Modification added to the Code

The original code is an example code by the name of lte-sl-in-cnvrg-comm-mode1.cc
Present in directory: src/lte/examples/d2d-examples/ in psc-ns3.

The following are the changes added to the code.

```
// class for d2d device
class d2d_user {
	public :

	NodeContainer d2dNodes;
	int Node_id ;
	int application_type;
	int cluster_id;
	Vector position;
	int count = -1;
	Ipv4Address address1, address2;

	void store(Vector ue1, Vector ue2, int id) {
		Node_id = id;
		application_type = rand()%2+1;
		position.x = (ue1.x + ue2.x)/2;
		position.y = (ue1.y + ue2.y)/2;
		position.z = (ue1.z + ue2.z)/2;
	}
};

// class for cu device
class cu_users {
	public :

	NodeContainer Node;
	int application_type;
	int cluster_id;
	Vector position;
	Ipv4Address address;
	int occupied = 0;
```

```
        void store(Vector cu) {
                position.x = cu.x;
                position.y = cu.y;
                position.z = cu.z;
        }
};

 // class for center
 struct point {
          double x, y;
          int cluster = -1;
          int application;
};
```

/*The above snippet of code refers to two class objects and point structure.
The d2d_user class stores various information such as application type, node_id,coordinates etc. The object stores the node container for the two pair of the nodes.

The cu_user is similar to d2d_user interms of format except the node container is for a cu-device

