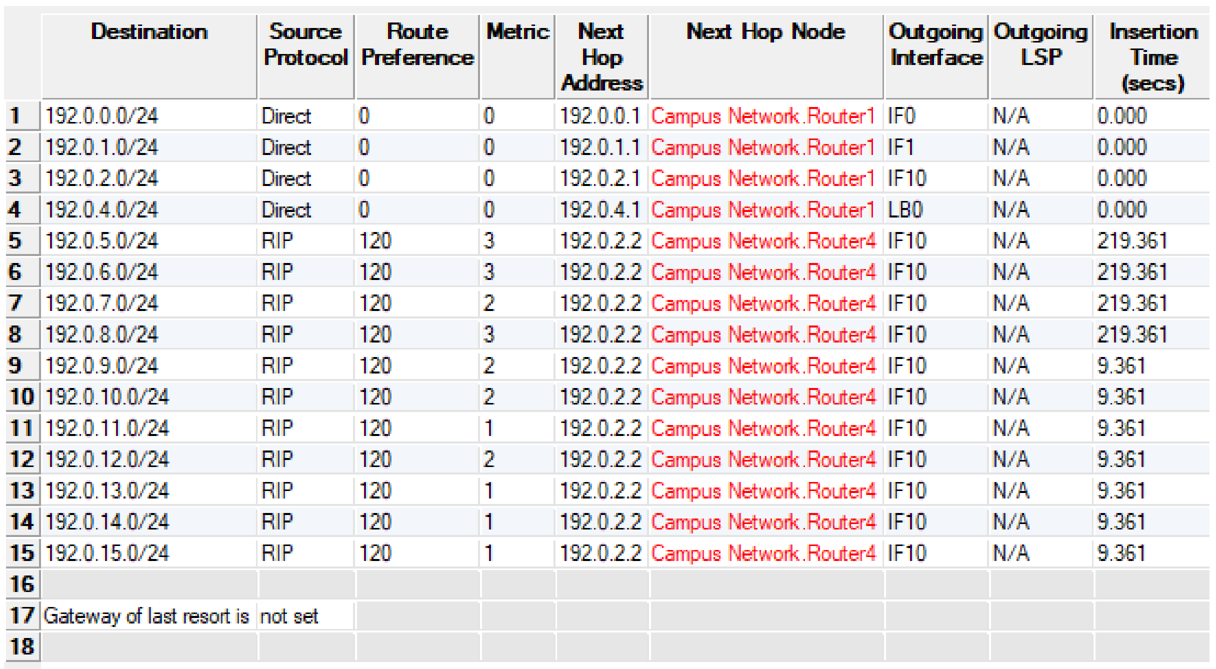


No Failure Router1 Routing Table:



	Destination	Source Protocol	Route Preference	Metric	Next Hop Address	Next Hop Node	Outgoing Interface	Outgoing LSP	Insertion Time (secs)
1	192.0.0.0/24	Direct	0	0	192.0.0.1	Campus Network.Router1	IF0	N/A	0.000
2	192.0.1.0/24	Direct	0	0	192.0.1.1	Campus Network.Router1	IF1	N/A	0.000
3	192.0.2.0/24	Direct	0	0	192.0.2.1	Campus Network.Router1	IF10	N/A	0.000
4	192.0.3.0/24	Direct	0	0	192.0.3.1	Campus Network.Router1	IF11	N/A	0.000
5	192.0.4.0/24	Direct	0	0	192.0.4.1	Campus Network.Router1	LB0	N/A	0.000
6	192.0.5.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	5.850
7	192.0.6.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	5.850
8	192.0.7.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	5.850
9	192.0.8.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	5.850
10	192.0.9.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
11	192.0.10.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
12	192.0.11.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
13	192.0.12.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
14	192.0.13.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
15	192.0.14.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
16	192.0.15.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
17									
18	Gateway of last resort is	not set							
19									

Failure Router1 Routing Table



	Destination	Source Protocol	Route Preference	Metric	Next Hop Address	Next Hop Node	Outgoing Interface	Outgoing LSP	Insertion Time (secs)
1	192.0.0.0/24	Direct	0	0	192.0.0.1	Campus Network.Router1	IF0	N/A	0.000
2	192.0.1.0/24	Direct	0	0	192.0.1.1	Campus Network.Router1	IF1	N/A	0.000
3	192.0.2.0/24	Direct	0	0	192.0.2.1	Campus Network.Router1	IF10	N/A	0.000
4	192.0.4.0/24	Direct	0	0	192.0.4.1	Campus Network.Router1	LB0	N/A	0.000
5	192.0.5.0/24	RIP	120	3	192.0.2.2	Campus Network.Router4	IF10	N/A	219.361
6	192.0.6.0/24	RIP	120	3	192.0.2.2	Campus Network.Router4	IF10	N/A	219.361
7	192.0.7.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	219.361
8	192.0.8.0/24	RIP	120	3	192.0.2.2	Campus Network.Router4	IF10	N/A	219.361
9	192.0.9.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
10	192.0.10.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
11	192.0.11.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
12	192.0.12.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
13	192.0.13.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
14	192.0.14.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
15	192.0.15.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
16									
17	Gateway of last resort is	not set							
18									

The effects of RIP can be seen by comparing the routing tables exported at the end of the simulation for the two scenarios.

W19 Questions

Question 1

Obtain and analyze the graphs that compare the sent RIP traffic for both scenarios. Make sure to change the draw style for the graphs to Bar Chart.

See Lab Report

Question 2

Describe and explain the effect of the failure of the link connecting Router1 to Router2 on the routing tables.

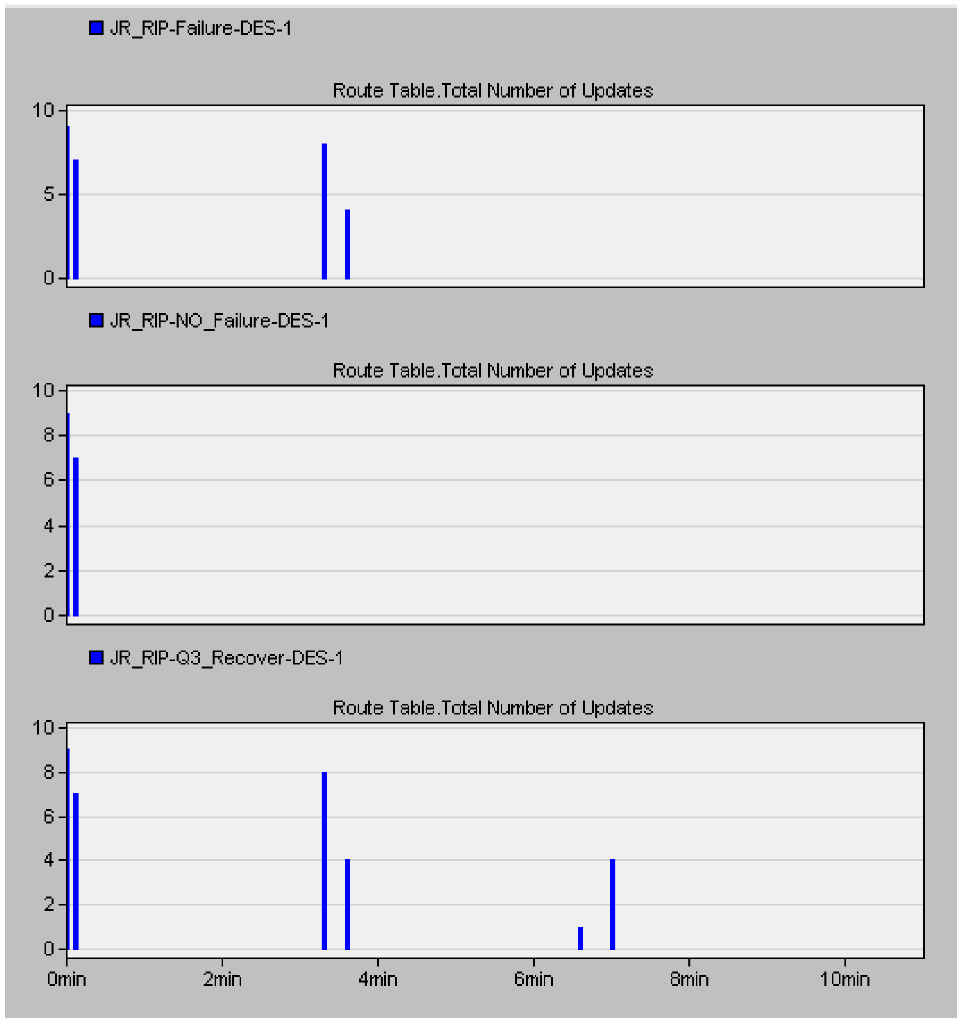
By looking at the Routing Tables for each scenario we can see that the link failure causes the metric from Router1 to Router2 and its LANs to increase by 2. This is because two additional connections need to be traversed from Router1 to Router2 (via Router4 and Router3) as the direct connection between Router 1 and Router2 is no longer available.

The effects of the link failure is further demonstrated by the fact that in the connection failure routing table, the routes between Router1 and Router2 and its LANs are inserted after ~219 seconds which is ~19 seconds after the connection fails, indicating that after a period after the connection has been identified as failed, a new route is generated.

Question 3

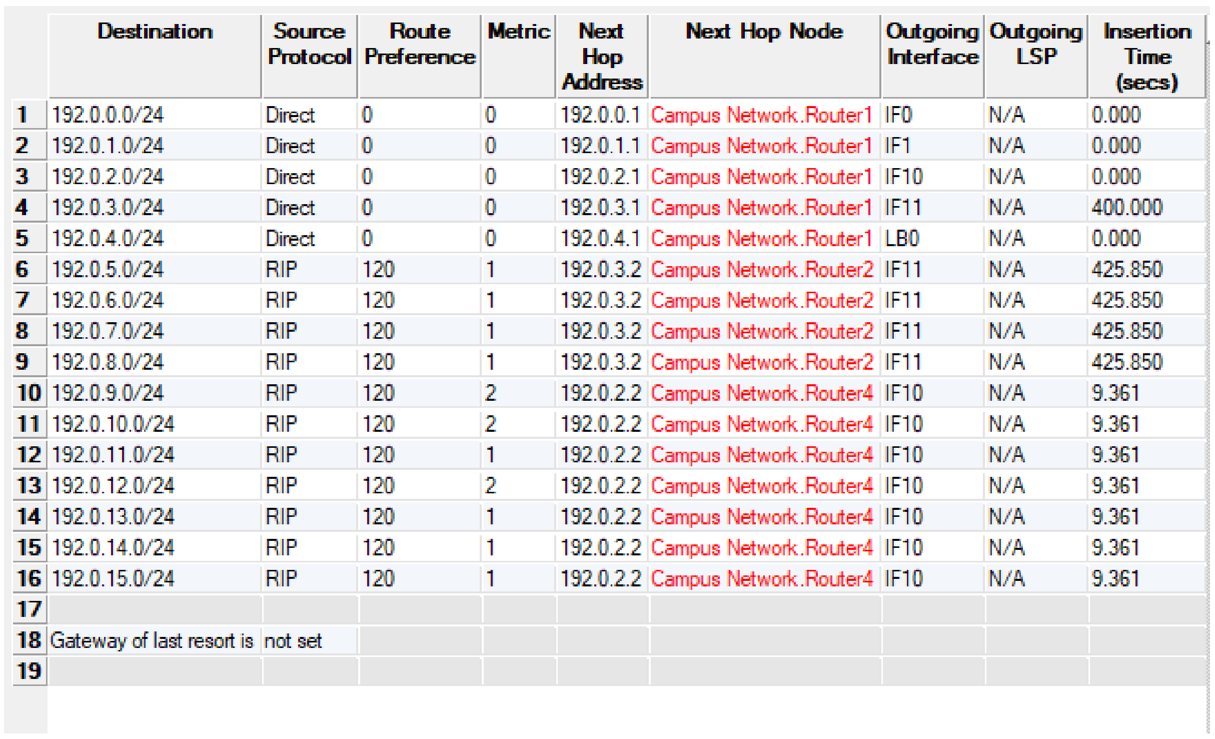
Create another scenario as a duplicate of the Failure scenario. Name the new scenario Q3_Recover. In this new scenario have the link connecting Router1 to Router2 recover after 400 seconds. Generate and analyze the graph that shows the effect of this recovery on the Total Number of Updates in the routing table of Router1. Check the contents of Router1's routing table. Compare this table with the corresponding routing tables generated in the NO_Failure and Failure scenarios.

