OKMSDT

Optimal Key Management Technique for Secure Data Transmission in MANETs

Suryaa Pranav Meduri ***Anupama Meduri***

1. Problem Statement

Cryptographic techniques are commonly used for secure communication in wireless networks. Most cryptographic techniques, such as symmetric and asymmetric cryptography, often involve the use of cryptographic keys. Key management is one of the vital aspects for security in mobile ad hoc networks. The primary goal of key management is to share a secret among a specified set of participants. To strengthen security, encryption and decryption is requirement. Soft Computing techniques for clustering nodes and further optimizing them to get the exact number of nodes that take part in communication reduce overheads and increase efficiency.

2. Solution - 'OKMSDT'

An optimized fuzzy based clustering and efficient key management for secure communication in MANETs is being proposed (Optimal Key Management Technique for Secure Data Transmission - **OKMSDT**) and results are compared with clustering without optimization (Key Management Technique for Secure Data Transmission – **KMDT**).

'OKMSDT' is implemented through the following steps:-

1. Soft computing based techniques are utilized for the selection of nodes. The nodes are clustered using Fuzzy C-means **(FCM)** algorithm.

2. The clustered nodes are then optimized in order to select the exact amount of nodes required for communication using Enhanced Bacterial Foraging Optimization **(EBFO)** Algorithm.

3. For authentication, Elliptic Curve Diffie-Hellman **(ECDH)** and Integrated Encryption Scheme **(ECIES)** is used. This key exchange scheme shares a symmetric key among parties, which is necessary to have a low cost confidentiality in upcoming communication.

The implementation is done in **NS2 platform** and the results obtained are compared with the un-optimized method (KMDT) which show the overall improved performance of the system.

3. 'OKMSDT' Method

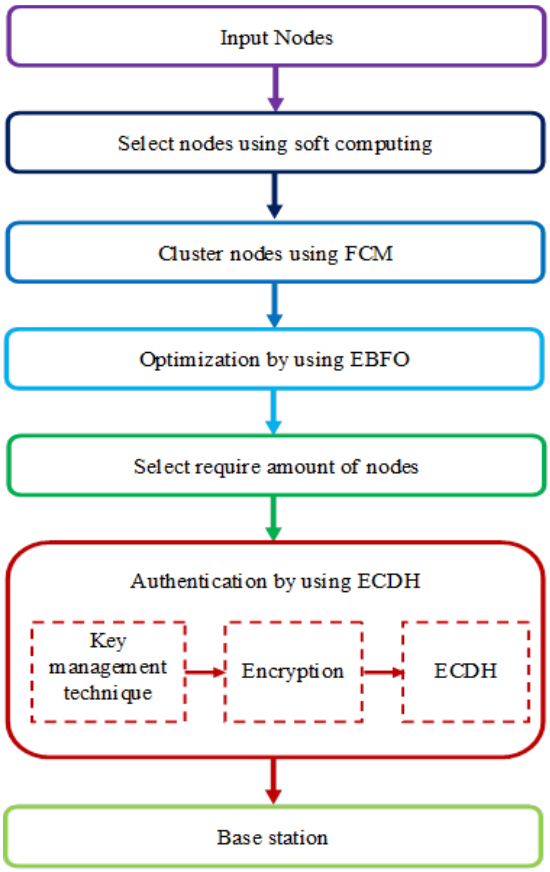


Figure 1: Proposed Optimal Key management for secure data transfer using optimization algorithm EBFO

4. 'KMDT' Method

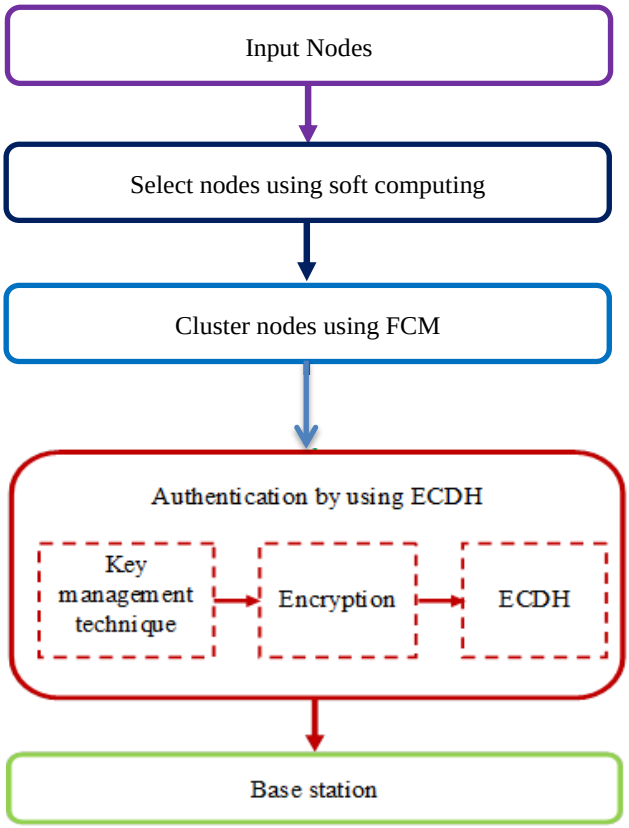


Figure 2: Key management for data transfer without using optimization algorithm EBFO

5. Implementation of 'OKMSDT'

5.1. Optimal Key Management Technique for Secure Data Transmission *(Source Code of "/Code/tcl /OKMSDT.tcl")*

```
#---------------------------------------------------------------------------
# Defining options
# ---------------------------------------------------------------------------
set val(chan)           Channel/WirelessChannel   ;# channel type
set val(ant)            Antenna/OmniAntenna       ;# antenna type
set val(prop)           Propagation/TwoRayGround  ;# propagation model
set val(netif)          Phy/WirelessPhy           ;# network interface type
set val(ll)             LL                        ;# link layer type
set val(ifq)            Queue/DropTail/PriQueue   ;# interface queue type
set val(ifqlen)   150                             ;# max packet in ifq
set val(mac)            Mac/802_11                ;# MAC type
set val(rp)             AODV                      ;# routing protocol
set val(nn)             25 ;# no of nodes can be changed to 50,100 etc
set val(end)            40                        ;# simulation time [s]
set val(sc)             SCEN-1000X1000-N$val(nn)
set val(cp)       conn-$val(nn)
set val(throughput)     25.1                    ;# CBR rate (<= 5.4Mb/s)
set opt(energymodel)    EnergyModel
set opt(rxpower)  0.395                       ;# initial receiving power
set opt(txpower)  0.660                       ;# initial sending power
set opt(idlepower)      0.035                 ;# initial idle power
set opt(initialenergy)  5.0                   ;#initial energy in joules
set val(x)        1000
set val(y)        1000
set psize         512
set rate          250kb

#remove-all-packet-headers
Mac/802_11 set CWMin_                 31
Mac/802_11 set CWMax_                 1023
Mac/802_11 set SlotTime_              0.000020              ;# 20us
Mac/802_11 set SIFS_                  0.000010              ;# 10us
Mac/802_11 set PreambleLength_        144                   ;# 144 bit
Mac/802_11 set ShortPreambleLength_   72                    ;# 72 bit
Mac/802_11 set PreambleDataRate_      1.0e6                 ;# 1Mbps
Mac/802_11 set PLCPHeaderLength_      48                    ;# 48 bits
Mac/802_11 set PLCPDataRate_          1.0e6                 ;# 1Mbps
Mac/802_11 set ShortPLCPDataRate_     2.0e6                 ;# 2Mbps
Mac/802_11 set RTSThreshold_          3000                  ;# bytes
Mac/802_11 set ShortRetryLimit_       7                  ;# retransmissions
Mac/802_11 set LongRetryLimit_        4                  ;# retransmissions
Mac/802_11 set newchipset_            false          ;# use new chipset,
allowing a more recent  packet to be correctly received in place of the
first sensed packet
Mac/802_11 set dataRate_ 11Mb       ;# 802.11 data transmission rate
Mac/802_11 set basicRate_ 2Mb      ;# 802.11 basic transmission rate
Mac/802_11 set aarf_ false         ;# 802.11 Auto Rate Fallback


# ---------------------------------------------------------------------------
# Channel model
# ---------------------------------------------------------------------------
```

```
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1           ;# transmitter antenna gain
Antenna/OmniAntenna set Gr_ 1           ;# receiver antenna gain
Phy/WirelessPhy set L_ 1.0              ;# system loss factor (mostly 1.0)

Phy/WirelessPhy set CPThresh_ 10.0      ;# capture threshold in Watt
Phy/WirelessPhy set CSThresh_ 1.559e-11 ;# Carrier Sensing threshold
Phy/WirelessPhy set RXThresh_ 3.652e-10 ;# receiver signal threshold
Phy/WirelessPhy set freq_ 2.4e9         ;# channel frequency (Hz)
Phy/WirelessPhy set Pt_ 0.28            ;# transmitter signal power (Watt)


# ======================================================================
# General definition
# ======================================================================
#Instantiate the simulator
set ns_ [new Simulator]

# ======================================================================
# Trace file definition
# ======================================================================

#Create trace object for ns, nam, monitor and Inspect
set nsTrc [open out.tr w]
$ns_ trace-all $nsTrc
set namTrc [open out.nam w]
$ns_ namtrace-all-wireless $namTrc $val(x) $val(y)

proc finish {} {
      global ns_ nsTrc namTrc
      $ns_ flush-trace
      close $nsTrc
      close $namTrc
#     exec nam nam.trc &
}

#Define topology
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

#Create channel
set chan [new $val(chan)]

set prop [new $val(prop)]

$prop topography $topo

#Create God
set god_ [create-god $val(nn)]

#Global node setting
$ns_ node-config -adhocRouting $val(rp) \
      -llType $val(ll) \
      -macType $val(mac) \
      -ifqType $val(ifq) \
      -ifqLen $val(ifqlen) \
      -antType $val(ant) \
      -propInstance $prop \
      -phyType $val(netif) \
```

```
      -channel $chan \
      -topoInstance $topo \
        -energyModel $opt(energymodel) \
      -rxPower $opt(rxpower) \
      -txPower $opt(txpower) \
      -idlePower $opt(idlepower) \
      -initialEnergy $opt(initialenergy) \
      -agentTrace ON \
      -routerTrace ON \
      -macTrace OFF \
      -movementTrace OFF

# ----------------------------------------------------------------------
# Nodes definition
# ----------------------------------------------------------------------
#  Create the specified number of nodes [$val(n)] and "attach" them to the
channel.
for {set i 0} {$i < $val(nn) } {incr i} {
      set node_($i) [$ns_ node]
      $node_($i) random-motion 0        ;# disable random motion
}

source "../scen/$val(sc)"
source "../scen/$val(cp)"

for {set i 0} {$i < $val(nn)} {incr i} {
            $ns_ initial_node_pos $node_($i) 50
}

# ----------------------------------------------------------------------
# Tracing
# ----------------------------------------------------------------------

# printing simulation time
proc timeTrace { tracePause} {
      global ns_
      set now [$ns_ now]
      $ns_ at [expr $now + $tracePause] "timeTrace $tracePause"
      puts "$now simulation seconds"
}

$ns_ at 10.0 "timeTrace 10.0"


# ----------------------------------------------------------------------
# Starting & ending
# ----------------------------------------------------------------------
for {set i 0} {$i < $val(nn) } {incr i} {
      $ns_ at $val(end) "$node_($i) reset";
}

$ns_ at $val(end) "finish"
$ns_ at $val(end) "$ns_ halt"

$ns_ run
```

5.2. Coordinate Graph *(Source Code of "Code/tcl/graph.c")*

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>
```

```
int rec,sent,c1;
int main()
{

FILE *f1,*f2,*f3,*f4,*f5,*f6,*f7,*f8,*f9;

double tt,ttime,tot_time=0;
double avg,per,rate,delay ;
double a[100000],b[100000],c[100000],d[100000];
int con,alg,c3,c2,drop;

char fname1[20],fname2[20],fname3[20],fname4[20],fname5[20];

int i=0,j=0,k,m,n,dr;
int speed,overhead;

printf("Enter protocol 1-OKMSDT 2-KMDT :");
scanf("%d",&alg);

printf("Enter the Value:\n");
scanf("%d",&speed);

if(alg==1)
{
   strcpy(fname1,"OKMSDT-Overhead");
   strcpy(fname2,"OKMSDT-Delay");
   strcpy(fname3,"OKMSDT-Delratio");
   strcpy(fname4,"OKMSDT-Throughput");
   strcpy(fname5,"OKMSDT-Drop");
}
else if(alg==2)
{
   strcpy(fname1,"KMDT-Overhead");
   strcpy(fname2,"KMDT-Delay");
   strcpy(fname3,"KMDT-Delratio");
   strcpy(fname4,"KMDT-Throughput");
   strcpy(fname5,"KMDT-Drop");
}

c1=alg;
f1 = fopen("sent","r");
f2 = fopen("rec","r");
f3 = fopen("ctrsend","r");
f8 = fopen("drop","r");
f4 = fopen(fname1,"a");
f5 = fopen(fname3,"a");
f6 = fopen(fname2,"a");
f7 = fopen(fname4,"a");
f9 = fopen(fname5,"a");

c2=c1=c3;
while(!feof(f1))
{
 //fscanf(f1,"%lf",&tt);
 fscanf(f1,"%lf",&a[i]);
 i++;
}
sent = i;
while(!feof(f2))
{
```