The point struct is for storing the positional coordinates cluster id and application type.*/

```
  // k means algorithm
  std::vector<point> k_means(std::vector<d2d_user> init, int k, d2d_user* ptr)
   {
    auto tmp = std::max_element(init.begin(), init.end(), [](d2d_user a, d2d_user b)
{return a.position.x < b.position.x; });
    int max_x = tmp->position.x;
    tmp = std::max_element(init.begin(), init.end(), [](d2d_user a, d2d_user b) {return
a.position.y < b.position.y; });
    int max_y = tmp->position.y;
    tmp = std::min_element(init.begin(), init.end(), [](d2d_user a, d2d_user b) {return
a.position.x < b.position.x; });
    int min_x = tmp->position.x;
    tmp = std::min_element(init.begin(), init.end(), [](d2d_user a, d2d_user b) {return
a.position.y < b.position.y; });
    int min_y = tmp->position.y;
```

```
std::vector<point> centers(k);
for (int i = 0; i < k; i++)
{
   centers[i].x = (rand() + i*23) % (max_x - min_x + 1) + min_x;
     centers[i].y = (rand() - i*17) % (max_y - min_y + 1) + min_y;
     centers[i].cluster = i;
}

for (int i = 0; i < 1000; i++)
 {
     for (long unsigned int j = 0; j < init.size(); j++)
     {
             double* dists = new double[k];
             for (int p = 0; p < k; p++)
             {
                     double a = std::abs(init[j].position.y - centers[p].y);
                     double b = std::abs(init[j].position.x - centers[p].x);
                     dists[p] = std::sqrt(std::pow(a, 2) + std::pow(b, 2));
             }
             init[j].cluster_id = std::min_element(dists, dists + k) - dists;
             ptr[init[j].Node_id].cluster_id = init[j].cluster_id;
             delete[] dists;
     }

     std::unique_ptr<double[]> sum_x(new double[k], std::default_delete<double[]>());
     std::unique_ptr<double[]> sum_y(new double[k], std::default_delete<double[]>());
     int count[k];

     for (int p = 0; p < k; p++)
     {
             sum_x[p] = 0;
             sum_y[p] = 0;
             count[p] = 0;
     }
     for (long unsigned int f = 0; f < init.size(); f++)
     {
             sum_x[init[f].cluster_id] += init[f].position.x;
             sum_y[init[f].cluster_id] += init[f].position.y;
             count[init[f].cluster_id]++;
             ptr[init[f].Node_id].count = count[init[f].cluster_id];
```

```
        }

        for (int f = 0; f < k; f++)
        {
                centers[f].x = sum_x[f] / count[f];
                centers[f].y = sum_y[f] / count[f];
        }
    }

   return centers;
  }


  //print the shards
  void print(d2d_user d2d[], cu_users cu[])
  {
    std::cout<<"\n";
    // for(int i=0; i<60 ; i++) std::cout<<"*";
    std::cout<<"\n";
    std::cout<< "Commercial"<<std::endl;
    std::cout<<"\n";
    std::cout<< "Cluster 1:"<<std::endl;
     for(int i=0;i<5;i++)
     {
     if(d2d[i].application_type == 1)
     {
     if(d2d[i].cluster_id == 0 )
     {
     std::cout<< "                "<<"UE"<<2*i<<" "<<" , IP_Address :
"<<d2d[i].address1<<std::endl;
     std::cout<< "                "<<"UE"<<2*i+1<<" "<<" , IP_Address :
"<<d2d[i].address2<<std::endl;
     }
     }
     }
     for(int i=0;i<3;i++)
     {
     if(cu[i].application_type == 1)
     {
     if(cu[i].cluster_id == 0 )
```

```
    {
    std::cout<< "                "<<"CU"<<i+1<<"  , IP_Address :
"<<cu[i].address<<std::endl;
    }
    }
    }
    std::cout<< ""<<std::endl<<std::endl;
    // for(int i=0; i<60 ; i++) std::cout<<"*";
    std::cout<<"\n";

    std::cout<< "Public Safety Communication"<<std::endl;
    std::cout<<"\n";
    std::cout<<"Cluster 1:" <<std::endl;
    for(int i=0;i<5;i++)
    {
    if(d2d[i].application_type == 2)
    {
    if(d2d[i].cluster_id == 0 )
    {
    std::cout<< "                "<<"UE"<<2*i<<" "<<" , IP_Address :
"<<d2d[i].address1<<std::endl;
    std::cout<< "                "<<"UE"<<2*i+1<<" "<<" , IP_Address :
"<<d2d[i].address2<<std::endl;
    }
    }
    }
    for(int i=0;i<3;i++)
    {
    if(cu[i].application_type == 2)
    {
    if(cu[i].cluster_id == 0)
    {
    std::cout<< "                "<<"CU"<<i+1<<"  , IP_Address :
"<<cu[i].address<<std::endl;
    }
    }
    }
    std::cout<< ""<<std::endl<<std::endl;

    std::cout<<"\n";
```

```
    std::cout<<"Cluster 2:" <<std::endl;
    for(int i=0;i<5;i++)
    {
    if(d2d[i].application_type == 2)
    {
    if(d2d[i].cluster_id == 1 )
    {
    std::cout<< "                "<<"UE"<<2*i<<" "<<" , IP_Address :
"<<d2d[i].address1<<std::endl;
    std::cout<< "                "<<"UE"<<2*i+1<<" "<<" , IP_Address :
"<<d2d[i].address2<<std::endl;
    }
    }
    }
    for(int i=0;i<3;i++)
    {
    if(cu[i].application_type == 2)
    {
    if(cu[i].cluster_id == 1 )
    {
    std::cout<< "                "<<"CU"<<i+1<<"  , IP_Address :
"<<cu[i].address<<std::endl;
    }
    }
    }
    std::cout<< ""<<std::endl<<std::endl;
    // for(int i=0; i<60 ; i++) std::cout<<"*";
    std::cout<<"\n";
  }
```