Router 1 Updates:



From this graph we can see that Router1 initially has its tables updated once when the link is recovered. Then we can see that once new routes have been established with the nodes connected by Router2 we can see that Router1 get four more updates.

Router1 Routing Table At End of Recovery Simulation



	Destination	Source Protocol	Route Preference	Metric	Next Hop Address	Next Hop Node	Outgoing Interface	Outgoing LSP	Insertion Time (secs)
1	192.0.0.0/24	Direct	0	0	192.0.0.1	Campus Network.Router1	IF0	N/A	0.000
2	192.0.1.0/24	Direct	0	0	192.0.1.1	Campus Network.Router1	IF1	N/A	0.000
3	192.0.2.0/24	Direct	0	0	192.0.2.1	Campus Network.Router1	IF10	N/A	0.000
4	192.0.3.0/24	Direct	0	0	192.0.3.1	Campus Network.Router1	IF11	N/A	400.000
5	192.0.4.0/24	Direct	0	0	192.0.4.1	Campus Network.Router1	LB0	N/A	0.000
6	192.0.5.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	425.850
7	192.0.6.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	425.850
8	192.0.7.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	425.850
9	192.0.8.0/24	RIP	120	1	192.0.3.2	Campus Network.Router2	IF11	N/A	425.850
10	192.0.9.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
11	192.0.10.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
12	192.0.11.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
13	192.0.12.0/24	RIP	120	2	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
14	192.0.13.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
15	192.0.14.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
16	192.0.15.0/24	RIP	120	1	192.0.2.2	Campus Network.Router4	IF10	N/A	9.361
17									
18	Gateway of last resort is	not set							
19									

We can see that the routing table from the Recovery simulation has the same metric values as the routing table for the No Fail simulation. This indicates that the simulation successfully recovered the failed line between Router1 and Router2. This is further indicated by the fact that the routing table values for Router2 and its connected LANs in the Router1 table Inserted at ~425 seconds which is ~25 seconds after we configured to recover the link. This is expected as there would have been a delay period for when the router failed as it would have taken some time for the addresses of Router2's LAN nodes to propagate to Router1. Additionally we can see that by comparing the routing table from the Failure scenario, the same destinations that are updated when the link fails are the same destinations that are updated when the link is recovered (192.0.5.0, 192.0.6.0, 192.0.7.0, 192.0.8.0).

W19 Conclusion

From this practical we can conclude that RIP is an effective mechanism for redistributing routing information throughout a network, such that is a network link fails, it can be recovered and converged effectively. However the scenario we simulated described a somewhat smaller network, so the simulation may not be demonstrative of the efficacy of RIP working on a larger scale. We can also see the working of RIP at the level of the routing table, to the extent that we can see the significance the routing table has in maintaining routes between destinations, and how these routes may be updated.

Week 20 - OSPF, Open Shortest Path First

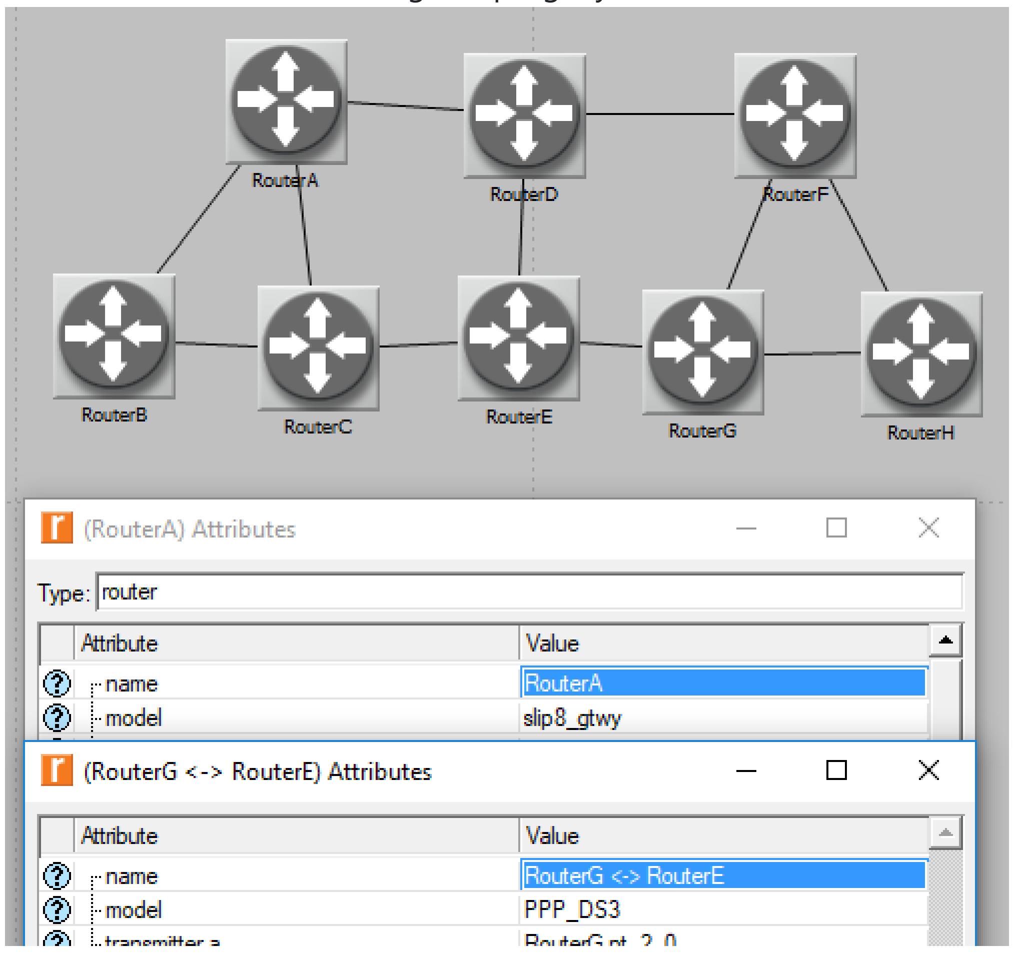
A Routing Protocol Based on the Link-State Algorithm

W20 Summary

In this practical session we simulated the function of the OSPF protocol on a connected network with metrics for each link. We explored this by evaluating the difference in the way the protocol functions when there are traffic demands by looking at scenarios where: no areas are used, areas are used, and load balancing is used. Then, by answering questions, we investigated further how OSPF functions.

W20 Implementation

For the No_Areas scenario eight slip8_gtwy routers connected with PPP_DS3 links were assembled in the form of the given topology.



(RouterA) Attributes

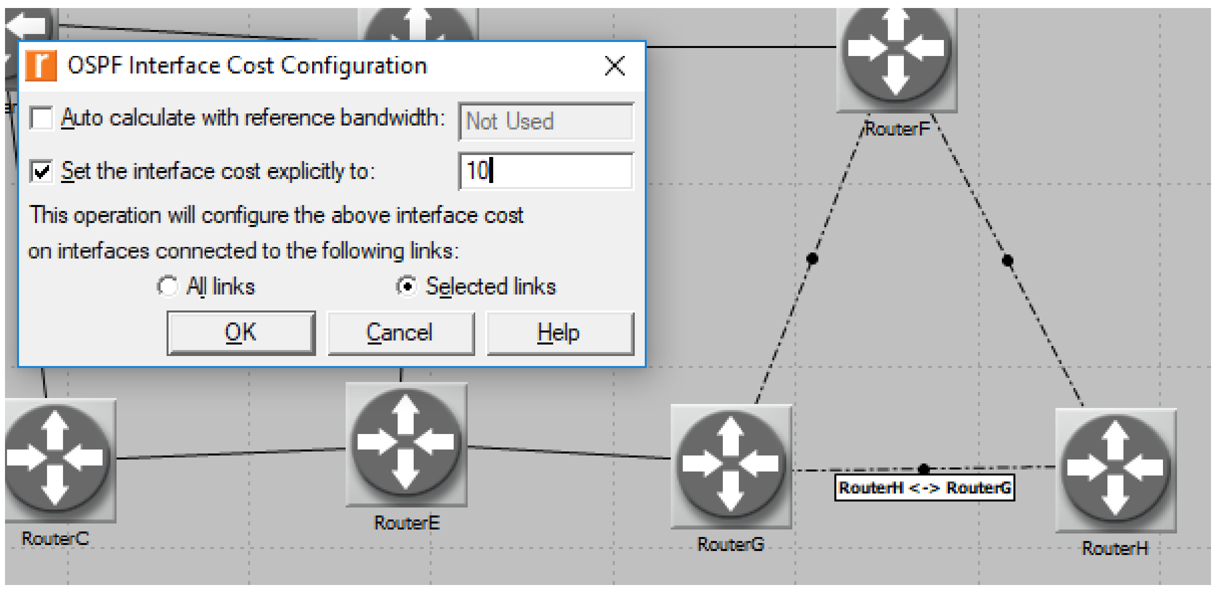
Type: router

Attribute	Value
name	RouterA
model	slip8_gtwy

(RouterG <-> RouterE) Attributes

Attribute	Value
name	RouterG <-> RouterE
model	PPP_DS3
transmitter a	RouterG.pt_2_0

The links' costs were then assigned to each of the links as specified.



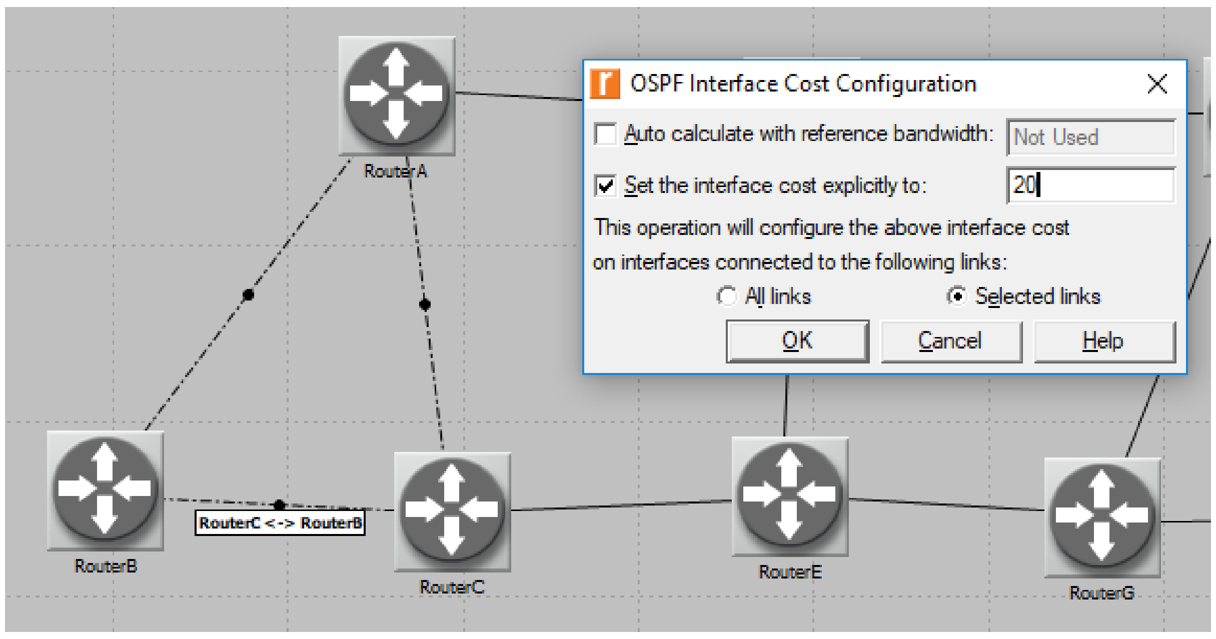
OSPF Interface Cost Configuration

☐ Auto calculate with reference bandwidth: Not Used

☑ Set the interface cost explicitly to: 10

This operation will configure the above interface cost on interfaces connected to the following links:

All links [✓ Selected links]



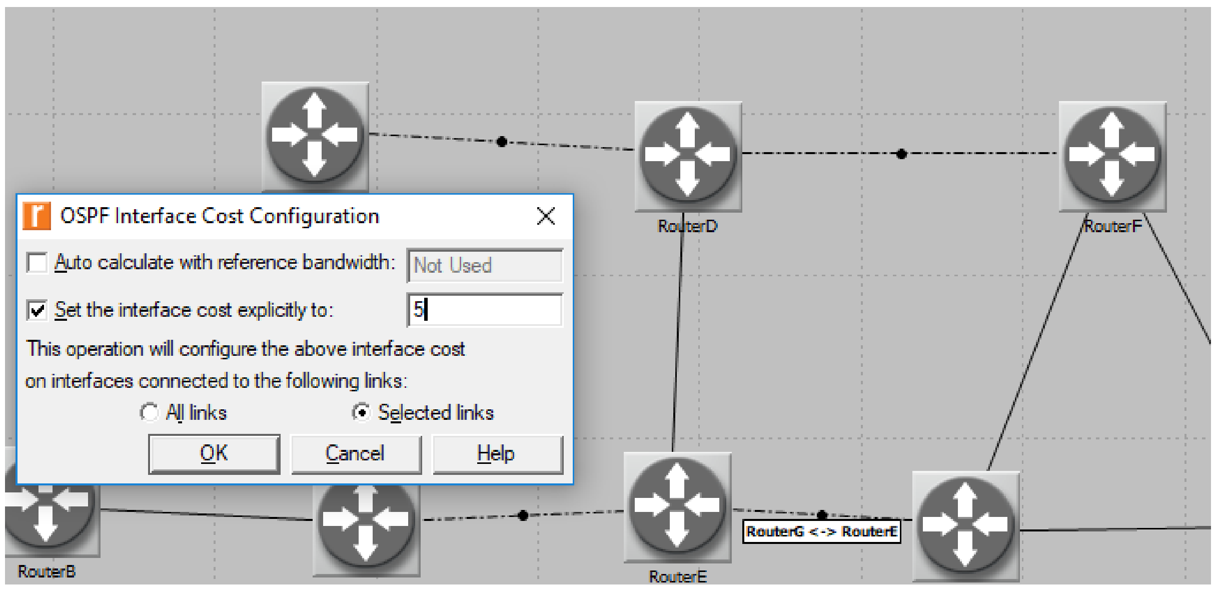
OSPF Interface Cost Configuration

☐ Auto calculate with reference bandwidth: Not Used

☑ Set the interface cost explicitly to: 20

This operation will configure the above interface cost on interfaces connected to the following links:

All links [✓ Selected links]


OSPF Interface Cost Configuration

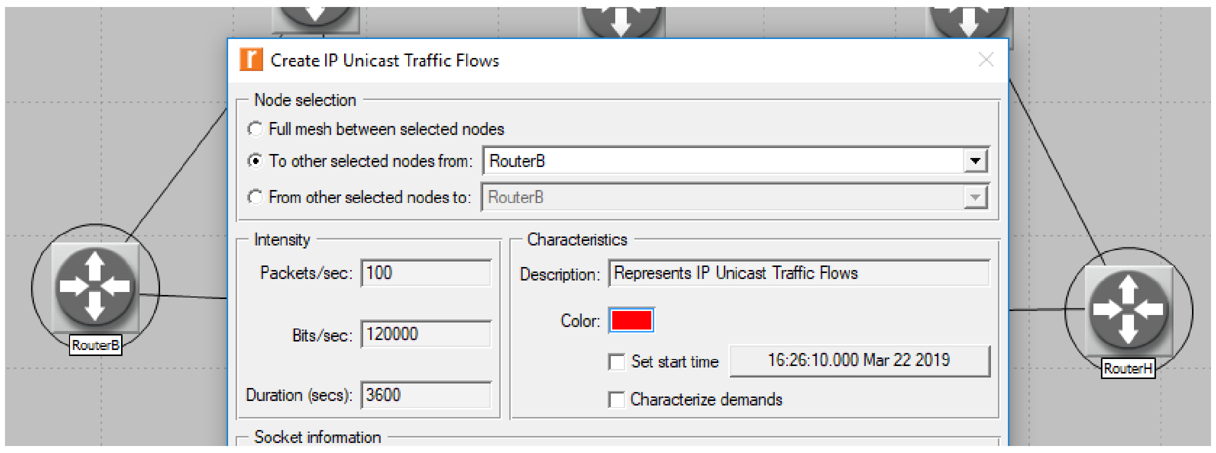
- [ ] Auto calculate with reference bandwidth: Not Used
- [✓] Set the interface cost explicitly to: 5

This operation will configure the above interface cost on interfaces connected to the following links:

All links [✓ Selected links]

OK Cancel Help

The Traffic Demands were then configured.

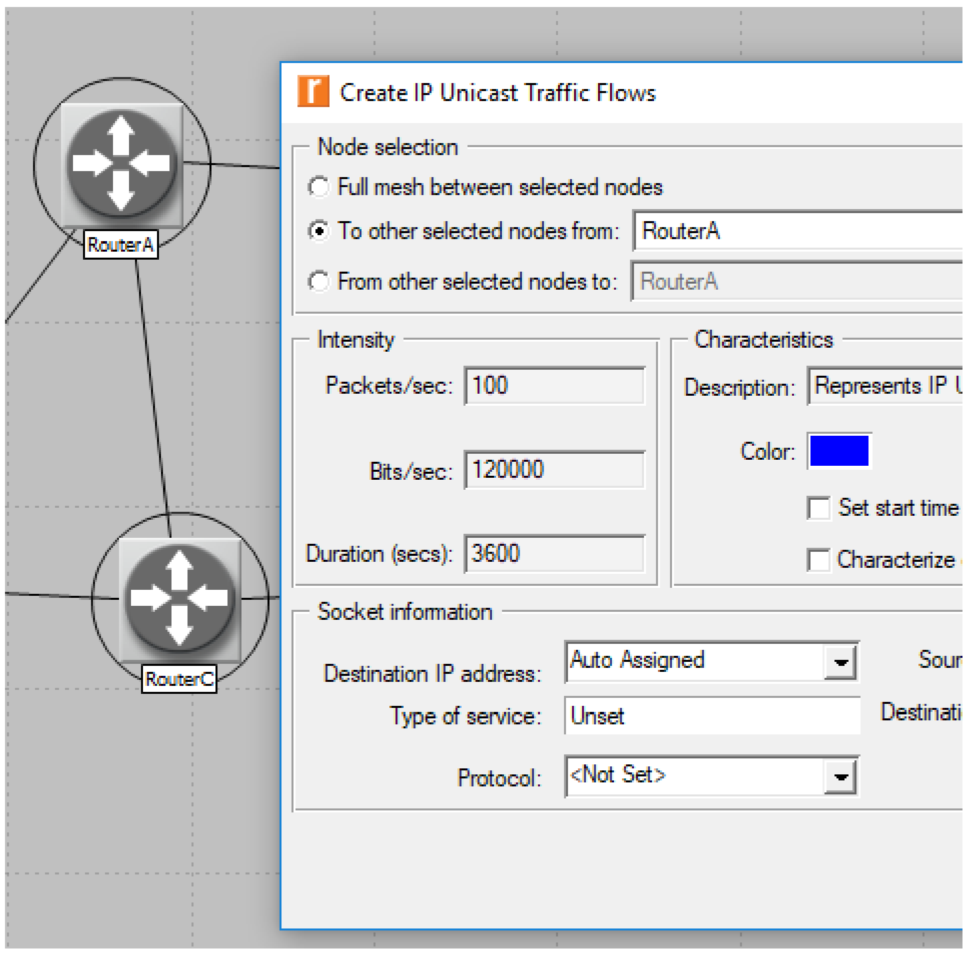

Create IP Unicast Traffic Flows

Node selection
- Full mesh between selected nodes
- [✓] To other selected nodes from: RouterB
- From other selected nodes to: RouterB

Intensity
- Packets/sec: 100
- Bits/sec: 120000
- Duration (secs): 3600

Characteristics
- Description: Represents IP Unicast Traffic Flows
- Color: 🟥
- [ ] Set start time 16:26:10.000 Mar 22 2019
- [ ] Characterize demands

Socket information


Create IP Unicast Traffic Flows

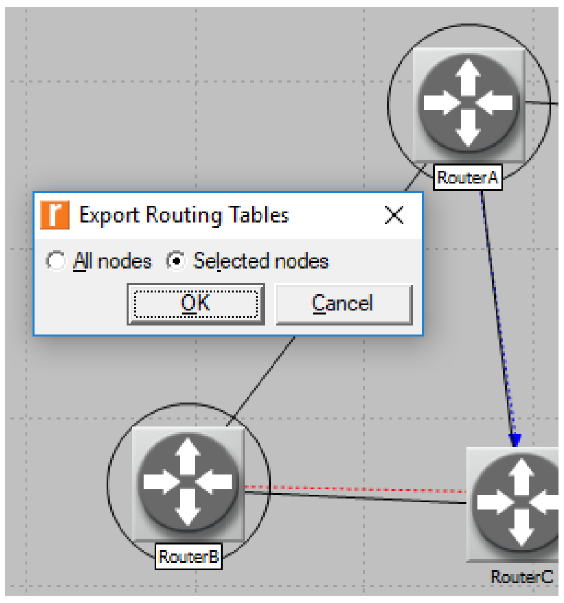
Node selection
- Full mesh between selected nodes
- [✓] To other selected nodes from: RouterA
- From other selected nodes to: RouterA

Intensity
- Packets/sec: 100
- Bits/sec: 120000
- Duration (secs): 3600

Characteristics
- Description: Represents IP U
- Color: 🟦
- [ ] Set start time
- [ ] Characterize

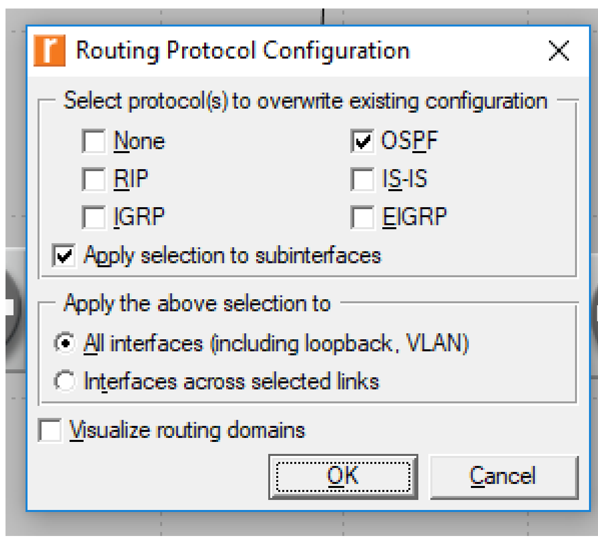
Socket information
- Destination IP address: Auto Assigned
- Type of service: Unset
- Protocol: <Not Set>
- Sour
- Destinati

The routing protocol and addresses setting were then configured, such that OSPF is selected as the protocol and IP addresses were auto-assigned.