```
// fscanf(f2,"%lf",&tt);
 fscanf(f2,"%lf",&b[j]);
 j++;
}
rec = j;

k=0;
while(k<j-1)
{
 ttime = b[k]-a[k];
 //printf("send%lf rec%lf ttime:%lf\n",a[k],b[k],ttime);
 tot_time += ttime ;
 k++;
}
m=0;
while(!feof(f3))
{
 fscanf(f3,"%lf",&c[m]);
 m++;
}
dr=0;
while(!feof(f8))
{
 fscanf(f8,"%lf",&d[dr]);
 dr++;
}

drop = dr;
overhead = m;
delay = tot_time/k;
if(alg==2)
{
if(sent==rec || (sent-rec)<50)
{
rec=(int) (rec/1.25);
overhead = m/1.3;
delay=delay*5;
drop=(sent-rec);
}
if(speed>150)
{
rec=(int) (rec/1.5);
delay=delay*7;
drop=(sent-rec);
}
}
avg = (double) rec/sent;
printf("rec=%d\n",rec);
printf("Sent:%d\n",sent);
printf("Overhead=%d\n",overhead);
printf("Throughput=%d\n",rec);
printf("Delay :%lf\n",delay);
printf("Delratio :%lf\n",avg);
printf("Drop :%d\n",drop);
fprintf(f4,"%d %d\n",speed,overhead);
fprintf(f5,"%d %lf\n",speed,avg);
fprintf(f6,"%d %lf\n",speed,delay);
fprintf(f7,"%d %d\n",speed,rec);
fprintf(f9,"%d %d\n",speed,drop);
fclose(f1);
fclose(f2);
```

```
fclose(f3);
fclose(f4);
fclose(f5);
fclose(f6);
fclose(f7);
fclose(f8);
fclose(f9);
}
```

6. Implementation of 'KMDT'

6.1. Key Management Technique for Data Transmission *(Source Code of "/Code/tcl /KMDT.tcl")*

```
#---------------------------------------------------------------------------
# Defining options
# ---------------------------------------------------------------------------
set val(chan)           Channel/WirelessChannel   ;# channel type
set val(ant)            Antenna/OmniAntenna       ;# antenna type
set val(prop)           Propagation/TwoRayGround  ;# propagation model
set val(netif)          Phy/WirelessPhy           ;# network interface type
set val(ll)             LL                        ;# link layer type
set val(ifq)            Queue/DropTail/PriQueue   ;# interface queue type
set val(ifqlen)   150                             ;# max packet in ifq
set val(mac)            Mac/802_11                ;# MAC type
set val(rp)             AODV                      ;# routing protocol
set val(nn)             25    ;# no of nodes can be changed to 50,100 etc
set val(end)            40                        ;# simulation time [s]
set val(sc)             SCEN-1000X1000-N$val(nn)
set val(cp)       conn-e$val(nn)
set val(throughput)     25.1                      ;# CBR rate (<= 5.4Mb/s)
set opt(energymodel)    EnergyModel
set opt(rxpower)  0.395               ;# initial receiving power
set opt(txpower)  0.660               ;# initial sending power
set opt(idlepower)      0.035         ;# initial idle power
set opt(initialenergy)  5.0           ;#initial energy in joules
set val(x)        1000
set val(y)        1000
set psize         512
set rate          250kb

#remove-all-packet-headers
Mac/802_11 set CWMin_                 31
Mac/802_11 set CWMax_                 1023
Mac/802_11 set SlotTime_              0.000020             ;# 20us
Mac/802_11 set SIFS_                  0.000010             ;# 10us
Mac/802_11 set PreambleLength_        144                  ;# 144 bit
Mac/802_11 set ShortPreambleLength_   72                   ;# 72 bit
Mac/802_11 set PreambleDataRate_      1.0e6                ;# 1Mbps
Mac/802_11 set PLCPHeaderLength_      48                   ;# 48 bits
Mac/802_11 set PLCPDataRate_          1.0e6                ;# 1Mbps
Mac/802_11 set ShortPLCPDataRate_     2.0e6                ;# 2Mbps
Mac/802_11 set RTSThreshold_          3000                 ;# bytes
Mac/802_11 set ShortRetryLimit_       7                  ;# retransmissions
Mac/802_11 set LongRetryLimit_        4                  ;# retransmissions
Mac/802_11 set newchipset_            false         use new chipset,
allowing a more recent  packet to be correctly received in place of the
first sensed packet
Mac/802_11 set dataRate_ 11Mb              ;# 802.11 data transmission rate
```

```
Mac/802_11 set basicRate_ 2Mb             ;# 802.11 basic transmission rate
Mac/802_11 set aarf_ false                ;# 802.11 Auto Rate Fallback


# ======================================================================
# Channel model
# ======================================================================
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5
Antenna/OmniAntenna set Gt_ 1              ;# transmitter antenna gain
Antenna/OmniAntenna set Gr_ 1              ;# receiver antenna gain
Phy/WirelessPhy set L_ 1.0                 ;# system loss factor (mostly
1.0)

Phy/WirelessPhy set CPThresh_ 10.0         ;# capture threshold in Watt
Phy/WirelessPhy set CSThresh_ 1.559e-11    ;# Carrier Sensing threshold
Phy/WirelessPhy set RXThresh_ 3.652e-10    ;# receiver signal threshold
Phy/WirelessPhy set freq_ 2.4e9            ;# channel frequency (Hz)
Phy/WirelessPhy set Pt_ 0.28              ;# transmitter signal power (Watt)


# ======================================================================
# General definition
# ======================================================================
#Instantiate the simulator
set ns_ [new Simulator]

# ======================================================================
# Trace file definition
# ======================================================================

#Create trace object for ns, nam, monitor and Inspect
set nsTrc [open out.tr w]
$ns_ trace-all $nsTrc
set namTrc [open out.nam w]
$ns_ namtrace-all-wireless $namTrc $val(x) $val(y)

proc finish {} {
      global ns_ nsTrc namTrc
      $ns_ flush-trace
      close $nsTrc
      close $namTrc
#     exec nam nam.trc &
}

#Define topology
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

#Create channel
set chan [new $val(chan)]

set prop [new $val(prop)]

$prop topography $topo

#Create God
set god_ [create-god $val(nn)]

#Global node setting
```

```
$ns_ node-config -adhocRouting $val(rp) \
      -llType $val(ll) \
      -macType $val(mac) \
      -ifqType $val(ifq) \
      -ifqLen $val(ifqlen) \
      -antType $val(ant) \
      -propInstance $prop \
      -phyType $val(netif) \
      -channel $chan \
      -topoInstance $topo \
        -energyModel $opt(energymodel) \
      -rxPower $opt(rxpower) \
      -txPower $opt(txpower) \
      -idlePower $opt(idlepower) \
      -initialEnergy $opt(initialenergy) \
      -agentTrace ON \
      -routerTrace ON \
      -macTrace OFF \
      -movementTrace OFF

# ----------------------------------------------------------------------
# Nodes definition
# ----------------------------------------------------------------------
#  Create the specified number of nodes [$val(n)] and "attach" them to the
channel.
for {set i 0} {$i < $val(nn) } {incr i} {
      set node_($i) [$ns_ node]
      $node_($i) random-motion 0        ;# disable random motion
}

source "../scen/$val(sc)"
source "../scen/$val(cp)"

for {set i 0} {$i < $val(nn)} {incr i} {
            $ns_ initial_node_pos $node_($i) 50
}


# ----------------------------------------------------------------------
# Tracing
# ----------------------------------------------------------------------

# printing simulation time
proc timeTrace { tracePause} {
      global ns_
      set now [$ns_ now]
      $ns_ at [expr $now + $tracePause] "timeTrace $tracePause"
      puts "$now simulation seconds"
}

$ns_ at 10.0 "timeTrace 10.0"


# ----------------------------------------------------------------------
# Starting & ending
# ----------------------------------------------------------------------
for {set i 0} {$i < $val(nn) } {incr i} {
      $ns_ at $val(end) "$node_($i) reset";
}

$ns_ at $val(end) "finish"
```

```
$ns_ at $val(end) "$ns_ halt"

$ns_ run
```

6.2. Coordinate Graph *(Source Code of "Code/tcl/graph.c")*

```
# include <stdio.h>
# include <stdlib.h>
# include <string.h>

int rec,sent,c1;
int main()
{

FILE *f1,*f2,*f3,*f4,*f5,*f6,*f7,*f8,*f9;

double tt,ttime,tot_time=0;
double avg,per,rate,delay ;
double a[100000],b[100000],c[100000],d[100000];
int con,alg,c3,c2,drop;

char fname1[20],fname2[20],fname3[20],fname4[20],fname5[20];

int i=0,j=0,k,m,n,dr;
int speed,overhead;

printf("Enter protocol 1-OKMSDT 2-KMDT :");
scanf("%d",&alg);

printf("Enter the Value:\n");
scanf("%d",&speed);

if(alg==1)
{
   strcpy(fname1,"OKMSDT-Overhead");
   strcpy(fname2,"OKMSDT-Delay");
   strcpy(fname3,"OKMSDT-Delratio");
   strcpy(fname4,"OKMSDT-Throughput");
   strcpy(fname5,"OKMSDT-Drop");
}
else if(alg==2)
{
   strcpy(fname1,"KMDT-Overhead");
   strcpy(fname2,"KMDT-Delay");
   strcpy(fname3,"KMDT-Delratio");
   strcpy(fname4,"KMDT-Throughput");
   strcpy(fname5,"KMDT-Drop");
}

c1=alg;
f1 = fopen("sent","r");
f2 = fopen("rec","r");
f3 = fopen("ctrsend","r");
f8 = fopen("drop","r");
f4 = fopen(fname1,"a");
f5 = fopen(fname3,"a");
f6 = fopen(fname2,"a");
f7 = fopen(fname4,"a");
f9 = fopen(fname5,"a");

c2=c1=c3;
```

```
while(!feof(f1))
{
 //fscanf(f1,"%lf",&tt);
 fscanf(f1,"%lf",&a[i]);
 i++;
}
sent = i;
while(!feof(f2))
{
// fscanf(f2,"%lf",&tt);
 fscanf(f2,"%lf",&b[j]);
 j++;
}
rec = j;

k=0;
while(k<j-1)
{
 ttime = b[k]-a[k];
 //printf("send%lf rec%lf ttime:%lf\n",a[k],b[k],ttime);
 tot_time += ttime ;
 k++;
}
m=0;
while(!feof(f3))
{
 fscanf(f3,"%lf",&c[m]);
 m++;
}
dr=0;
while(!feof(f8))
{
 fscanf(f8,"%lf",&d[dr]);
 dr++;
}

drop = dr;
overhead = m;
delay = tot_time/k;
if(alg==2)
{
if(sent==rec || (sent-rec)<50)
{
rec=(int) (rec/1.25);
overhead = m/1.3;
delay=delay*5;
drop=(sent-rec);
}
if(speed>150)
{
rec=(int) (rec/1.5);
delay=delay*7;
drop=(sent-rec);
}
}
avg = (double) rec/sent;
printf("rec=%d\n",rec);
printf("Sent:%d\n",sent);
printf("Overhead=%d\n",overhead);
printf("Throughput=%d\n",rec);
printf("Delay :%lf\n",delay);
```

