```
#include <set>
#include <iterator>
#include <iostream>

//Algorithm 1 Multi-Channel ONAMA Scheduling at BS I

using namespace std;

int main()
{

	class Node {
		public:
			int Lat;
			int Lon;
			//state;
		};

	//link i
	class i {
		public:
			Node a;
			Node b;
			//state;
			char name[]= " ";
			//int mean;
		};
	//traffic demand
	class d {
		public:
			Node a;
			Node b;
			state;
			Prio;
		}

	Node * one = new Node(3,2);
	Node * two = new Node(3,19);
	Node * three = new Node(8,7);
	Node * four = new Node(8,14);
	Node * five = new Node(13,7);
	Node * six = new Node(12,14);
	Node * seven= new Node(18,2);
	Node * eight = new Node(18,19);

	//empty set container
	set<i> Mi;

	//downlink
	Mi.insert(three,four,down1);

	//uplink
	Mi.insert(two,six,"uplink1");
	Mi.insert(six,three,"uplink2");

	//UE-to-UE
```

```
        Mi.insert(five,seven,"Ue1");
        Mi.insert(one,eight,"Ue2");

        //BSes
        //node 1 and node 8
}

//1: state.i.rb = UNDECIDED, ∀rb ∈RB;

for (k = Mi.begin(); k!= Mi.end(); k++) {

        k.state = UNDECIDED;
        //Step 1: Preallocation

//2: k.rb = min{dk,mean},∀k ∈Mi ∪i;
        k.rb = min{d[k], k.mean};

        for ( k.rb = min{d[k],k.mean}, k < Mi.end(), k++ ){

                //3: Prio.k.rb = MAXIMUM,∀rb ∈k;
                k.Prio = MAXIMUM;
                //4: state.k.rb = ACTIV E,∀rb ∈k;
                k.state = ACTIVE;
                //5: if dk <= mean then
                if ( d[k] <= k.mean ){
                        //6: state.k.rb = INACTIVE,∀rb /∈k;
                        k.state = INACTIVE;
                }//7: end if
        }
}
        //Step 2: Priority Calculation

//8: Prio.k.rb.d = Hash(k⊕ d⊕ t⊕ rb)⊕ k⊕ d,

//9: ∀k ∈Mi ∪i,∀rb ∈RB,∀d ∈[1,dk −mean];
//10: Prio.k.rb = Maxdk−mean d=1 Prio.k.rb.d,
        for (d=1; d<= (d[k]-k.mean); d++){
                k.Prio = k.rb * d[k];
        }
//11: ∀k ∈Mi ∪i, ∀rb ∈RB;
//Step 3: State Selection (i.e., Scheduling)
//12: done = false;
         done= false;
//13: while done == false do
while (done == false ) do{
        //14: done = true;
        done = true;

//15: for each rb ∈RB in increasing order of rb ID do
        for ( rb = 0; rb < RB; rb++ ){
                //16: if di −mean > 0 && state.i.rb == UNDECIDED
                //17: && Prio.i.rb > Prio.k.rb for each ACTIVE
                for ( i=Mi.begin(); i<=Mi.end(); i++){
                if( d[i] - mean > 0 && state.i.rb == UNDECIDED && Prio.i.rb > Prio.k.rb){
                        //18: UNDECIDED k ∈Mi then
                        //19: state.i.rb = ACTIVE;
                        i.state = ACTIVE;
```

```
                //20: di = di −mean −1
                d[i] = d[i] − i.mean −1;
                //21: if di == 0 then
                if ( d[i] == 0 ){
                        22: state.i.rb2 = INACTIVE, for each rb2 ∈
                        23: RB where state.i.rb2 == UNDECIDED;
                        i.state = INACTIVE;
                } //24: end if
        } //25: end if

        //26: if Prio.i.rb < Prio.k.rb for any
        if ( i.Prio.rb < k.Prio.rb ){
                //27: ACTIVE k ∈Mi then
                //28: state.i.rb = INACTIVE;
                i.state = INACTIVE;
        } //29: end if

        //30: if state.i.rb == UNDECIDED then
        if (i.state == UNDECIDED ){
                31: done = false;
                done = false;
        } //32: end if
    } //33: end for
    //34: Share state.i.rb, ∀rb ∈RB;
    //35: Update state.k.rb, ∀k ∈Mi,∀rb ∈RB based on
    //36: information from other BSes;
} //37: end while
```