Export Routing Tables

All nodes [✓ Selected nodes]

OK Cancel



Routing Protocol Configuration

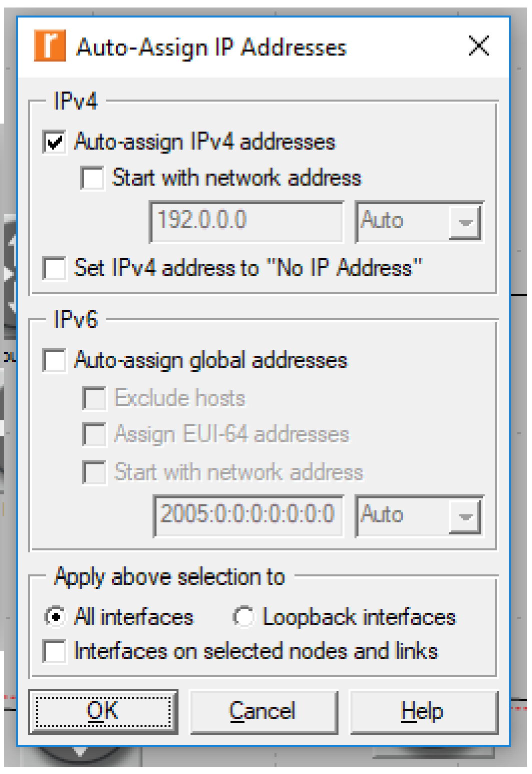
Select protocol(s) to overwrite existing configuration

- [ ] None
- [x] OSPF
- [ ] RIP
- [ ] IS-IS
- [ ] IGRP
- [ ] EIGRP
- [x] Apply selection to subinterfaces

Apply the above selection to

- [✓ All interfaces (including loopback, VLAN)]
- Interfaces across selected links

- [ ] Visualize routing domains



Auto-Assign IP Addresses

IPv4

- [x] Auto-assign IPv4 addresses
- [ ] Start with network address: 192.0.0.0 Auto
- [ ] Set IPv4 address to "No IP Address"

IPv6

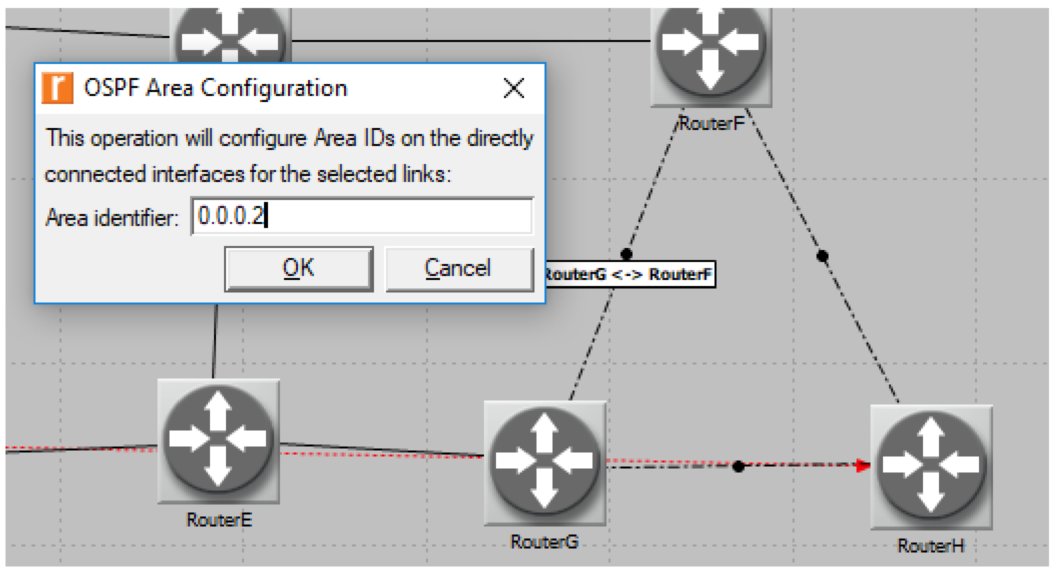
- [ ] Auto-assign global addresses
- [ ] Exclude hosts
- [ ] Assign EUI-64 addresses
- [ ] Start with network address: 2005:0:0:0:0:0:0:0 Auto

Apply above selection to

- [✓ All interfaces] Loopback interfaces
- [ ] Interfaces on selected nodes and links

The simulation is configured to run for 10 minutes then ran.

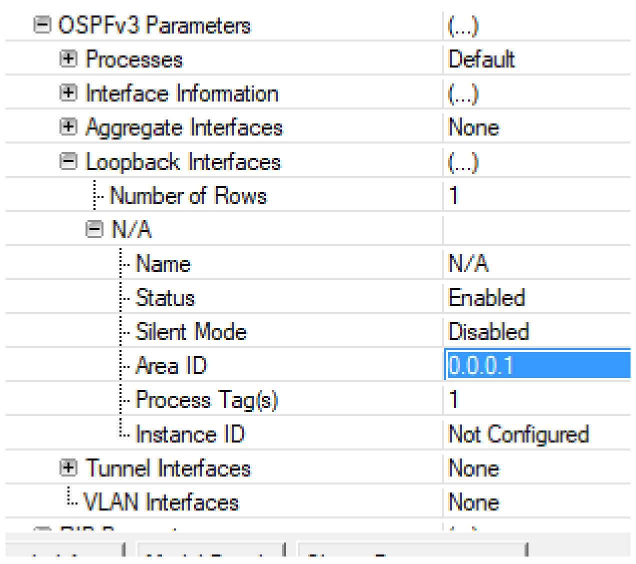
The scenario is then duplicated to consider the use of Areas. This is done by assigning Router A, B, and C to under the Area Identifier 0.0.0.1 and for Routers F, G, and H to be under the Area Identifier 0.0.0.2. This means that the area that was not configured (Routers D and E) are under the Area Identifier 0.0.0.0.



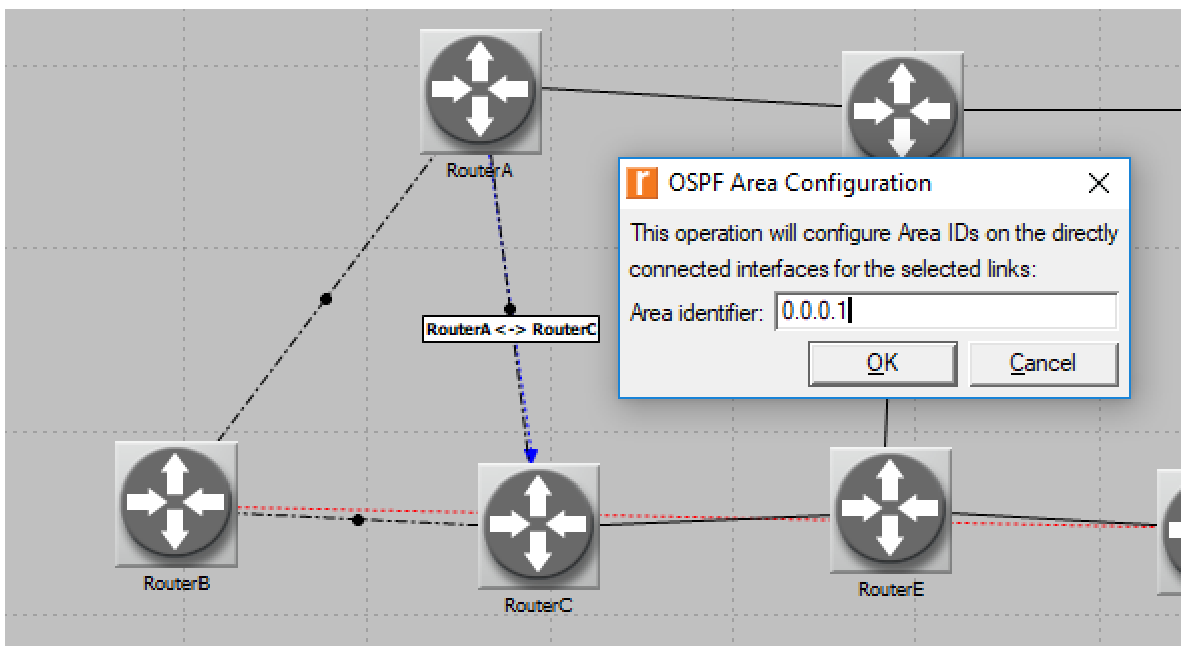
OSPF Area Configuration

This operation will configure Area IDs on the directly connected interfaces for the selected links:

Area identifier: 0.0.0.2



⊟ OSPFv3 Parameters	(...)
⊞ Processes	Default
⊞ Interface Information	(...)
⊞ Aggregate Interfaces	None
⊟ Loopback Interfaces	(...)
Number of Rows	1
⊟ N/A	
Name	N/A
Status	Enabled
Silent Mode	Disabled
Area ID	0.0.0.1
Process Tag(s)	1
Instance ID	Not Configured
⊞ Tunnel Interfaces	None
VLAN Interfaces	None

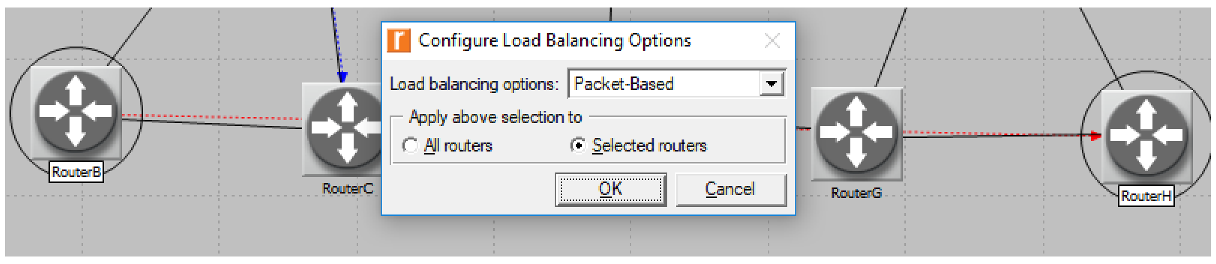


OSPF Area Configuration

This operation will configure Area IDs on the directly connected interfaces for the selected links:

Area identifier: 0.0.0.1

The Balanced Scenario is then configured to balance traffic across different routes between two routers. This is done by the Packet-Based load balancing option when Routers B and H are selected.


Configure Load Balancing Options

Load balancing options: Packet-Based

Apply above selection to: All routers [✓ Selected routers]

OK Cancel

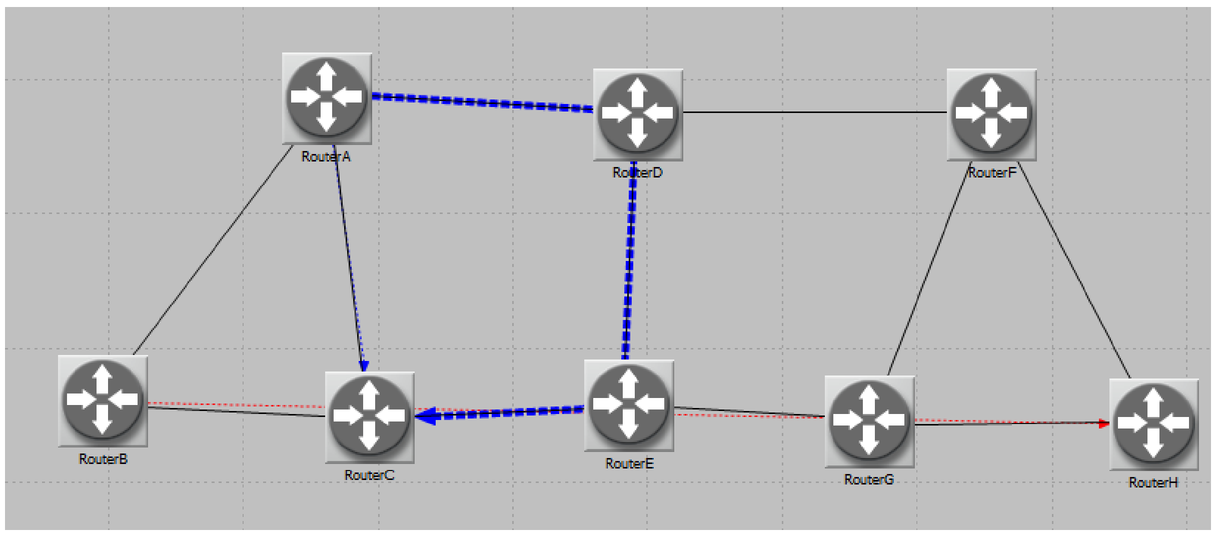
Results are then gathered from the results browser.