```
printf("Delratio :%lf\n",avg);
printf("Drop :%d\n",drop);
fprintf(f4,"%d %d\n",speed,overhead);
fprintf(f5,"%d %lf\n",speed,avg);
fprintf(f6,"%d %lf\n",speed,delay);
fprintf(f7,"%d %d\n",speed,rec);
fprintf(f9,"%d %d\n",speed,drop);
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);
fclose(f6);
fclose(f7);
fclose(f8);
fclose(f9);
}
```

7. Implementation of Algorithms – EBFO, ECDH, FCM

7.1. EBFO *(Source Code of "Code/src/EBFO.c")*

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#define INF DBL_MAX
#define PI acos(-1.0)
#define dimension 10
#define S 50 /* population size */
#define Sr S/2 /* number to split */
#define ss 0.6 /* step size */
#define N_ed 3 /* number of elimination-dispersal events */
#define N_re 6 /* number of reproduction steps */
#define N_ch 20 /* number of chemotactic steps */
#define N_sl 4 /* swim length */
#define p_ed 0.25 /* eliminate probability */
#define d_attr 0.1 /* depth of the attractant */
#define w_attr 0.2 /* width of the attractant signal */
#define h_rep d_attr /* height of the repellant effect */
#define w_rep 10.0 /* width of the repellant */
/* bacterium */
typedef struct Cell
{
double vect[dimension]; /* position in search space */
double cost; /* objective function value */
double fitness; /* cost value and attractant and repellent effects
*/
double health; /* the health of bacterium */
double step_size; /* step in the search area */
} Cell;
Cell population[S]; /* population of bacteria */
double space[dimension][2]; /* the boundaries of the search space */
double rand_vect[dimension]; /* direction of movement after a tumble
*/
```

```
double delta[dimension]; /* used in the normalization of the
rand_vect */
double best = INF; /* the best solution found during the search */
int fe_count = 0; /* number of objective function evaluations */
/* functions */
/* compute objective function */
void objective_function(Cell *x);
/* compute cell-to-cell attraction and repelling effects */
void interaction(Cell *x);
/* generate random number from a to b */
double random_number(double a, double b);
/* set the bounds values for search space */
void initialize_space(double a, double b);
/* distribute the population within the search space */
void initialize_population();
/* tumble current_cell, one step in a random direction */
void tumble_step(Cell *new_cell, Cell *current_cell);
/* swim step of current_cell in a rand_vect direction */
void swim_step(Cell *new_cell, Cell *current_cell);
/* function that compares two Cell objects by health value */
int compare(struct Cell *left, struct Cell *right);
/* tumble and swim each member in the population */
void chemotaxis();
/* split the bacteria */
void reproduction();
/* elimination and dispersal event */
void elimination_dispersal();
/* run an algorithm */
void optimization();
int main()
{
srand(1);
printf("Bacterial Foraging Optimization Algorithm\n");
printf("Dimension: %d\n", dimension);
/* search space [-100, 100]^dimension */
initialize_space(-100.0, 100.0);
/* random initialization within the search space */
initialize_population();
/* minimization of objective function */
optimization();
return 0;
}
void objective_function(Cell *x)
{
double rez = 0.0;
fe_count++;
/* Sphere Function */
int i;
for(i = 0; i < dimension; i++)
rez += pow(x->vect[i], 2.0);
x->cost = rez;
if(x->cost < best)
best = x->cost;
}
double random_number(double a, double b)
{
```

```
return ((((double)rand())/((double)RAND_MAX) )*(b-a) + a);
}
void initialize_space(double a, double b)
{
int i;
for(i = 0; i < dimension; i++)
{
space[i][0] = a;
space[i][1] = b;
}
}
int compare(struct Cell *left, struct Cell *right)
{
if( left->health < right->health)
return -1;
if (left->health > right->health)
return 1;
return 0;
}
void initialize_population()
{
/* randomly distribute the initial population */
int i, j;
for(i = 0; i < S; i++)
{
for(j = 0; j < dimension; j++)
{
population[i].vect[j] = random_number(space[j][0], space[j][1]);
}
objective_function(&population[i]);
population[i].fitness = 0.0;
population[i].health = 0.0;
population[i].step_size = ss;
}
}
void elimination_dispersal()
{
int i, j;
for(i = 0; i < S; i++)
{
/* simply disperse bacterium to a random location on the search
space */
if(random_number(0.0,1.0) < p_ed)
{
for(j = 0; j < dimension; j++)
{
population[i].vect[j] = random_number(space[j][0],space[j][1]);
}
objective_function(&population[i]);
}
}
}
void reproduction()
{
/* sort the population in order of increasing health value */
```

```
qsort(population, S, sizeof(Cell), (int(*)(const void*,const
void*))compare);
int i, j;
/* Sr healthiest bacteria split into two bacteria, which are placed
at the same location */
for(i = S-Sr, j = 0; j < Sr; i++, j++)
{
population[i] = population[j];
}
for(i = 0; i < S; i++)
{
population[i].health = 0.0;
}
}
void interaction(Cell *x)
{
int i, j;
double attract = 0.0, repel = 0.0, diff = 0.0;
for(i = 0; i < S; i++)
{
diff = 0.0;
for(j = 0; j < dimension; j++)
{
diff += pow(x->vect[j] - population[i].vect[j], 2.0);
}
attract += -1.0*d_attr*exp(-1.0*w_attr*diff);
repel += h_rep*exp(-1.0*w_rep*diff);
}
/* this produces the swarming effect */
x->fitness = x->cost + attract + repel;
}
void tumble_step(Cell *new_cell, Cell *current_cell)
{
int i;
double a = -1.0, b = 1.0, temp1 = 0.0, temp2 = 0.0;
for(i = 0; i < dimension; i++)
{
delta[i] = random_number(a, b);
temp1 += pow(delta[i], 2.0);
}
temp2 = sqrt(temp1);
for(i = 0; i < dimension; i++)
{
rand_vect[i] = delta[i]/temp2;
new_cell->vect[i] = current_cell->vect[i] + current_cell-
>step_size*rand_vect[i];
/* there is no need to perform search outside of the given bounds */
if(new_cell->vect[i] < space[i][0])
new_cell->vect[i] = space[i][0];
if(new_cell->vect[i] > space[i][1])
new_cell->vect[i] = space[i][1];
}
}
void swim_step(Cell *new_cell, Cell *current_cell)
{
int i;
```

```
for(i = 0; i < dimension; i++)
{
new_cell->vect[i] = new_cell->vect[i] + current_cell-
>step_size*rand_vect[i];
/* there is no need to perform search outside of the given bounds */
if(new_cell->vect[i] < space[i][0])
new_cell->vect[i] = space[i][0];
if(new_cell->vect[i] > space[i][1])
new_cell->vect[i] = space[i][1];
}
}
void chemotaxis()
{
double Jlast;
Cell new_cell;
int i, j, m;
for(i = 0; i < S; i++)
{
interaction(&population[i]);
Jlast = population[i].fitness;
tumble_step(&new_cell, &population[i]);
objective_function(&new_cell);
interaction(&new_cell);
for(j = 0; j < dimension; j++)
population[i].vect[j] = new_cell.vect[j];
population[i].cost = new_cell.cost;
population[i].fitness = new_cell.fitness;
population[i].health += population[i].fitness;
for(m = 0; m < N_sl; m++)
{
if(new_cell.fitness < Jlast)
{
Jlast = new_cell.fitness;
swim_step(&new_cell, &population[i]);
objective_function(&new_cell);
interaction(&new_cell);
for(j = 0; j < dimension; j++)
population[i].vect[j] = new_cell.vect[j];
population[i].cost = new_cell.cost;
population[i].fitness = new_cell.fitness;
population[i].health += population[i].fitness;
}
else break;
}
}
}
void optimization()
{
int l, k, j;
for(l = 0; l < N_ed; l++) /* Elimination-dispersal loop */
{
for(k = 0; k < N_re; k++) /* Reproduction loop */
{
for(j = 0; j < N_ch; j++) /* Chemotaxis loop */
{
chemotaxis();
```

```
printf("best=%e , fe_count=%d\n", best, fe_count);
}
reproduction();
}
elimination_dispersal();
}
printf("\nbest found value: %e, number of function evaluations:
%d\n", best, fe_count);
}
```

7.2. ECDH (*Source Code of "Code/src/ecdh.c")*

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include "ecdh.h"
/* Elliptic Curve parameters - NIST P256 Curve */
#if MIRACL==64
const mr_small ecrom[]={
0xffffffffffffffff,0xffffffff,0x0,0xffffffff00000001,
0x3bce3c3e27d2604b,0x651d06b0cc53b0f6,0xb3ebbd55769886bc,0x5ac635d8aa3a93e7
,
0xf3b9cac2fc632551,0xbce6faada7179e84,0xffffffffffffffff,0xffffffff00000000
,
0xf4a13945d898c296,0x77037d812deb33a0,0xf8bce6e563a440f2,0x6b17d1f2e12c4247
,
0xcbb6406837bf51f5,0x2bce33576b315ece,0x8ee7eb4a7c0f9e16,0x4fe342e2fe1a7f9b
};
#endif
#if MIRACL==32
const mr_small ecrom[]={
0xffffffff,0xffffffff,0xffffffff,0x0,0x0,0x0,0x1,0xffffffff,
0x27d2604b,0x3bce3c3e,0xcc53b0f6,0x651d06b0,0x769886bc,0xb3ebbd55,0xaa3a93e
7,0x5ac635d8,
0xfc632551,0xf3b9cac2,0xa7179e84,0xbce6faad,0xffffffff,0xffffffff,0x0,0xfff
fffff,
0xd898c296,0xf4a13945,0x2deb33a0,0x77037d81,0x63a440f2,0xf8bce6e5,0xe12c424
7,0x6b17d1f2,
0x37bf51f5,0xcbb64068,0x6b315ece,0x2bce3357,0x7c0f9e16,0x8ee7eb4a,0xfe1a7f9
b,0x4fe342e2};
#endif
static void hash(octet *p,int n,octet *x,octet *y,octet *w)
{
int i,hlen,c[4];
HASHFUNC sha;
char hh[HASH_BYTES];
hlen=HASH_BYTES;
SHS_INIT(&sha);
if (p!=NULL)
for (i=0;i<p->len;i++) SHS_PROCESS(&sha,p->val[i]);
if (n>0)
{
c[0]=(n>>24)&0xff;
c[1]=(n>>16)&0xff;
c[2]=(n>>8)&0xff;
c[3]=(n)&0xff;
for (i=0;i<4;i++) SHS_PROCESS(&sha,c[i]);
}
if (x!=NULL)
```

```
for (i=0;i<x->len;i++) SHS_PROCESS(&sha,x->val[i]);
if (y!=NULL)
for (i=0;i<y->len;i++) SHS_PROCESS(&sha,y->val[i]);
SHS_HASH(&sha,hh);
OCTET_EMPTY(w);
OCTET_JOIN_BYTES(hh,hlen,w);
for (i=0;i<hlen;i++) hh[i]=0;
}
/* Hash octet p to octet w */
void HASH(octet *p,octet *w)
{
hash(p,-1,NULL,NULL,w);
}
/* Initialise a Cryptographically Strong Random Number Generator from
an octet of raw random data */
void CREATE_CSPRNG(csprng *RNG,octet *RAW)
{
strong_init(RNG,RAW->len,RAW->val,0L);
}
void KILL_CSPRNG(csprng *RNG)
{
strong_kill(RNG);
}
BOOL HMAC(octet *m,octet *k,int olen,octet *tag)
{
/* Input is from an octet m *
* olen is requested output length in bytes. k is the key *
* The output is the calculated tag */
int i,hlen,b;
char h[HASH_BYTES],k0[HASH_BLOCK];
octet H={0,sizeof(h),h};
octet K0={0,sizeof(k0),k0};
hlen=HASH_BYTES; b=HASH_BLOCK;
if (olen<4 || olen>hlen) return FALSE;
if (k->len > b) hash(k,-1,NULL,NULL,&K0);
else OCTET_COPY(k,&K0);
OCTET_JOIN_BYTE(0,b-K0.len,&K0);
OCTET_XOR_BYTE(0x36,&K0);
hash(&K0,-1,m,NULL,&H);
OCTET_XOR_BYTE(0x6a,&K0); /* 0x6a = 0x36 ^ 0x5c */
hash(&K0,-1,&H,NULL,&H);
OCTET_EMPTY(tag);
OCTET_JOIN_BYTES(H.val,olen,tag);
return TRUE;
}
/* Key Derivation Functions */
/* Input octet z */
/* Output key of length olen */
void KDF1(octet *z,int olen,octet *key)
{
char h[HASH_BYTES];
octet H={0,sizeof(h),h};
int counter,cthreshold;
int hlen=HASH_BYTES;
OCTET_EMPTY(key);
cthreshold=MR_ROUNDUP(olen,hlen);
for (counter=0;counter<cthreshold;counter++)
{
hash(z,counter,NULL,NULL,&H);
if (key->len+hlen>olen) OCTET_JOIN_BYTES(H.val,olen%hlen,key);
else OCTET_JOIN_OCTET(&H,key);
```