This function prints the information related to sharding . It prints the cluster id and ip addresses for the nodes belonging to the respective cluster

```
// function to print the SINR values
void PhySnirTrace (std::string context ,uint16_t cellId, uint16_t rnti, double rsrp, double
sinr, uint8_t componentCarrierId)
{


   if(Simulator::Now().GetSeconds() >= 2.5 && Simulator::Now().GetSeconds() <=
2.50100 )
   {

   int nth=3;  //looking for the second ocurrence of "/"
   int cnt=0;
   size_t pos=0;

   while( cnt != nth )
   {
      pos = context.find("/", pos);
      if ( pos == std::string::npos )
        std::cout << nth << "th ocurrence not found!"<< std::endl;
      pos+=1;
      cnt++;
   }

   std::string str1;
   str1 = context.substr(10,pos-11);
   int ue= atoi(str1.c_str())-7;



   for(int i=0; i< size2; i++)
   {
     if(2*shard2[i] == ue && i == 0)
     {
        std::cout<<"UE"<<ue<<" belonging to group "<<i+1<<"\n";
        std::cout<<"RNTI : "<<rnti<<"\nRSRP : "<<rsrp<<"\nSINR : "<<sinr<<std::endl;
        std::cout<<"Time : " <<Simulator::Now().GetSeconds()<<"\n";
        std::cout<<"\n";
     }
   }
    //std::cout<<" Time : " <<Simulator::Now().GetSeconds()<<"\n";
```

```
   }

   if(Simulator::Now().GetSeconds() >= 3.5 && Simulator::Now().GetSeconds() <=
3.50100 )
   {

   int nth=3;  //looking for the second ocurrence of "/"
   int cnt=0;
   size_t pos=0;

   while( cnt != nth )
   {
      pos = context.find("/", pos);
      if ( pos == std::string::npos )
        std::cout << nth << "th ocurrence not found!"<< std::endl;
      pos+=1;
      cnt++;
   }

   std::string str1;
   str1 = context.substr(10,pos-11);
   int ue= atoi(str1.c_str())-7;


   for(int i=0; i< size2; i++)
   {
     if(2*shard2[i] == ue && (i == 0 || i == 1) )
     {
        //std::cout<<context<<std::endl;
        std::cout<<"UE"<<ue<<" belonging to group "<<i+1<<"\n";
        std::cout<<"RNTI : "<<rnti<<"\nRSRP : "<<rsrp<<"\nSINR : "<<sinr<<std::endl;
        std::cout<<"Time : " <<Simulator::Now().GetSeconds()<<"\n";
        std::cout<<"\n";
     }
   }
   }

   if(Simulator::Now().GetSeconds() >= 4.5 && Simulator::Now().GetSeconds() <=
4.50100)
   {
```

```
   int nth=3;  //looking for the second ocurrence of "/"
   int cnt=0;
   size_t pos=0;

   while( cnt != nth )
   {
      pos = context.find("/", pos);
      if ( pos == std::string::npos )
        std::cout << nth << "th ocurrence not found!"<< std::endl;
      pos+=1;
      cnt++;
   }

   std::string str1;
   str1 = context.substr(10,pos-11);
   int ue= atoi(str1.c_str())-7;


   for(int i=0; i< size2; i++)
   {
     if(2*shard2[i] == ue && (i == 0 || i == 1 || i == 2))
     {
        //std::cout<<context<<std::endl;
        std::cout<<"UE"<<ue<<" belonging to group "<<i+1<<"\n";
        std::cout<<"RNTI : "<<rnti<<"\nRSRP : "<<rsrp<<"\nSINR : "<<sinr<<std::endl;
        std::cout<<"Time : " <<Simulator::Now().GetSeconds()<<"\n";
        std::cout<<"\n";
     }
   }

   }

   if(Simulator::Now().GetSeconds() >= 5.5 && Simulator::Now().GetSeconds() <=
5.50100)
   {

   int nth=3;  //looking for the second ocurrence of "/"
   int cnt=0;
   size_t pos=0;
```

```
    while( cnt != nth )
    {
        pos = context.find("/", pos);
        if ( pos == std::string::npos )
          std::cout << nth << "th ocurrence not found!"<< std::endl;
        pos+=1;
        cnt++;
    }

    std::string str1;
    str1 = context.substr(10,pos-11);
    int ue= atoi(str1.c_str())-7;


    for(int i=0; i< size2; i++)
    {
      if(2*shard2[i] == ue && (i == 0 || i == 1 || i == 2))
      {
        //std::cout<<context<<std::endl;
        std::cout<<" UE"<<ue<<" belonging to group "<<i+1<<"\n";
        std::cout<<" RNTI : "<<rnti<<"\nRSRP : "<<rsrp<<"\nSINR : "<<sinr<<std::endl;
        std::cout<<" Time : " <<Simulator::Now().GetSeconds()<<"\n";
        std::cout<<"\n";
      }
    }

    }
}
```

This function prints the following:
Temporary id for the node provided by the eNB
The RSRP value which measures the quality of signal
We also print the SINR value