W20 Results

No Areas Scenario

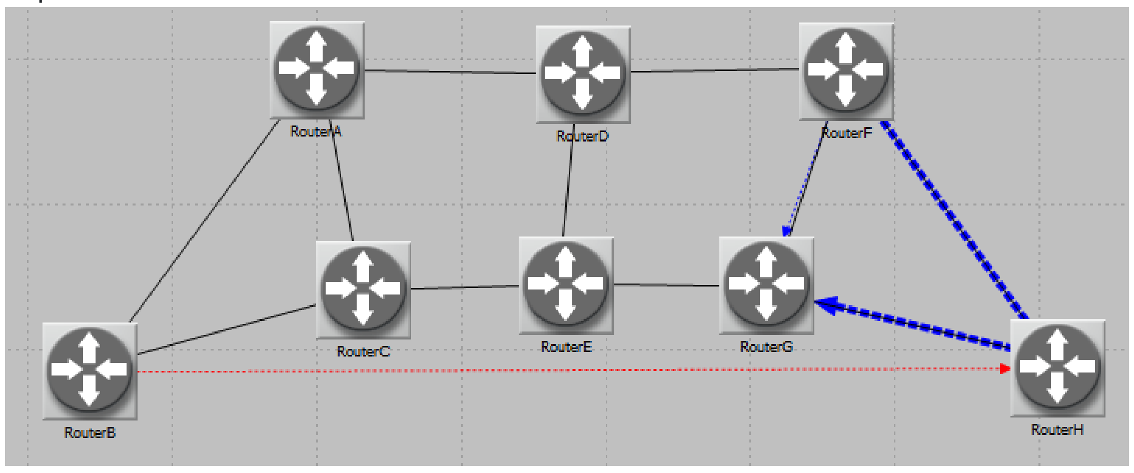
Route for Traffic Demand Between Router A and Router C:

Without areas the shortest path for the traffic is chosen, with a metric of 15. This result was expected.



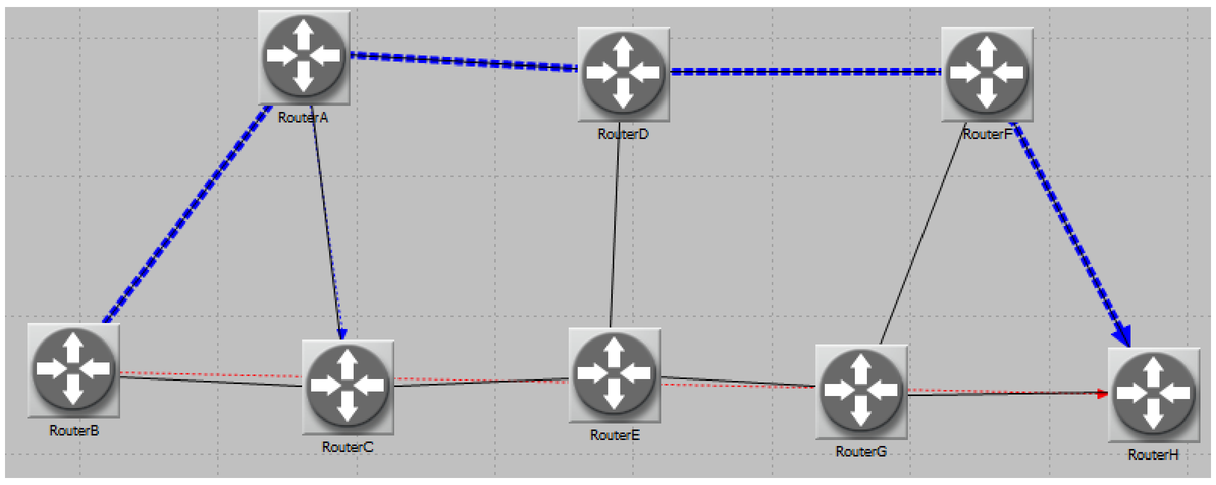
Route for Traffic Demand Between Router F and Router F:

This traffic demand was also implemented as the No Areas scenario was not effective in showing the change in route from A to C. This result was expected.



Route for Traffic Demand Between Router B and Router H:

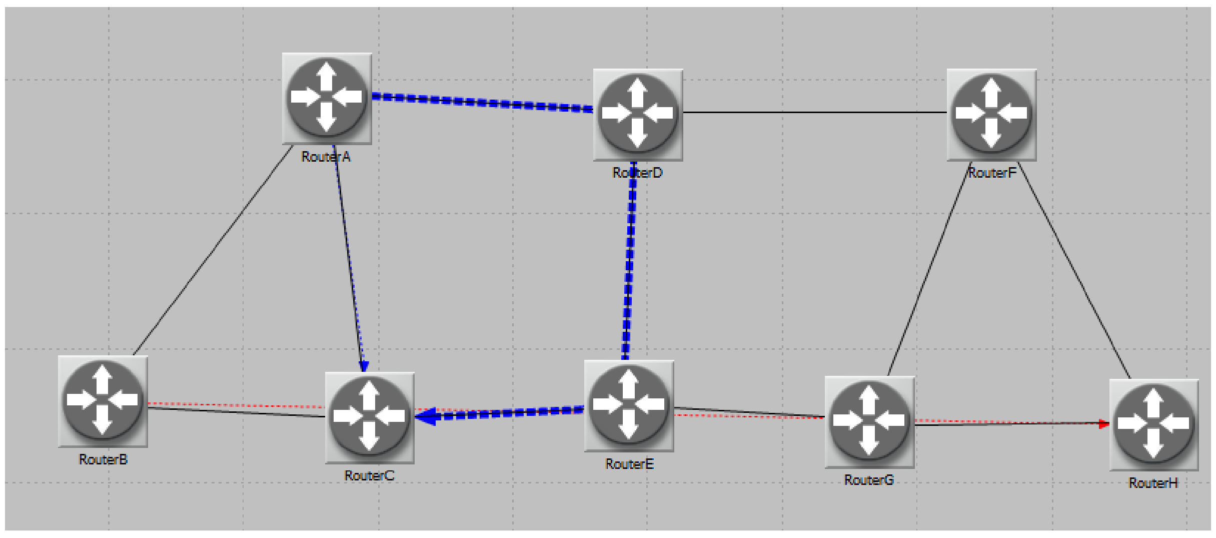
This result was expected, but the route could have also been B>C>E>G>H as it would have of equal metric.



Areas Scenario

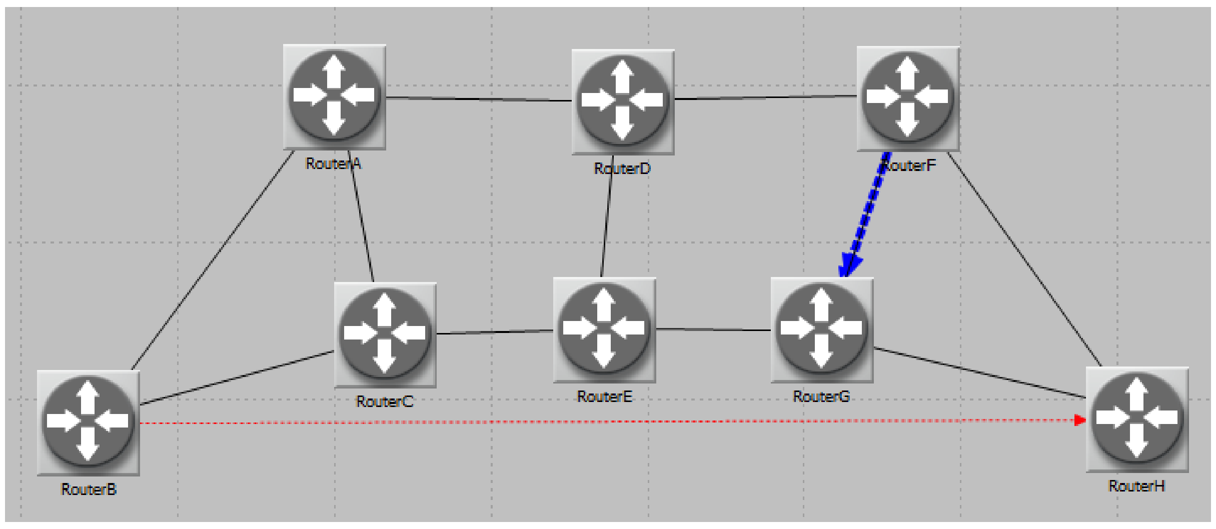
Route for Traffic Demand Between Router A and Router C:

Expected results were not achieved. No difference from the scenario without areas.



Route for Traffic Demand Between Router F and Router G:

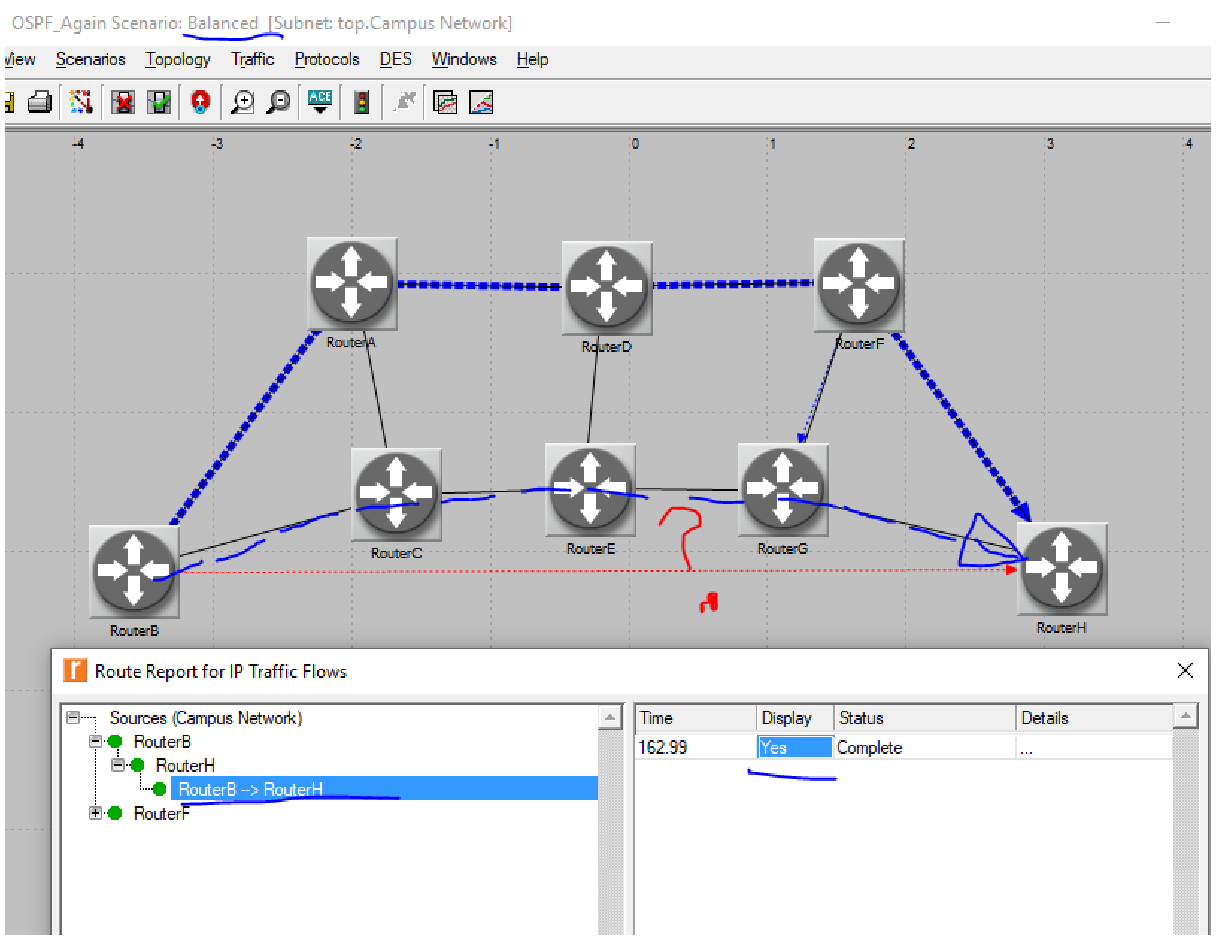
Expected result was achieved, as in the areas scenario the Route from F to H changes to follow the direct link.