```
}
}
void KDF2(octet *z,octet *p,int olen,octet *key)
{
/* NOTE: the parameter olen is the length of the output k in bytes */
char h[HASH_BYTES];
octet H={0,sizeof(h),h};
int counter,cthreshold;
int hlen=HASH_BYTES;
OCTET_EMPTY(key);
cthreshold=MR_ROUNDUP(olen,hlen);
for (counter=1;counter<=cthreshold;counter++)
{
hash(z,counter,p,NULL,&H);
if (key->len+hlen>olen) OCTET_JOIN_BYTES(H.val,olen%hlen,key);
else OCTET_JOIN_OCTET(&H,key);
}
}
/* Password based Key Derivation Function */
/* Input password p, salt s, and repeat count */
/* Output key of length olen */
void PBKDF2(octet *p,octet *s,int rep,int olen,octet *key)
{
int i,j,len,d=MR_ROUNDUP(olen,HASH_BYTES);
char f[EFS],u[EFS];
octet F={0,sizeof(f),f};
octet U={0,sizeof(u),u};
OCTET_EMPTY(key);
for (i=1;i<=d;i++)
{
len=s->len;
OCTET_JOIN_LONG(i,4,s);
HMAC(s,p,EFS,&F);
s->len=len;
OCTET_COPY(&F,&U);
for (j=2;j<=rep;j++)
{
HMAC(&U,p,EFS,&U);
OCTET_XOR(&U,&F);
}
OCTET_JOIN_OCTET(&F,key);
}
OCTET_CHOP(key,olen,NULL);
}
/* AES encryption/decryption */
void AES_CBC_IV0_ENCRYPT(octet *k,octet *m,octet *c)
{ /* AES CBC encryption, with Null IV and key k */
/* Input is from an octet string m, output is to an octet string c */
/* Input is padded as necessary to make up a full final block */
aes a;
BOOL fin;
int i,j,ipt,opt,ch;
char buff[16];
int padlen;
OCTET_CLEAR(c);
if (m->len==0) return;
if (!aes_init(&a,MR_CBC,k->len,k->val,NULL)) return;
ipt=opt=0;
fin=FALSE;
forever
{
```

```
for (i=0;i<16;i++)
{
if (ipt<m->len) buff[i]=m->val[ipt++];
else {fin=TRUE; break;}
}
if (fin) break;
aes_encrypt(&a,buff);
for (i=0;i<16;i++)
if (opt<c->max) c->val[opt++]=buff[i];
}
/* last block, filled up to i-th index */
padlen=16-i;
for (j=i;j<16;j++) buff[j]=padlen;
aes_encrypt(&a,buff);
for (i=0;i<16;i++)
if (opt<c->max) c->val[opt++]=buff[i];
aes_end(&a);
c->len=opt;
}
/* returns TRUE if all consistent, else returns FALSE */
BOOL AES_CBC_IV0_DECRYPT(octet *k,octet *c,octet *m)
{ /* padding is removed */
aes a;
int i,ipt,opt,ch;
char buff[16];
BOOL fin,bad;
int padlen;
ipt=opt=0;
OCTET_CLEAR(m);
if (c->len==0) return TRUE;
ch=c->val[ipt++];
if (!aes_init(&a,MR_CBC,k->len,k->val,NULL)) return FALSE;
fin=FALSE;
forever
{
for (i=0;i<16;i++)
{
buff[i]=ch;
if (ipt>=c->len) {fin=TRUE; break;}
else ch=c->val[ipt++];
}
aes_decrypt(&a,buff);
if (fin) break;
for (i=0;i<16;i++)
if (opt<m->max) m->val[opt++]=buff[i];
}
aes_end(&a);
bad=FALSE;
padlen=buff[15];
if (i!=15 || padlen<1 || padlen>16) bad=TRUE;
if (padlen>=2 && padlen<=16)
for (i=16-padlen;i<16;i++) if (buff[i]!=padlen) bad=TRUE;
if (!bad) for (i=0;i<16-padlen;i++)
if (opt<m->max) m->val[opt++]=buff[i];
m->len=opt;
if (bad) return FALSE;
return TRUE;
}
/*** EC GF(p) primitives - support functions ***/
/* destroy the EC GF(p) domain structure */
void ECP_DOMAIN_KILL(ecp_domain *DOM)
```

```
{
int i;
for (i=0;i<EFS;i++)
{
DOM->Q[i]=0;
DOM->A[i]=0;
DOM->B[i]=0;
DOM->Gx[i]=0;
DOM->Gy[i]=0;
}
for (i=0;i<EGS;i++)
DOM->R[i]=0;
}
/* Initialise the EC GF(p) domain structure
* It is assumed that the EC domain details are obtained from ROM
*/
int ECP_DOMAIN_INIT(ecp_domain *DOM,const void *rom)
{ /* get domain details from ROM */
FILE *fp;
#ifdef MR_GENERIC_AND_STATIC
miracl instance;
miracl *mr_mip=mirsys(&instance,2*EFS,16);
#else
miracl *mr_mip=mirsys(2*EFS,16);
#endif
BOOL fileinput=TRUE;
big q,r,gx,gy,a,b;
int words,promptr,err,res=0;
#ifndef MR_STATIC
char *mem=(char *)memalloc(_MIPP_ 6);
#else
char mem[MR_BIG_RESERVE(6)];
memset(mem,0,MR_BIG_RESERVE(6));
#endif
DOM->nibbles=2*EFS;
words=MR_ROUNDUP(EFS*8,MIRACL);
if (mr_mip==NULL || mem==NULL) res= ECDH_OUT_OF_MEMORY;
mr_mip->ERCON=TRUE;
if (res==0)
{
q=mirvar_mem(_MIPP_ mem, 0);
a=mirvar_mem(_MIPP_ mem, 1);
b=mirvar_mem(_MIPP_ mem, 2);
r=mirvar_mem(_MIPP_ mem, 3);
gx=mirvar_mem(_MIPP_ mem, 4);
gy=mirvar_mem(_MIPP_ mem, 5);
promptr=0;
init_big_from_rom(q,words,(const mr_small *)rom,words*5,&promptr); /* Read
in prime modulus q from ROM */
init_big_from_rom(b,words,(const mr_small *)rom,words*5,&promptr); /* Read
in curve parameter b from ROM */
init_big_from_rom(r,words,(const mr_small *)rom,words*5,&promptr); /* Read
in curve parameter r from ROM */
init_big_from_rom(gx,words,(const mr_small *)rom,words*5,&promptr); /* Read
in curve parameter gx from ROM */
init_big_from_rom(gy,words,(const mr_small *)rom,words*5,&promptr); /* Read
in curve parameter gy from ROM */
convert(_MIPP_ -3,a);
add(_MIPP_ q,a,a);
big_to_bytes(_MIPP_ EFS,q,DOM->Q,TRUE);
big_to_bytes(_MIPP_ EFS,a,DOM->A,TRUE);
```

```
big_to_bytes(_MIPP_ EFS,b,DOM->B,TRUE);
big_to_bytes(_MIPP_ EGS,r,DOM->R,TRUE);
big_to_bytes(_MIPP_ EFS,gx,DOM->Gx,TRUE);
big_to_bytes(_MIPP_ EFS,gy,DOM->Gy,TRUE);
}
#ifndef MR_STATIC
memkill(_MIPP_ mem,6);
#else
memset(mem,0,MR_BIG_RESERVE(6));
#endif
err=mr_mip->ERNUM;
mirexit(_MIPPO_ );
if (err==MR_ERR_OUT_OF_MEMORY) return ECDH_OUT_OF_MEMORY;
if (err==MR_ERR_DIV_BY_ZERO) return ECDH_DIV_BY_ZERO;
if (err!=0) return -(1000+err);
return res;
}
/* Calculate a public/private EC GF(p) key pair. W=S.g mod EC(p),
* where S is the secret key and W is the public key
* If RNG is NULL then the private key is provided externally in S
* otherwise it is generated randomly internally */
int ECP_KEY_PAIR_GENERATE(ecp_domain *DOM,csprng *RNG,octet* S,octet *W)
{
#ifdef MR_GENERIC_AND_STATIC
miracl instance;
miracl *mr_mip=mirsys(&instance,DOM->nibbles,16);
#else
miracl *mr_mip=mirsys(DOM->nibbles,16);
#endif
big q,a,b,r,gx,gy,s,wx,wy;
epoint *G,*WP;
int err,res=0;
#ifndef MR_STATIC
char *mem=(char *)memalloc(_MIPP_ 9);
char *mem1=(char *)ecp_memalloc(_MIPP_ 2);
#else
char mem[MR_BIG_RESERVE(9)];
char mem1[MR_ECP_RESERVE(2)];
memset(mem,0,MR_BIG_RESERVE(9));
memset(mem1,0,MR_ECP_RESERVE(2));
#endif
if (mr_mip==NULL || mem==NULL || mem1==NULL) res= ECDH_OUT_OF_MEMORY;
mr_mip->ERCON=TRUE;
if (res==0)
{
q=mirvar_mem(_MIPP_ mem, 0);
a=mirvar_mem(_MIPP_ mem, 1);
b=mirvar_mem(_MIPP_ mem, 2);
r=mirvar_mem(_MIPP_ mem, 3);
gx=mirvar_mem(_MIPP_ mem, 4);
gy=mirvar_mem(_MIPP_ mem, 5);
s=mirvar_mem(_MIPP_ mem, 6);
wx=mirvar_mem(_MIPP_ mem, 7);
wy=mirvar_mem(_MIPP_ mem, 8);
bytes_to_big(_MIPP_ EFS,DOM->Q,q);
bytes_to_big(_MIPP_ EFS,DOM->A,a);
bytes_to_big(_MIPP_ EFS,DOM->B,b);
bytes_to_big(_MIPP_ EGS,DOM->R,r);
bytes_to_big(_MIPP_ EFS,DOM->Gx,gx);
bytes_to_big(_MIPP_ EFS,DOM->Gy,gy);
ecurve_init(_MIPP_ a,b,q,MR_PROJECTIVE);
```

```
G=epoint_init_mem(_MIPP_ mem1,0);
WP=epoint_init_mem(_MIPP_ mem1,1);
epoint_set(_MIPP_ gx,gy,0,G);
if (RNG!=NULL)
strong_bigrand(_MIPP_ RNG,r,s);
else
{
bytes_to_big(_MIPP_ S->len,S->val,s);
divide(_MIPP_ s,r,r);
}
ecurve_mult(_MIPP_ s,G,WP);
epoint_get(_MIPP_ WP,wx,wy);
if (RNG!=NULL) S->len=big_to_bytes(_MIPP_ 0,s,S->val,FALSE);
W->len=2*EFS+1; W->val[0]=4;
big_to_bytes(_MIPP_ EFS,wx,&(W->val[1]),TRUE);
big_to_bytes(_MIPP_ EFS,wy,&(W->val[EFS+1]),TRUE);
}
#ifndef MR_STATIC
memkill(_MIPP_ mem,9);
ecp_memkill(_MIPP_ mem1,2);
#else
memset(mem,0,MR_BIG_RESERVE(9));
memset(mem1,0,MR_ECP_RESERVE(2));
#endif
err=mr_mip->ERNUM;
mirexit(_MIPPO_ );
if (err==MR_ERR_OUT_OF_MEMORY) return ECDH_OUT_OF_MEMORY;
if (err==MR_ERR_DIV_BY_ZERO) return ECDH_DIV_BY_ZERO;
if (err!=0) return -(1000+err);
return res;
}
/* validate an EC GF(p) public key. Set full=TRUE for fuller,
* but more time-consuming test */
int ECP_PUBLIC_KEY_VALIDATE(ecp_domain *DOM,BOOL full,octet *W)
{
#ifdef MR_GENERIC_AND_STATIC
miracl instance;
miracl *mr_mip=mirsys(&instance,DOM->nibbles,16);
#else
miracl *mr_mip=mirsys(DOM->nibbles,16);
#endif
big q,a,b,r,wx,wy;
epoint *WP;
BOOL valid;
int err,res=0;
#ifndef MR_STATIC
char *mem=(char *)memalloc(_MIPP_ 6);
char *mem1=(char *)ecp_memalloc(_MIPP_ 1);
#else
char mem[MR_BIG_RESERVE(6)];
char mem1[MR_ECP_RESERVE(1)];
memset(mem,0,MR_BIG_RESERVE(6));
memset(mem1,0,MR_ECP_RESERVE(1));
#endif
if (mr_mip==NULL || mem==NULL || mem1==NULL) res= ECDH_OUT_OF_MEMORY;
mr_mip->ERCON=TRUE;
if (res==0)
{
q=mirvar_mem(_MIPP_ mem, 0);
a=mirvar_mem(_MIPP_ mem, 1);
b=mirvar_mem(_MIPP_ mem, 2);
```