Balanced Scenario

Route for Traffic Demand Between Router B and Router H

Expected results were not achieved. Expected results were supposed to demonstrate that the traffic flows through both B>A>D>F>H and B>C>E>G>H.


OSPF_Again Scenario: Balanced [Subnet: top.Campus Network]
View Scenarios Topology Traffic Protocols DES Windows Help
RouterA
RouterD
RouterF
RouterC
RouterE
RouterG
RouterB
RouterH
?
Route Report for IP Traffic Flows
Sources (Campus Network)
RouterB
RouterH
RouterB --> RouterH
RouterF

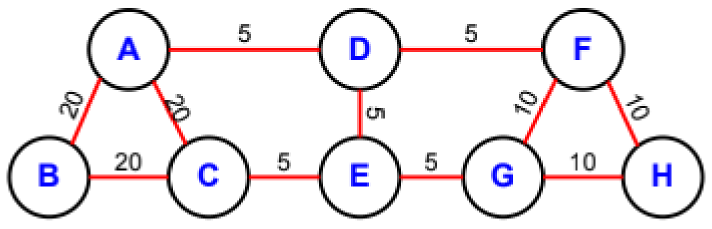
Time	Display	Status	Details
162.99	Yes	Complete	...

W20 Questions

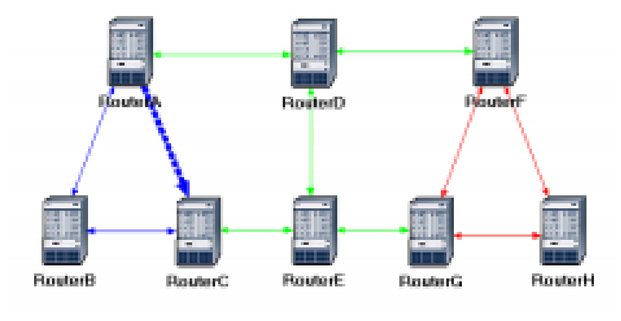
"Questions 3 and 4 are required for Marking (Summative assessment)"

1 . Explain Differences In Areas, Balanced And No_areas Scenarios

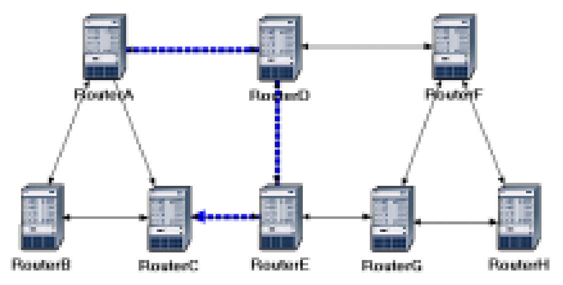
Using the diagrams below, explain why, for the same pair of routers, the Areas and Balanced scenarios result in different routes than those observed in the No_Areas scenario.



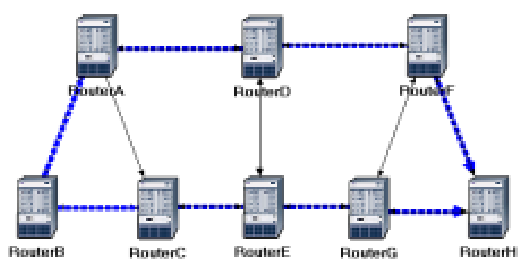
Areas



No_Areas



Balanced



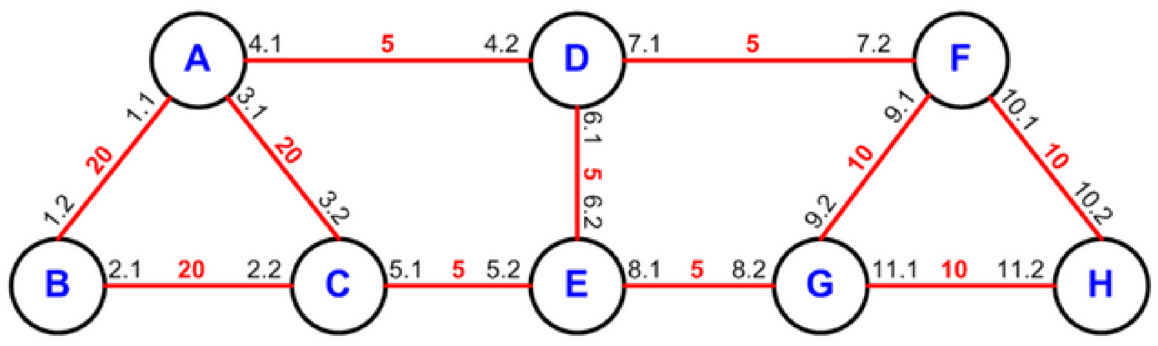
ANSWER:

For the same routers (A and C) in the Areas and No_Areas scenario, different routes are gained as on the Areas scenario Router A and C are in the same area, so the lowest metric route from A to C is the highlighted route of metric 20. However, in the No_Areas scenario the route chosen is under no constraint to stay within the same area, such that the shortest path from A to C is A->D->E->C with a total metric of 15.

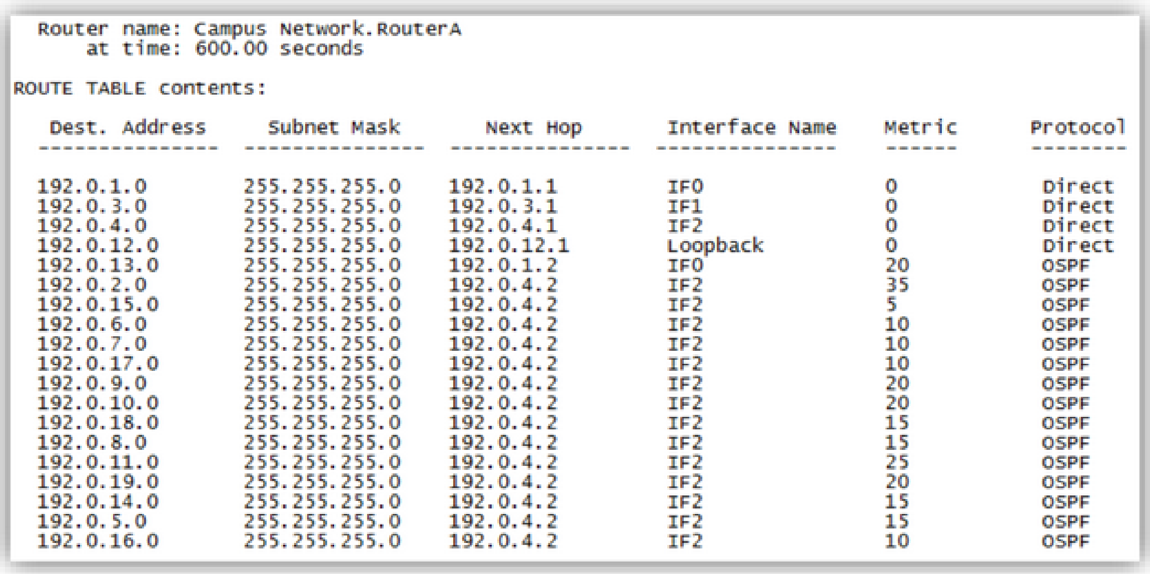
When load balancing is not used, only one route from B to H is highlighted as the shortest route. However, when load balancing is implemented both routes B->A->D->F->H and B->C->E->G->H are selected for communication as they both have a metric of 40. By splitting traffic across these two routes the traffic is balanced.

2. Explain Changes To Metric Value In Routing Table

Using the diagram and routing tables for Router A for all three scenarios (given below), explain any changes to the values assigned to the metric column in the routing tables for each route.

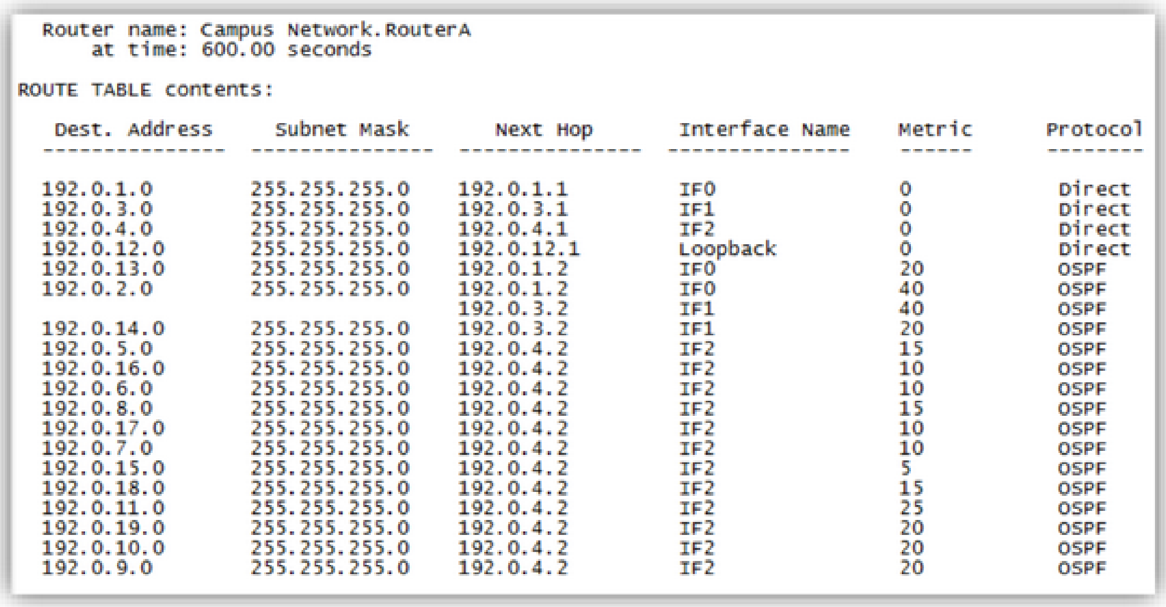


No_Areas scenario: Routing table of **RouterA**.



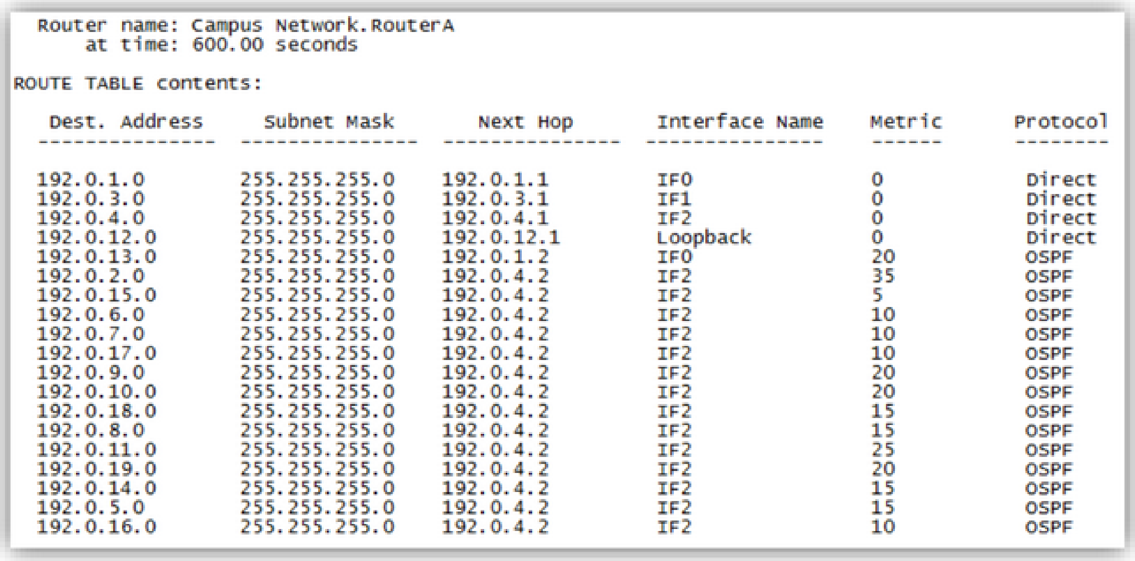
```
  Router name: Campus Network.RouterA
      at time: 600.00 seconds

ROUTE TABLE contents:

   Dest. Address     Subnet Mask       Next Hop         Interface Name     Metric      Protocol
  ---------------   ---------------   ---------------   ---------------    ------      --------

  192.0.1.0         255.255.255.0     192.0.1.1         IF0                0            Direct
  192.0.3.0         255.255.255.0     192.0.3.1         IF1                0            Direct
  192.0.4.0         255.255.255.0     192.0.4.1         IF2                0            Direct
  192.0.12.0        255.255.255.0     192.0.12.1        Loopback           0            Direct
  192.0.13.0        255.255.255.0     192.0.1.2         IF0                20           OSPF
  192.0.2.0         255.255.255.0     192.0.4.2         IF2                35           OSPF
  192.0.15.0        255.255.255.0     192.0.4.2         IF2                5            OSPF
  192.0.6.0         255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.7.0         255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.17.0        255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.9.0         255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.10.0        255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.18.0        255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.8.0         255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.11.0        255.255.255.0     192.0.4.2         IF2                25           OSPF
  192.0.19.0        255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.14.0        255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.5.0         255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.16.0        255.255.255.0     192.0.4.2         IF2                10           OSPF
```

Areas scenario: Routing table of **RouterA**



```
  Router name: Campus Network.RouterA
      at time: 600.00 seconds

ROUTE TABLE contents:

   Dest. Address     Subnet Mask       Next Hop         Interface Name     Metric      Protocol
  ---------------   ---------------   ---------------   ---------------    ------      --------

  192.0.1.0         255.255.255.0     192.0.1.1         IF0                0            Direct
  192.0.3.0         255.255.255.0     192.0.3.1         IF1                0            Direct
  192.0.4.0         255.255.255.0     192.0.4.1         IF2                0            Direct
  192.0.12.0        255.255.255.0     192.0.12.1        Loopback           0            Direct
  192.0.13.0        255.255.255.0     192.0.1.2         IF0                20           OSPF
  192.0.2.0         255.255.255.0     192.0.1.2         IF0                40           OSPF
                                      192.0.3.2         IF1                40           OSPF
  192.0.14.0        255.255.255.0     192.0.3.2         IF1                20           OSPF
  192.0.5.0         255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.16.0        255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.6.0         255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.8.0         255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.17.0        255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.7.0         255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.15.0        255.255.255.0     192.0.4.2         IF2                5            OSPF
  192.0.18.0        255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.11.0        255.255.255.0     192.0.4.2         IF2                25           OSPF
  192.0.19.0        255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.10.0        255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.9.0         255.255.255.0     192.0.4.2         IF2                20           OSPF
```

Balanced scenario: Routing table of **RouterA**



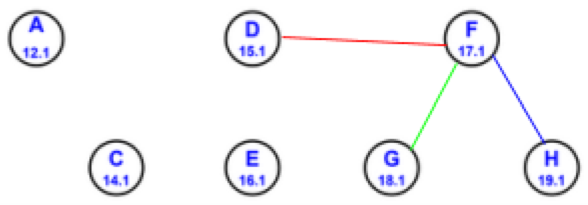
```
  Router name: Campus Network.RouterA
      at time: 600.00 seconds

ROUTE TABLE contents:

   Dest. Address     Subnet Mask       Next Hop         Interface Name     Metric      Protocol
  ---------------   ---------------   ---------------   ---------------    ------      --------

  192.0.1.0         255.255.255.0     192.0.1.1         IF0                0            Direct
  192.0.3.0         255.255.255.0     192.0.3.1         IF1                0            Direct
  192.0.4.0         255.255.255.0     192.0.4.1         IF2                0            Direct
  192.0.12.0        255.255.255.0     192.0.12.1        Loopback           0            Direct
  192.0.13.0        255.255.255.0     192.0.1.2         IF0                20           OSPF
  192.0.2.0         255.255.255.0     192.0.4.2         IF2                35           OSPF
  192.0.15.0        255.255.255.0     192.0.4.2         IF2                5            OSPF
  192.0.6.0         255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.7.0         255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.17.0        255.255.255.0     192.0.4.2         IF2                10           OSPF
  192.0.9.0         255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.10.0        255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.18.0        255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.8.0         255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.11.0        255.255.255.0     192.0.4.2         IF2                25           OSPF
  192.0.19.0        255.255.255.0     192.0.4.2         IF2                20           OSPF
  192.0.14.0        255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.5.0         255.255.255.0     192.0.4.2         IF2                15           OSPF
  192.0.16.0        255.255.255.0     192.0.4.2         IF2                10           OSPF
```

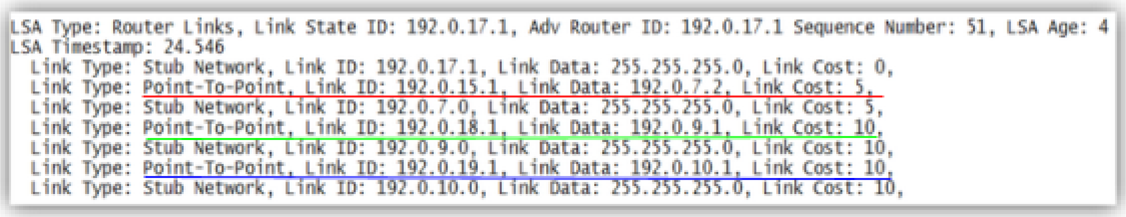
ANSWER:

The metric value in the routing tables for some destinations increase when using areas, as there are routes that become off limits due to links leading to routers that are not in the same area. For example, the metric from Router A to 192.0.2.0 increases from 35 to 40. The Balanced scenario has no impact on the metric value in the Routing table as The balanced configuration does not impose any new preferences on a route between routers.

3. Explain How Router A Utilizes The Information To Create A Map For The Topology Of The Network.

The diagram below is a snapshot of the link state database for Router A with information about another router on the network. Using the IP addresses for each Router, explain how Router A utilizes the information to create a map for the topology of the network. Router IP addresses (note the 192.0 part of the IP address has been omitted).





```
LSA Type: Router Links, Link State ID: 192.0.17.1, Adv Router ID: 192.0.17.1 Sequence Number: 51, LSA Age: 4
LSA Timestamp: 24.546
  Link Type: Stub Network, Link ID: 192.0.17.1, Link Data: 255.255.255.0, Link Cost: 0,
  Link Type: Point-To-Point, Link ID: 192.0.15.1, Link Data: 192.0.7.2, Link Cost: 5,
  Link Type: Stub Network, Link ID: 192.0.7.0, Link Data: 255.255.255.0, Link Cost: 5,
  Link Type: Point-To-Point, Link ID: 192.0.18.1, Link Data: 192.0.9.1, Link Cost: 10,
  Link Type: Stub Network, Link ID: 192.0.9.0, Link Data: 255.255.255.0, Link Cost: 10,
  Link Type: Point-To-Point, Link ID: 192.0.19.1, Link Data: 192.0.10.1, Link Cost: 10,
  Link Type: Stub Network, Link ID: 192.0.10.0, Link Data: 255.255.255.0, Link Cost: 10,
```

ANSWER:

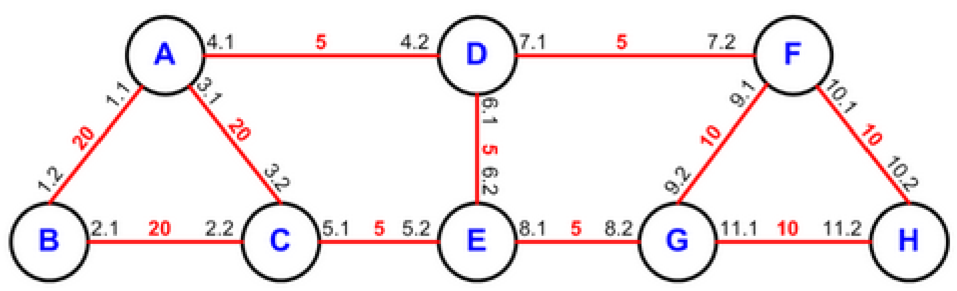
The Router can build a map for the topology of the network through first creating an association with the neighbouring routers. The relationship with that neighbouring router is then upgraded to be regarded as 'adjacent' so that link state information can be shared. As each router relays information about its directly connected routers to all of the routers in an area, when the Shortest Path algorithm is run each router is able to build up an understanding of the topology of the local network by knowing about these directly connected routers which are stored in the Link State database.

For example, in the snapshot of the Link State database it can be seen that Router A can create a map of the topology of the network as it can read from the LSA that from Router F (17.1), there is a 'Point-To-Point' link of cost 5 to Router D (15.1) and 'Point-To-Point' links of cost 10 to Routers G (18.1) and H (19.1). This small portion of the LSA gives each router a significant information on the structure of the network, and it can be seen how that with link information about each router, the entire topology for the network can be known.

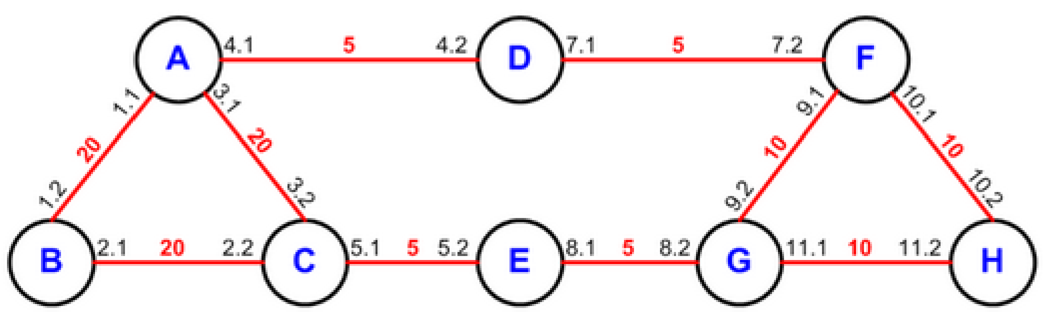
4. Explain How The Failure Has Affected The Content Of The Routing Table And Link State Database

Imagine you have created a duplicate scenario of the No_Areas scenario. The new scenario is called Q4_No_Areas_Failure. In this new scenario you have simulated a failure of the link connecting Router D and Router E. The failure starts after 100 seconds. Below is a snapshot of the link state database and the routing table for Router A. Explain how the failure has affected the content of the routing table and link state database.