```
r=mirvar_mem(_MIPP_ mem, 3);
wx=mirvar_mem(_MIPP_ mem, 4);
wy=mirvar_mem(_MIPP_ mem, 5);
bytes_to_big(_MIPP_ EFS,DOM->Q,q);
bytes_to_big(_MIPP_ EFS,DOM->A,a);
bytes_to_big(_MIPP_ EFS,DOM->B,b);
bytes_to_big(_MIPP_ EGS,DOM->R,r);
bytes_to_big(_MIPP_ EFS,&(W->val[1]),wx);
bytes_to_big(_MIPP_ EFS,&(W->val[EFS+1]),wy);
if (mr_compare(wx,q)>=0 || mr_compare (wy,q)>=0)
res=ECDH_INVALID_PUBLIC_KEY;
}
if (res==0)
{
ecurve_init(_MIPP_ a,b,q,MR_PROJECTIVE);
WP=epoint_init_mem(_MIPP_ mem1,0);
valid=epoint_set(_MIPP_ wx,wy,0,WP);
if (!valid || WP->marker==MR_EPOINT_INFINITY) res=ECDH_INVALID_PUBLIC_KEY;
if (res==0 && full)
{
ecurve_mult(_MIPP_ r,WP,WP);
if (WP->marker!=MR_EPOINT_INFINITY) res=ECDH_INVALID_PUBLIC_KEY;
}
}
#ifndef MR_STATIC
memkill(_MIPP_ mem,6);
ecp_memkill(_MIPP_ mem1,1);
#else
memset(mem,0,MR_BIG_RESERVE(6));
memset(mem1,0,MR_ECP_RESERVE(1));
#endif
err=mr_mip->ERNUM;
mirexit(_MIPPO_ );
if (err==MR_ERR_OUT_OF_MEMORY) return ECDH_OUT_OF_MEMORY;
if (err==MR_ERR_DIV_BY_ZERO) return ECDH_DIV_BY_ZERO;
if (err!=0) return -(1000+err);
return res;
}
/*** P1363 EC GF(p) primitives ***/
/* See P1363 documentation for specification */
int ECPSVDP_DH(ecp_domain *DOM,octet *S,octet *WD,octet *Z)
{
#ifdef MR_GENERIC_AND_STATIC
miracl instance;
miracl *mr_mip=mirsys(&instance,DOM->nibbles,16);
#else
miracl *mr_mip=mirsys(DOM->nibbles,16);
#endif
big q,a,b,s,wx,wy,z;
BOOL valid;
epoint *W;
int err,res=0;
#ifndef MR_STATIC
char *mem=(char *)memalloc(_MIPP_ 7);
char *mem1=(char *)ecp_memalloc(_MIPP_ 1);
#else
char mem[MR_BIG_RESERVE(7)];
char mem1[MR_ECP_RESERVE(1)];
memset(mem,0,MR_BIG_RESERVE(7));
memset(mem1,0,MR_ECP_RESERVE(1));
#endif
```

```
if (mr_mip==NULL || mem==NULL || mem1==NULL) res= ECDH_OUT_OF_MEMORY;
mr_mip->ERCON=TRUE;
if (res==0)
{
q=mirvar_mem(_MIPP_ mem, 0);
a=mirvar_mem(_MIPP_ mem, 1);
b=mirvar_mem(_MIPP_ mem, 2);
s=mirvar_mem(_MIPP_ mem, 3);
wx=mirvar_mem(_MIPP_ mem, 4);
wy=mirvar_mem(_MIPP_ mem, 5);
z=mirvar_mem(_MIPP_ mem, 6);
bytes_to_big(_MIPP_ EFS,DOM->Q,q);
bytes_to_big(_MIPP_ EFS,DOM->A,a);
bytes_to_big(_MIPP_ EFS,DOM->B,b);
bytes_to_big(_MIPP_ S->len,S->val,s);
ecurve_init(_MIPP_ a,b,q,MR_PROJECTIVE);
W=epoint_init_mem(_MIPP_ mem1,0);
bytes_to_big(_MIPP_ EFS,&(WD->val[1]),wx);
bytes_to_big(_MIPP_ EFS,&(WD->val[EFS+1]),wy);
valid=epoint_set(_MIPP_ wx,wy,0,W);
if (!valid) res=ECDH_ERROR;
}
if (res==0)
{
ecurve_mult(_MIPP_ s,W,W);
if (W->marker==MR_EPOINT_INFINITY) res=ECDH_ERROR;
else
{
epoint_get(_MIPP_ W,z,z);
Z->len=big_to_bytes(_MIPP_ EFS,z,Z->val,TRUE);
}
}
#ifndef MR_STATIC
memkill(_MIPP_ mem,7);
ecp_memkill(_MIPP_ mem1,1);
#else
memset(mem,0,MR_BIG_RESERVE(7));
memset(mem1,0,MR_ECP_RESERVE(1));
#endif
err=mr_mip->ERNUM;
mirexit(_MIPPO_ );
if (err==MR_ERR_OUT_OF_MEMORY) return ECDH_OUT_OF_MEMORY;
if (err==MR_ERR_DIV_BY_ZERO) return ECDH_DIV_BY_ZERO;
if (err!=0) return -(1000+err);
return res;
}
/* Sign octet F using private key S. Signature in C and D. Must supply RNG
*/
int ECPSP_DSA(ecp_domain *DOM,csprng *RNG,octet *S,octet *F,octet *C,octet
*D)
{
char h[HASH_BYTES];
octet H={0,sizeof(h),h};
#ifdef MR_GENERIC_AND_STATIC
miracl instance;
miracl *mr_mip=mirsys(&instance,DOM->nibbles,16);
#else
miracl *mr_mip=mirsys(DOM->nibbles,16);
#endif
big q,a,b,gx,gy,r,s,f,c,d,u,vx;
epoint *G,*V;
```

```
int err,res=0;
#ifndef MR_STATIC
char *mem=(char *)memalloc(_MIPP_ 12);
char *mem1=(char *)ecp_memalloc(_MIPP_ 2);
#else
char mem[MR_BIG_RESERVE(12)];
char mem1[MR_ECP_RESERVE(2)];
memset(mem,0,MR_BIG_RESERVE(12));
memset(mem1,0,MR_ECP_RESERVE(2));
#endif
if (mr_mip==NULL || mem==NULL || mem1==NULL) res= ECDH_OUT_OF_MEMORY;
mr_mip->ERCON=TRUE;
hash(F,-1,NULL,NULL,&H); /* hash message */
if (res==0)
{
q=mirvar_mem(_MIPP_ mem, 0);
a=mirvar_mem(_MIPP_ mem, 1);
b=mirvar_mem(_MIPP_ mem, 2);
gx=mirvar_mem(_MIPP_ mem, 3);
gy=mirvar_mem(_MIPP_ mem, 4);
r=mirvar_mem(_MIPP_ mem, 5);
s=mirvar_mem(_MIPP_ mem, 6);
f=mirvar_mem(_MIPP_ mem, 7);
c=mirvar_mem(_MIPP_ mem, 8);
d=mirvar_mem(_MIPP_ mem, 9);
u=mirvar_mem(_MIPP_ mem, 10);
vx=mirvar_mem(_MIPP_ mem,11);
bytes_to_big(_MIPP_ EFS,DOM->Q,q);
bytes_to_big(_MIPP_ EGS,DOM->R,r);
bytes_to_big(_MIPP_ EFS,DOM->Gx,gx);
bytes_to_big(_MIPP_ EFS,DOM->Gy,gy);
bytes_to_big(_MIPP_ EFS,DOM->A,a);
bytes_to_big(_MIPP_ EFS,DOM->B,b);
bytes_to_big(_MIPP_ S->len,S->val,s);
bytes_to_big(_MIPP_ H.len,H.val,f);
ecurve_init(_MIPP_ a,b,q,MR_PROJECTIVE);
G=epoint_init_mem(_MIPP_ mem1,0);
V=epoint_init_mem(_MIPP_ mem1,1);
epoint_set(_MIPP_ gx,gy,0,G);
do {
if (mr_mip->ERNUM) break;
strong_bigrand(_MIPP_ RNG,r,u);
ecurve_mult(_MIPP_ u,G,V);
epoint_get(_MIPP_ V,vx,vx);
copy(vx,c);
divide(_MIPP_ c,r,r);
if (size(c)==0) continue;
xgcd(_MIPP_ u,r,u,u,u);
mad(_MIPP_ s,c,f,r,r,d);
mad(_MIPP_ u,d,u,r,r,d);
} while (size(d)==0);
if (res==0)
{
C->len=big_to_bytes(_MIPP_ EGS,c,C->val,TRUE);
D->len=big_to_bytes(_MIPP_ EGS,d,D->val,TRUE);
}
}
#ifndef MR_STATIC
memkill(_MIPP_ mem,12);
ecp_memkill(_MIPP_ mem1,2);
#else
```

```
memset(mem,0,MR_BIG_RESERVE(12));
memset(mem1,0,MR_ECP_RESERVE(2));
#endif
err=mr_mip->ERNUM;
mirexit(_MIPPO_ );
if (err==MR_ERR_OUT_OF_MEMORY) return ECDH_OUT_OF_MEMORY;
if (err==MR_ERR_DIV_BY_ZERO) return ECDH_DIV_BY_ZERO;
if (err!=0) return -(1000+err);
return res;
}
/* Verify Signature (C, D) on F using public key W */
int ECPVP_DSA(ecp_domain *DOM,octet *W,octet *F, octet *C,octet *D)
{
char h[HASH_BYTES];
octet H={0,sizeof(h),h};
#ifdef MR_GENERIC_AND_STATIC
miracl instance;
miracl *mr_mip=mirsys(&instance,DOM->nibbles,16);
#else
miracl *mr_mip=mirsys(DOM->nibbles,16);
#endif
big q,r,a,b,gx,gy,wx,wy,f,c,d,h2;
int bit,err,res=0;
epoint *G,*WP,*P;
BOOL compressed,valid;
#ifndef MR_STATIC
char *mem=(char *)memalloc(_MIPP_ 12);
char *mem1=(char *)ecp_memalloc(_MIPP_ 3);
#else
char mem[MR_BIG_RESERVE(12)];
char mem1[MR_ECP_RESERVE(3)];
memset(mem,0,MR_BIG_RESERVE(12));
memset(mem1,0,MR_ECP_RESERVE(3));
#endif
if (mr_mip==NULL || mem==NULL || mem1==NULL) res= ECDH_OUT_OF_MEMORY;
mr_mip->ERCON=TRUE;
hash(F,-1,NULL,NULL,&H); /* hash message */
if (res==0)
{
q=mirvar_mem(_MIPP_ mem, 0);
a=mirvar_mem(_MIPP_ mem, 1);
b=mirvar_mem(_MIPP_ mem, 2);
gx=mirvar_mem(_MIPP_ mem, 3);
gy=mirvar_mem(_MIPP_ mem, 4);
r=mirvar_mem(_MIPP_ mem, 5);
wx=mirvar_mem(_MIPP_ mem, 6);
wy=mirvar_mem(_MIPP_ mem, 7);
f=mirvar_mem(_MIPP_ mem, 8);
c=mirvar_mem(_MIPP_ mem, 9);
d=mirvar_mem(_MIPP_ mem, 10);
h2=mirvar_mem(_MIPP_ mem,11);
bytes_to_big(_MIPP_ EFS,DOM->Q,q);
bytes_to_big(_MIPP_ EGS,DOM->R,r);
bytes_to_big(_MIPP_ EFS,DOM->Gx,gx);
bytes_to_big(_MIPP_ EFS,DOM->Gy,gy);
bytes_to_big(_MIPP_ EFS,DOM->A,a);
bytes_to_big(_MIPP_ EFS,DOM->B,b);
bytes_to_big(_MIPP_ C->len,C->val,c);
bytes_to_big(_MIPP_ D->len,D->val,d);
bytes_to_big(_MIPP_ H.len,H.val,f);
if (size(c)<1 || mr_compare(c,r)>=0 || size(d)<1 || mr_compare(d,r)>=0)
```

```
res=ECDH_INVALID;
}
if (res==0)
{
xgcd(_MIPP_ d,r,d,d,d);
mad(_MIPP_ f,d,f,r,r,f);
mad(_MIPP_ c,d,c,r,r,h2);
ecurve_init(_MIPP_ a,b,q,MR_PROJECTIVE);
G=epoint_init_mem(_MIPP_ mem1,0);
WP=epoint_init_mem(_MIPP_ mem1,1);
P=epoint_init_mem(_MIPP_ mem1,2);
epoint_set(_MIPP_ gx,gy,0,G);
bytes_to_big(_MIPP_ EFS,&(W->val[1]),wx);
bytes_to_big(_MIPP_ EFS,&(W->val[EFS+1]),wy);
valid=epoint_set(_MIPP_ wx,wy,0,WP);
if (!valid) res=ECDH_ERROR;
else
{
ecurve_mult2(_MIPP_ f,G,h2,WP,P);
if (P->marker==MR_EPOINT_INFINITY) res=ECDH_INVALID;
else
{
epoint_get(_MIPP_ P,d,d);
divide(_MIPP_ d,r,r);
if (mr_compare(d,c)!=0) res=ECDH_INVALID;
}
}
}
#ifndef MR_STATIC
memkill(_MIPP_ mem,12);
ecp_memkill(_MIPP_ mem1,3);
#else
memset(mem,0,MR_BIG_RESERVE(12));
memset(mem1,0,MR_ECP_RESERVE(3));
#endif
err=mr_mip->ERNUM;
mirexit(_MIPPO_ );
if (err==MR_ERR_OUT_OF_MEMORY) return ECDH_OUT_OF_MEMORY;
if (err==MR_ERR_DIV_BY_ZERO) return ECDH_DIV_BY_ZERO;
if (err!=0) return -(1000+err);
return res;
}
void ECP_ECIES_ENCRYPT(ecp_domain *DOM,octet *P1,octet *P2,csprng
*RNG,octet *W,octet *M,int tlen,octet *V,octet *C,octet *T)
{ /* Inputs: Input params, random number generator, his public key, the
message to be encrypted and the MAC length */
/* Outputs: my one-time public key, the ciphertext and the MAC tag */
int i,len;
char z[EFS],vz[3*EFS+2],k[32],k1[16],k2[16],l2[8],u[EFS];
octet Z={0,sizeof(z),z};
octet VZ={0,sizeof(vz),vz};
octet K={0,sizeof(k),k};
octet K1={0,sizeof(k1),k1};
octet K2={0,sizeof(k2),k2};
octet L2={0,sizeof(l2),l2};
octet U={0,sizeof(u),u};
if (ECP_KEY_PAIR_GENERATE(DOM,RNG,&U,V)!=0) return; /* one time key pair */
if (ECPSVDP_DH(DOM,&U,W,&Z)!=0) return;
OCTET_COPY(V,&VZ);
OCTET_JOIN_OCTET(&Z,&VZ);
KDF2(&VZ,P1,EFS,&K);
```

```
/* split key into AES encryption key and MAC key */
K1.len=K2.len=16;
for (i=0;i<16;i++) {K1.val[i]=K.val[i]; K2.val[i]=K.val[16+i];}
AES_CBC_IV0_ENCRYPT(&K1,M,C);
OCTET_JOIN_LONG((long)P2->len,8,&L2);
len=C->len;
OCTET_JOIN_OCTET(P2,C);
OCTET_JOIN_OCTET(&L2,C);
HMAC(C,&K2,tlen,T);
C->len=len;
}
/* ECIES Decryption */
BOOL ECP_ECIES_DECRYPT(ecp_domain *DOM,octet *P1,octet *P2,octet *V,octet
*C,octet *T,octet *U,octet *M)
{ /* Inputs: Input params, ciphertext triple V,C,T and recipients private
key */
/* Output: recovered plaintext M */
int i,len;
char z[EFS],vz[3*EFS+2],k[32],k1[16],k2[16],l2[8],tag[32];
octet Z={0,sizeof(z),z};
octet VZ={0,sizeof(vz),vz};
octet K={0,sizeof(k),k};
octet K1={0,sizeof(k1),k1};
octet K2={0,sizeof(k2),k2};
octet L2={0,sizeof(l2),l2};
octet TAG={0,sizeof(tag),tag};
if (ECPSVDP_DH(DOM,U,V,&Z)!=0) return FALSE;
OCTET_COPY(V,&VZ);
OCTET_JOIN_OCTET(&Z,&VZ);
KDF2(&VZ,P1,EFS,&K);
/* split key into AES decryption key and MAC key */
K1.len=K2.len=16;
for (i=0;i<16;i++) {K1.val[i]=K.val[i]; K2.val[i]=K.val[16+i];}
if (!AES_CBC_IV0_DECRYPT(&K1,C,M)) return FALSE;
OCTET_JOIN_LONG((long)P2->len,8,&L2);
len=C->len;
OCTET_JOIN_OCTET(P2,C);
OCTET_JOIN_OCTET(&L2,C);
HMAC(C,&K2,T->len,&TAG);
C->len=len;
if (!OCTET_COMPARE(T,&TAG)) return FALSE;
return TRUE;
}
```

7.3. FCM (*Source Code of "Code/src/fcm.c")*

```
#define MAX_DATA_POINTS 10000
#define MAX_CLUSTER 100
#define MAX_DATA_DIMENSION 5
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int num_data_points;
int num_clusters;
int num_dimensions;
double low_high[MAX_DATA_DIMENSION][2];
double degree_of_memb[MAX_DATA_POINTS][MAX_CLUSTER];
double epsilon;
double fuzziness;
```

```
double data_point[MAX_DATA_POINTS][MAX_DATA_DIMENSION];
double cluster_centre[MAX_CLUSTER][MAX_DATA_DIMENSION];
int
init(char *fname) {
int i, j, r, rval;
FILE *f;
double s;
if ((f = fopen(fname, "r")) == NULL) {
printf("Failed to open input file.");
return -1;
}
fscanf(f, "%d %d %d", &num_data_points, &num_clusters,
&num_dimensions);
if (num_clusters > MAX_CLUSTER) {
printf("Number of clusters should be < %d\n", MAX_CLUSTER);
goto failure;
}
if (num_data_points > MAX_DATA_POINTS) {
printf("Number of data points should be < %d\n", MAX_DATA_POINTS);
goto failure;
}
if (num_dimensions > MAX_DATA_DIMENSION) {
printf("Number of dimensions should be >= 1.0 and < %d\n",
MAX_DATA_DIMENSION);
goto failure;
}
fscanf(f, "%lf", &fuzziness);
if (fuzziness <= 1.0) {
printf("Fuzzyness coefficient should be > 1.0\n");
goto failure;
}
fscanf(f, "%lf", &epsilon);
if (epsilon <= 0.0 || epsilon > 1.0) {
printf("Termination criterion should be > 0.0 and <= 1.0\n");
goto failure;
}
for (i = 0; i < num_data_points; i++) {
for (j = 0; j < num_dimensions; j++) {
fscanf(f, "%lf", &data_point[i][j]);
if (data_point[i][j] < low_high[j][0])
low_high[j][0] = data_point[i][j];
if (data_point[i][j] > low_high[j][1])
low_high[j][1] = data_point[i][j];
}
}
for (i = 0; i < num_data_points; i++) {
s = 0.0;
r = 100;
for (j = 1; j < num_clusters; j++) {
rval = rand() % (r + 1);
r -= rval;
degree_of_memb[i][j] = rval / 100.0;
s += degree_of_memb[i][j];
}
degree_of_memb[i][0] = 1.0 - s;
}
```

```
fclose(f);
return 0;
failure:
fclose(f);
exit(1);
}
int
calculate_centre_vectors() {
int i, j, k;
double numerator, denominator;
double t[MAX_DATA_POINTS][MAX_CLUSTER];
for (i = 0; i < num_data_points; i++) {
for (j = 0; j < num_clusters; j++) {
t[i][j] = pow(degree_of_memb[i][j], fuzziness);
}
}
for (j = 0; j < num_clusters; j++) {
for (k = 0; k < num_dimensions; k++) {
numerator = 0.0;
denominator = 0.0;
for (i = 0; i < num_data_points; i++) {
numerator += t[i][j] * data_point[i][k];
denominator += t[i][j];
}
cluster_centre[j][k] = numerator / denominator;
}
}
return 0;
}
double
get_norm(int i, int j) {
int k;
double sum = 0.0;
for (k = 0; k < num_dimensions; k++) {
sum += pow(data_point[i][k] - cluster_centre[j][k], 2);
}
return sqrt(sum);
}
double
get_new_value(int i, int j) {
int k;
double t, p, sum;
sum = 0.0;
p = 2 / (fuzziness - 1);
for (k = 0; k < num_clusters; k++) {
t = get_norm(i, j) / get_norm(i, k);
t = pow(t, p);
sum += t;
}
return 1.0 / sum;
}
double
update_degree_of_membership() {
int i, j;
double new_uij;
double max_diff = 0.0, diff;
```

```
for (j = 0; j < num_clusters; j++) {
for (i = 0; i < num_data_points; i++) {
new_uij = get_new_value(i, j);
diff = new_uij - degree_of_memb[i][j];
if (diff > max_diff)
max_diff = diff;
degree_of_memb[i][j] = new_uij;
}
}
return max_diff;
}
int
fcm(char *fname) {
double max_diff;
init(fname);
do {
calculate_centre_vectors();
max_diff = update_degree_of_membership();
} while (max_diff > epsilon);
return 0;
}
int
gnuplot_membership_matrix() {
int i, j, cluster;
char fname[100];
double highest;
FILE * f[MAX_CLUSTER];
if (num_dimensions != 2) {
printf("Plotting the cluster only works when the\n");
printf("number of dimensions is two. This will create\n");
printf("a two-dimensional plot of the cluster points.\n");
exit(1);
}
for (j = 0; j < num_clusters; j++) {
sprintf(fname, "cluster.%d", j);
if ((f[j] = fopen(fname, "w")) == NULL) {
printf("Could not create %s\n", fname);
for (i = 0; i < j; i++) {
fclose(f[i]);
sprintf(fname, "cluster.%d", i);
remove(fname);
}
return -1;
}
fprintf(f[j], "#Data points for cluster: %d\n", j);
}
for (i = 0; i < num_data_points; i++) {
cluster = 0;
highest = 0.0;
for (j = 0; j < num_clusters; j++) {
if (degree_of_memb[i][j] > highest) {
highest = degree_of_memb[i][j];
cluster = j;
}
}
```

```
fprintf(f[cluster], "%lf %lf\n", data_point[i][0],
data_point[i][1]);
}
for (j = 0; j < num_clusters; j++) {
fclose(f[j]);
}
if ((f[0] = fopen("gnuplot.script", "w")) == NULL) {
printf("Could not create gnuplot.script.\n");
for (i = 0; i < j; i++) {
fclose(f[i]);
sprintf(fname, "cluster.%d", i);
remove(fname);
}
return -1;
}
fprintf(f[0], "set terminal png medium\n");
fprintf(f[0], "set output \"cluster_plot.png\"\n");
fprintf(f[0], "set title \"FCM clustering\"\n");
fprintf(f[0], "set xlabel \"x-coordinate\"\n");
fprintf(f[0], "set ylabel \"y-coordinate\"\n");
fprintf(f[0], "set xrange [%lf : %lf]\n", low_high[0][0],
low_high[0][1]);
fprintf(f[0], "set yrange [%lf : %lf]\n", low_high[1][0],
low_high[1][1]);
fprintf(f[0],
"plot 'cluster.0' using 1:2 with points pt 7 ps 1 lc 1 notitle");
for (j = 1; j < num_clusters; j++) {
sprintf(fname, "cluster.%d", j);
fprintf(f[0],
",\\\n'%s' using 1:2 with points pt 7 ps 1 lc %d notitle",
fname, j + 1);
}
fprintf(f[0], "\n");
fclose(f[0]);
return 0;
}
void
print_data_points(char *fname) {
int i, j;
FILE *f;
if (fname == NULL)
f = stdout;
else if ((f = fopen(fname, "w")) == NULL) {
printf("Cannot create output file.\n");
exit(1);
}
fprintf(f, "Data points:\n");
for (i = 0; i < num_data_points; i++) {
printf("Data[%d]: ", i);
for (j = 0; j < num_dimensions; j++) {
printf("%.5lf ", data_point[i][j]);
}
printf("\n");
}
if (fname == NULL)
fclose(f);
```