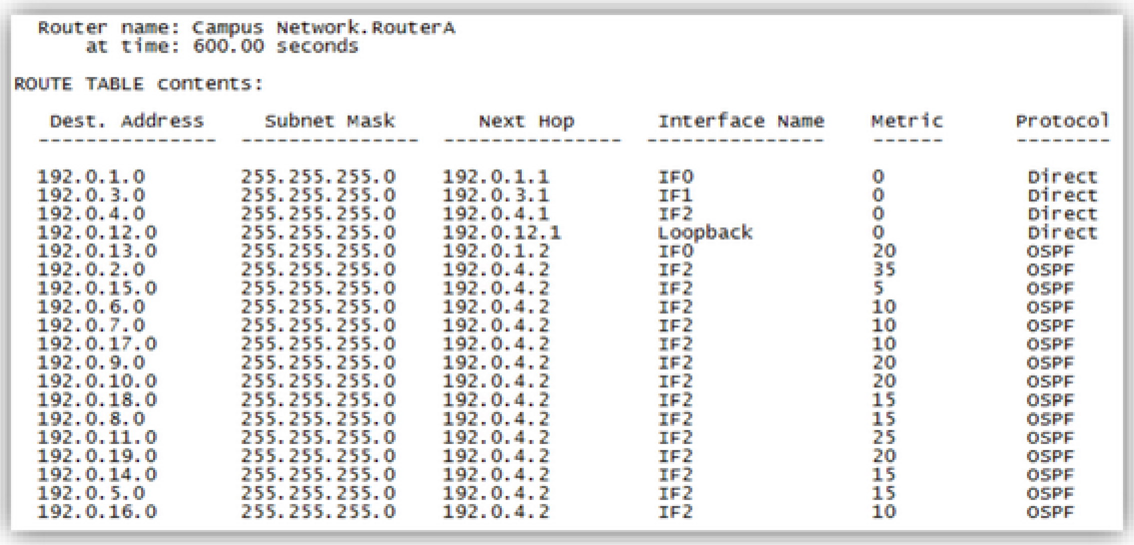
Before Failure



After Failure

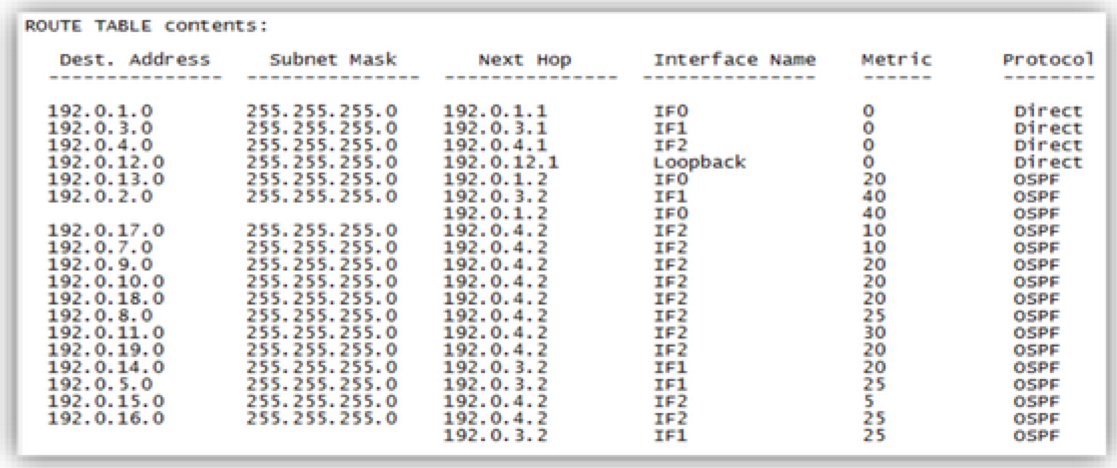


RouterA routing table for No_Areas scenario before the link failure



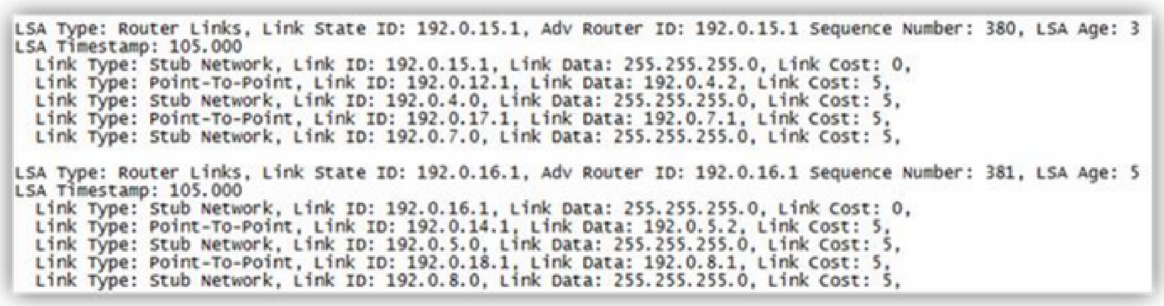
```
  Router name: Campus Network.RouterA
      at time: 600.00 seconds

ROUTE TABLE contents:

   Dest. Address      Subnet Mask       Next Hop        Interface Name     Metric       Protocol
  ---------------   ---------------  ---------------   ---------------    ------       --------

  192.0.1.0         255.255.255.0    192.0.1.1         IF0                0             Direct
  192.0.3.0         255.255.255.0    192.0.3.1         IF1                0             Direct
  192.0.4.0         255.255.255.0    192.0.4.1         IF2                0             Direct
  192.0.12.0        255.255.255.0    192.0.12.1        Loopback           0             Direct
  192.0.13.0        255.255.255.0    192.0.1.2         IF0                20            OSPF
  192.0.2.0         255.255.255.0    192.0.4.2         IF2                35            OSPF
  192.0.15.0        255.255.255.0    192.0.4.2         IF2                5             OSPF
  192.0.6.0         255.255.255.0    192.0.4.2         IF2                10            OSPF
  192.0.7.0         255.255.255.0    192.0.4.2         IF2                10            OSPF
  192.0.17.0        255.255.255.0    192.0.4.2         IF2                10            OSPF
  192.0.9.0         255.255.255.0    192.0.4.2         IF2                20            OSPF
  192.0.10.0        255.255.255.0    192.0.4.2         IF2                20            OSPF
  192.0.18.0        255.255.255.0    192.0.4.2         IF2                15            OSPF
  192.0.8.0         255.255.255.0    192.0.4.2         IF2                15            OSPF
  192.0.11.0        255.255.255.0    192.0.4.2         IF2                25            OSPF
  192.0.19.0        255.255.255.0    192.0.4.2         IF2                20            OSPF
  192.0.14.0        255.255.255.0    192.0.4.2         IF2                15            OSPF
  192.0.5.0         255.255.255.0    192.0.4.2         IF2                15            OSPF
  192.0.16.0        255.255.255.0    192.0.4.2         IF2                10            OSPF
```

RouterA routing table for No_Areas scenario after the link failure



```
ROUTE TABLE contents:

   Dest. Address       Subnet Mask         Next Hop          Interface Name     Metric      Protocol
   ---------------     ---------------     ---------------   ---------------    ------      --------

   192.0.1.0           255.255.255.0       192.0.1.1         IF0                0            Direct
   192.0.3.0           255.255.255.0       192.0.3.1         IF1                0            Direct
   192.0.4.0           255.255.255.0       192.0.4.1         IF2                0            Direct
   192.0.12.0          255.255.255.0       192.0.12.1        Loopback           0            Direct
   192.0.13.0          255.255.255.0       192.0.1.2         IF0                20           OSPF
   192.0.2.0           255.255.255.0       192.0.3.2         IF1                40           OSPF
                                           192.0.1.2         IF0                40           OSPF
   192.0.17.0          255.255.255.0       192.0.4.2         IF2                10           OSPF
   192.0.7.0           255.255.255.0       192.0.4.2         IF2                10           OSPF
   192.0.9.0           255.255.255.0       192.0.4.2         IF2                20           OSPF
   192.0.10.0          255.255.255.0       192.0.4.2         IF2                20           OSPF
   192.0.18.0          255.255.255.0       192.0.4.2         IF2                20           OSPF
   192.0.8.0           255.255.255.0       192.0.4.2         IF2                25           OSPF
   192.0.11.0          255.255.255.0       192.0.4.2         IF2                30           OSPF
   192.0.19.0          255.255.255.0       192.0.4.2         IF2                20           OSPF
   192.0.14.0          255.255.255.0       192.0.3.2         IF1                20           OSPF
   192.0.5.0           255.255.255.0       192.0.3.2         IF1                25           OSPF
   192.0.15.0          255.255.255.0       192.0.4.2         IF2                5            OSPF
   192.0.16.0          255.255.255.0       192.0.4.2         IF2                25           OSPF
                                           192.0.3.2         IF1                25           OSPF
```

Snapshot of Link State Database



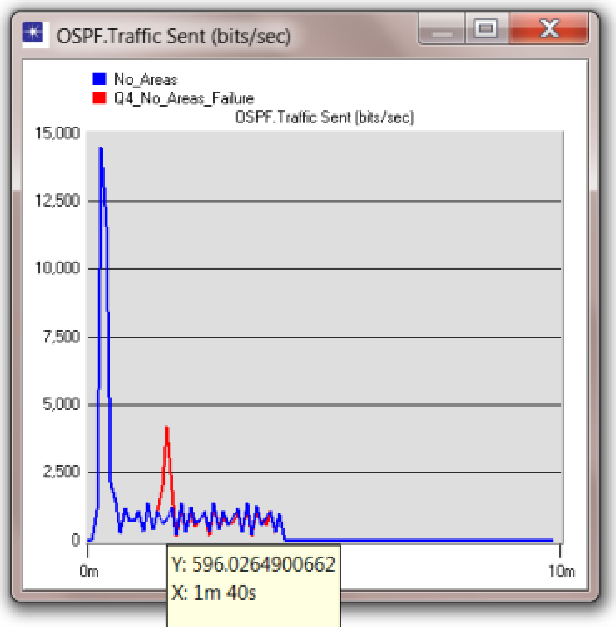
```
LSA Type: Router Links, Link State ID: 192.0.15.1, Adv Router ID: 192.0.15.1 Sequence Number: 380, LSA Age: 3
LSA Timestamp: 105.000
  Link Type: Stub Network, Link ID: 192.0.15.1, Link Data: 255.255.255.0, Link Cost: 0,
  Link Type: Point-To-Point, Link ID: 192.0.12.1, Link Data: 192.0.4.2, Link Cost: 5,
  Link Type: Stub Network, Link ID: 192.0.4.0, Link Data: 255.255.255.0, Link Cost: 5,
  Link Type: Point-To-Point, Link ID: 192.0.17.1, Link Data: 192.0.7.1, Link Cost: 5,
  Link Type: Stub Network, Link ID: 192.0.7.0, Link Data: 255.255.255.0, Link Cost: 5,

LSA Type: Router Links, Link State ID: 192.0.16.1, Adv Router ID: 192.0.16.1 Sequence Number: 381, LSA Age: 5
LSA Timestamp: 105.000
  Link Type: Stub Network, Link ID: 192.0.16.1, Link Data: 255.255.255.0, Link Cost: 0,
  Link Type: Point-To-Point, Link ID: 192.0.14.1, Link Data: 192.0.5.2, Link Cost: 5,
  Link Type: Stub Network, Link ID: 192.0.5.0, Link Data: 255.255.255.0, Link Cost: 5,
  Link Type: Point-To-Point, Link ID: 192.0.18.1, Link Data: 192.0.8.1, Link Cost: 5,
  Link Type: Stub Network, Link ID: 192.0.8.0, Link Data: 255.255.255.0, Link Cost: 5,
```

ANSWER:

The failure effects the content of the Routing Table and the Link State Database by increasing the metric for some of the routes. For example the route from Router A (12.1) to Router G (18.1) increases from a metric of 15 to 20. This is because the route A>D>E>G cannot be taken and instead the next smallest route A>D>F>G is used. This is also true for routes from A to any other router where the link between Routers D and E were used in the route.

In the snapshot of the LSD we can see that there is no link of metric 5 between Router D (15.1) and and Router E (16.1) as there would have been before the link failure.

5. Explain The Graph
Shown below is a graph comparing the traffic sent (bits/sec) for both the No_Areas scenario and the Q4_No_Areas_Failure scenario. Explain the graph.



ANSWER:

The No_Areas_Failure scenario features a spike in Traffic sent at 100 seconds as that is when the link between Router D and Router E is failed. This is due to the updates that need to be sent to update routers that the link between Router D and Router E has failed.

Conclusion

From this simulation of using OSPF in multiple scenarios we can see that, while not as simple as RIP, OSPF functions effectively and is extensible as a protocol for maintaining routes within a network while using minimal network traffic. Through the use of Areas, we saw how traffic can be reduced by restricting traffic between adjacent routers by following direct links and through the use of balancing we attempted to see how traffic could be distributed across multiple routes. We also investigated what happens to routing tables and the LSA database when a link fails within a network, and how a router can build up an understanding of a networks topology from the LSA database.