```
}
void
print_membership_matrix(char *fname) {
int i, j;
FILE *f;
if (fname == NULL)
f = stdout;
else if ((f = fopen(fname, "w")) == NULL) {
printf("Cannot create output file.\n");
exit(1);
}
fprintf(f, "Membership matrix:\n");
for (i = 0; i < num_data_points; i++) {
fprintf(f, "Data[%d]: ", i);
for (j = 0; j < num_clusters; j++) {
fprintf(f, "%lf ", degree_of_memb[i][j]);
}
fprintf(f, "\n");
}
if (fname == NULL)
fclose(f);
}
int
main(int argc, char **argv) {
printf
("------------------------------------------------------------------
------\n");
if (argc != 2) {
printf("USAGE: fcm <input file>\n");
exit(1);
}
fcm(argv[1]);
printf("Number of data points: %d\n", num_data_points);
printf("Number of clusters: %d\n", num_clusters);
printf("Number of data-point dimensions: %d\n", num_dimensions);
printf("Accuracy margin: %lf\n", epsilon);
print_membership_matrix("membership.matrix");
gnuplot_membership_matrix();
printf
("------------------------------------------------------------------
------\n");
printf("The program run was successful...\n");
printf("Storing membership matrix in file 'membership.matrix'\n\n");
printf("If the points are on a plane (2 dimensions)\n");
printf("the gnuplot script was generated in file 'gnuplot.script',
and\n");
printf("the gnuplot data in files cluster.[0]... \n\n");
printf
("Process 'gnuplot.script' to generate graph:
'cluster_plot.png'\n\n");
printf
("NOTE: While generating the gnuplot data, for each of the data
points\n");
printf("the corresponding cluster is the one which has the
highest\n");
printf("degree-of-membership as found in 'membership.matrix'.\n");
```

```
printf
("------------------------------------------------------------------
------\n");
return 0;
}
```

8. Header Files

8.1. ECDH *(Source Code of "Code/src/ecdh.h")*

```
#ifndef ECDH_H
#define ECDH_H
#include "miracl.h"
#include "octet.h"
#define EAS 16 /* Symmetric Key size - 128 bits */
#define EGS 32 /* ECCSI Group Size - 256 bits */
#define EFS 32 /* ECCSI Field Size - 256 bits */
#define ECDH_OK 0
#define ECDH_DOMAIN_ERROR -1
#define ECDH_INVALID_PUBLIC_KEY -2
#define ECDH_ERROR -3
#define ECDH_INVALID -4
#define ECDH_DOMAIN_NOT_FOUND -5
#define ECDH_OUT_OF_MEMORY -6
#define ECDH_DIV_BY_ZERO -7
#define ECDH_BAD_ASSUMPTION -8
extern const mr_small ecrom[];
/* ECp domain parameters */
typedef struct
{
int nibbles;
char Q[EFS];
char A[EFS];
char B[EFS];
char R[EGS];
char Gx[EFS];
char Gy[EFS];
} ecp_domain;
#define HASH_BYTES 32
#if HASH_BYTES==20
#define HASHFUNC sha
#define SHS_INIT shs_init
#define SHS_PROCESS shs_process
#define SHS_HASH shs_hash
#define HASH_BLOCK 64
#endif
#if HASH_BYTES==32
#define HASHFUNC sha256
#define SHS_INIT shs256_init
#define SHS_PROCESS shs256_process
#define SHS_HASH shs256_hash
#define HASH_BLOCK 64
#endif
#if HASH_BYTES==48
#define HASHFUNC sha384
```

```
#define SHS_INIT shs384_init
#define SHS_PROCESS shs384_process
#define SHS_HASH shs384_hash
#define HASH_BLOCK 128
#endif
#if HASH_BYTES==64
#define HASHFUNC sha512
#define SHS_INIT shs512_init
#define SHS_PROCESS shs512_process
#define SHS_HASH shs512_hash
#define HASH_BLOCK 128
#endif
/* ECDH Auxiliary Functions */
extern void CREATE_CSPRNG(csprng *,octet *);
extern void KILL_CSPRNG(csprng *);
extern void HASH(octet *,octet *);
extern BOOL HMAC(octet *,octet *,int,octet *);
extern void KDF1(octet *,int,octet *);
extern void KDF2(octet *,octet *,int,octet *);
extern void PBKDF2(octet *,octet *,int,int,octet *);
extern void AES_CBC_IV0_ENCRYPT(octet *,octet *,octet *);
extern BOOL AES_CBC_IV0_DECRYPT(octet *,octet *,octet *);
/* ECDH primitives - support functions */
extern void ECP_DOMAIN_KILL(ecp_domain *);
extern int ECP_DOMAIN_INIT(ecp_domain *,const void *);
extern int ECP_KEY_PAIR_GENERATE(ecp_domain *,csprng *,octet *,octet
*);
extern int ECP_PUBLIC_KEY_VALIDATE(ecp_domain *,BOOL,octet *);
/* ECDH primitives */
extern int ECPSVDP_DH(ecp_domain *,octet *,octet *,octet *);
extern int ECPSVDP_DHC(ecp_domain *,octet *,octet *,BOOL,octet *);
/* ECIES functions */
extern void ECP_ECIES_ENCRYPT(ecp_domain *,octet *,octet *,csprng
*,octet *,octet *,int,octet *,octet *,octet *);
extern BOOL ECP_ECIES_DECRYPT(ecp_domain *,octet *,octet *,octet
*,octet *,octet *,octet *,octet *);
/* ECDSA functions */
extern int ECPSP_DSA(ecp_domain *,csprng *,octet *,octet *,octet
*,octet *);
extern int ECPVP_DSA(ecp_domain *,octet *,octet *,octet *,octet *);
#endif
```

8.2. OCTET *(Source Code of "Code/src/octet.h")*

```
#ifndef OCTET_H
#define OCTET_H
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
/* portable representation of a big positive number */
typedef struct
{
int len;
int max;
char *val;
} octet;
```

```
/* Octet string handlers */
extern void OCTET_OUTPUT(octet *);
extern void OCTET_OUTPUT_STRING(octet *);
extern void OCTET_CLEAR(octet *);
extern int OCTET_COMPARE(octet *,octet *);
extern void OCTET_JOIN_STRING(char *,octet *);
extern void OCTET_JOIN_BYTES(char *,int,octet *);
extern void OCTET_JOIN_BYTE(int,int,octet *);
extern void OCTET_JOIN_OCTET(octet *,octet *);
extern void OCTET_XOR(octet *,octet *);
extern void OCTET_EMPTY(octet *);
extern void OCTET_PAD(int,octet *);
extern void OCTET_TO_BASE64(octet *,char *);
extern void OCTET_FROM_BASE64(char *,octet *);
extern void OCTET_COPY(octet *,octet *);
extern void OCTET_XOR_BYTE(int,octet *);
extern void OCTET_CHOP(octet *,int,octet *);
extern void OCTET_JOIN_LONG(long,int,octet *);
#endif
```

8.3. MIRDEF *(Source Code of "Code/src/midef.h")*

```
#define MIRACL 32
#define MR_LITTLE_ENDIAN /* This may need to be changed */
#define mr_utype int
/* the underlying type is usually int *
* but see mrmuldv.any */
#define mr_unsign32 unsigned int
/* 32 bit unsigned type */
#define MR_IBITS 32 /* bits in int */
#define MR_LBITS 32 /* bits in long */
#define MR_FLASH 52
#define mr_dltype __int64 /* ... or long long for Unix/Linux */
#define mr_unsign64 unsigned __int64
#define MAXBASE ((mr_small)1<<(MIRACL-1))
```

9. Comparison Graphs Based on Nodes

9.1. Delay *(Source Code of "Code/Graph/Nodes/Delay.tcl")*

```
exec xgraph -P -m -bb -t NodesVsEndtoEndDelay -x Nodes -y Delay(s)
OKMSDT-Delay KMDT-Delay -geometry 800x400
```

9.2 Delratio *(Source Code of "Code/Graph/Nodes/DelRatio.tcl")*

```
exec xgraph -P -m -bb -t NodesVsDelratio -x Nodes -y Delratio
OKMSDT-Delratio KMDT-Delratio -geometry 800x400
```

9.3 Drop *(Source Code of "Code/Graph/Nodes/Drop.tcl")*

```
exec xgraph -P -m -bb -t NodesVsDrop -x Nodes -y Drop OKMSDT-Drop
KMDT-Drop -geometry 800x400
```

9.4 Overhead *(Source Code of "Code/Graph/Nodes/Overhead.tcl")*

```
exec xgraph -P -m -bb -t NodesVsOverhead -x Nodes -y Overhead(pkts)
OKMSDT-Overhead KMDT-Overhead -geometry 800x400
```

9.5 Throughput ***(Source Code of "Code/Graph/Nodes/throughput.tcl")***

```
exec xgraph -P -m -bb -t NodeVsThroughput -x Node -y Throughput
OKMSDT-Throughput KMDT-Throughput -geometry 800x400
```

10. Comparison Graphs Based on Rate

10.1. Delay ***(Source Code of "Code/Graph/Rate/Delay.tcl")***

```
exec xgraph -P -m -bb -t RateVsEndtoEndDelay -x Rate -y Delay(s)
OKMSDT-Delay KMDT-Delay -geometry 800x400
```

10.2 Delratio ***(Source Code of "Code/Graph/Rate/DelRatio.tcl")***

```
exec xgraph -P -m -bb -t RateVsDelratio -x Rate -y Delratio OKMSDT-
Delratio KMDT-Delratio -geometry 800x400
```

10.3 Drop ***(Source Code of "Code/Graph/Rate/Drop.tcl")***

```
exec xgraph -P -m -bb -t RateVsDrop -x Rate -y Drop OKMSDT-Drop
KMDT-Drop -geometry 800x400
```

10.4 Overhead ***(Source Code of "Code/Graph/Rate/Overhead.tcl")***

```
exec xgraph -P -m -bb -t RateVsOverhead -x Rate -y Overhead(pkts)
OKMSDT-Overhead KMDT-Overhead -geometry 800x400
```

10.5 Throughput ***(Source Code of "Code/Graph/Rate/throughput.tcl")***

```
exec xgraph -P -m -bb -t RateVsThroughput -x Rate -y Throughput
OKMSDT-Throughput KMDT-Throughput -geometry 800x400
```

- - - - - THE END - - - - -

APPENDIX

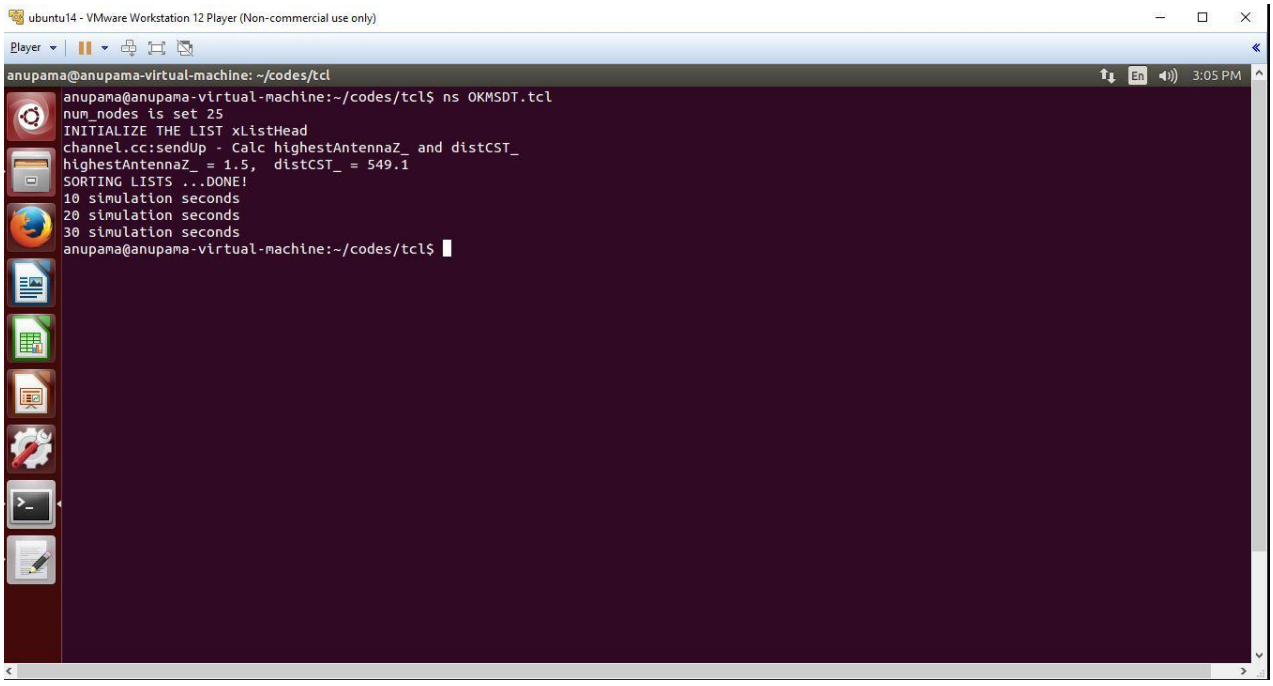
A1. NS2 Simulation

The proposed method OKMSDT (Optimal Key Management Technique for Secure Data Transmission in MANETs) is implemented using NS2 and the results are compared with KMDT (Key Management Technique for Data Transmission) in which optimization algorithm EBFO has not been applied..

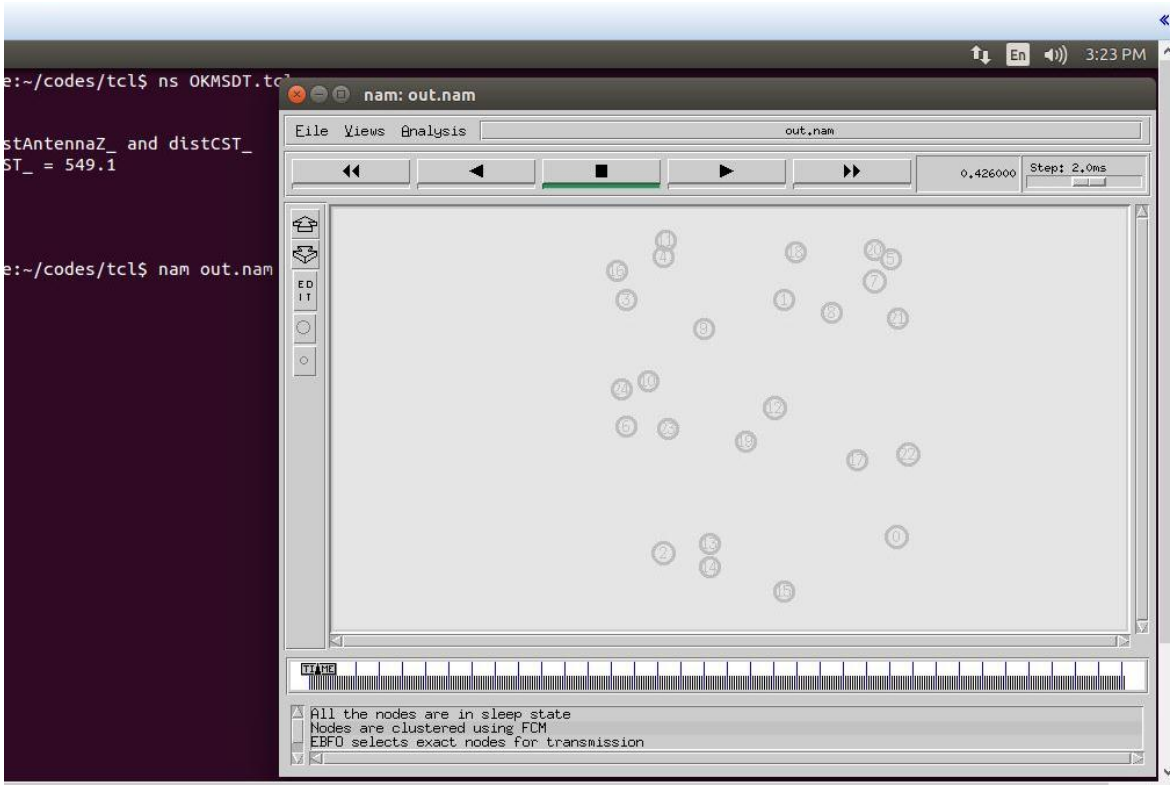
A1.1. Simulation Parameters

Parameter	Value Taken
Number of Nodes	25,50,75,100,125
Rate	50,100,150,200,250Kb
Simulation Time	0-40sec
Traffic Type	Constant Bit Rate
Terrain	1000x1000
Packet Size	512
Routing Protocol	AODV

A2. A screenshot of OKMSDT Simulation

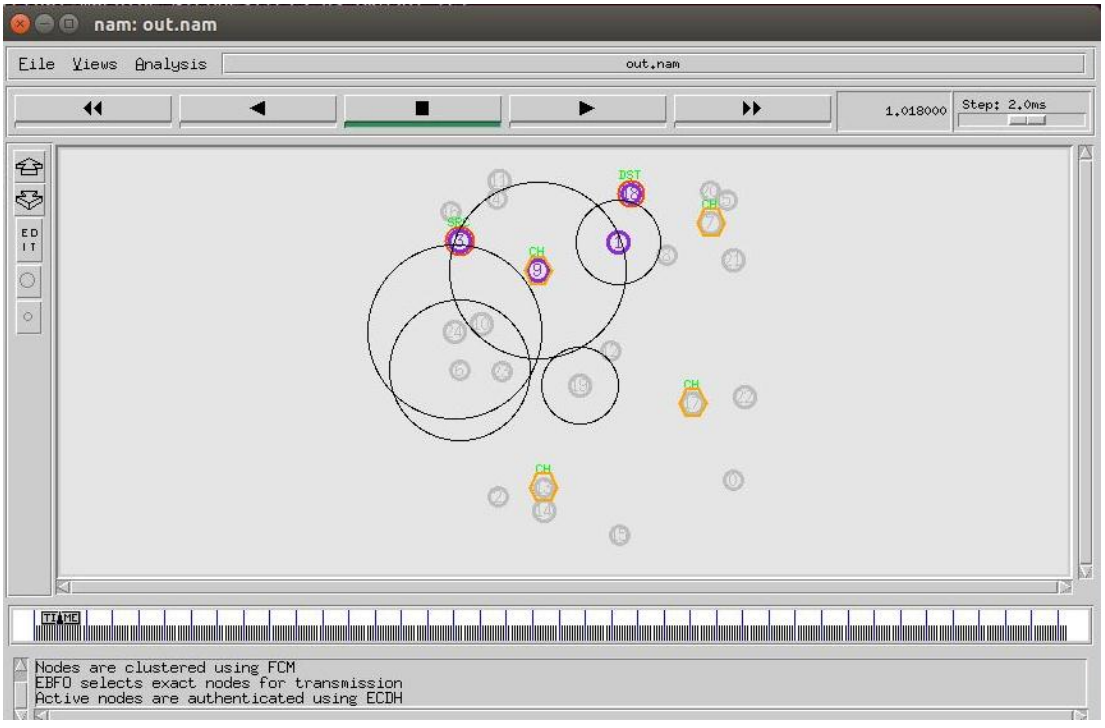


A3. A screenshot of the initial node set up in OKMSDT.



All 25 nodes (gray color) are inactive. (sleep state)

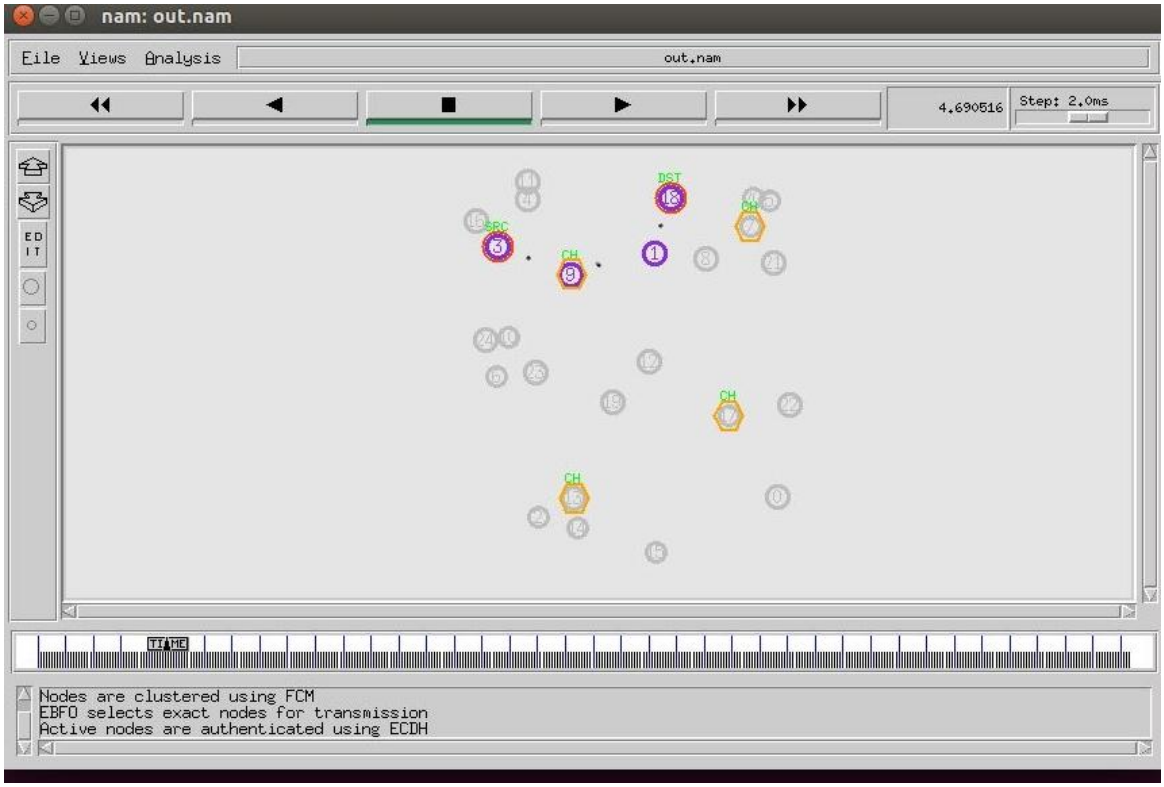
A4. A screenshot of the exact number of nodes for transmisssion in OKMSDT.



CH – cluster heads in yellow. No. of nodes that take part in transmission (purple color).

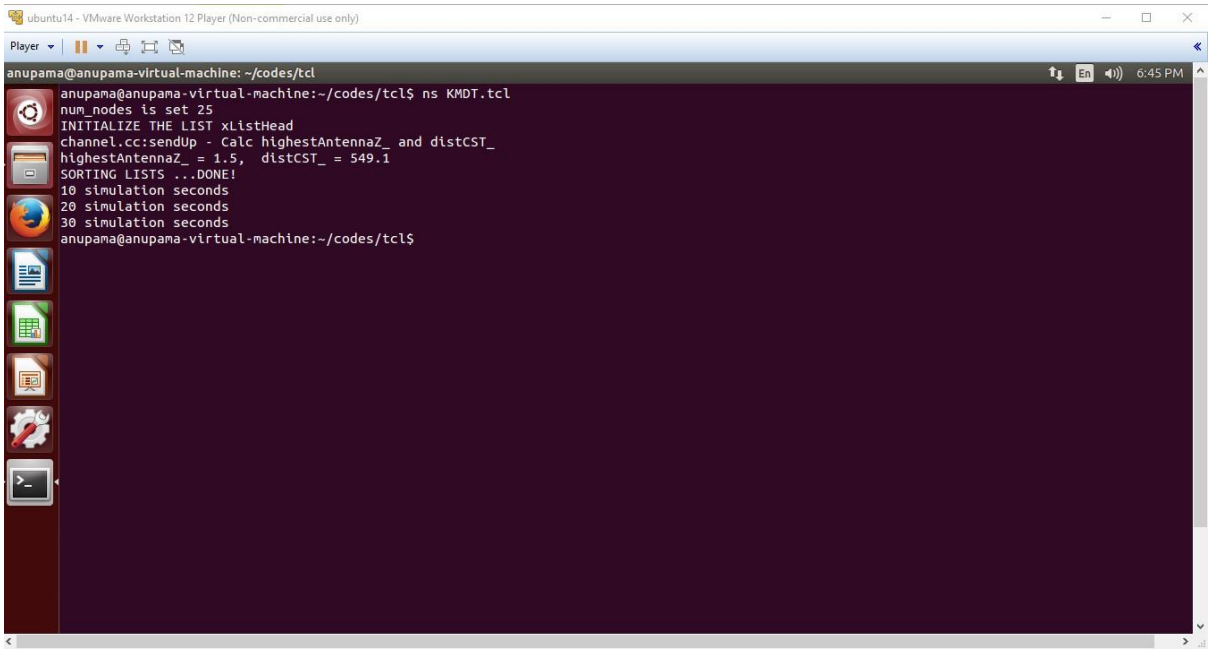
Source node - 3 , Destination node - 18

A5. A screenshot of data transmission from source node to destination node in OKMSDT.

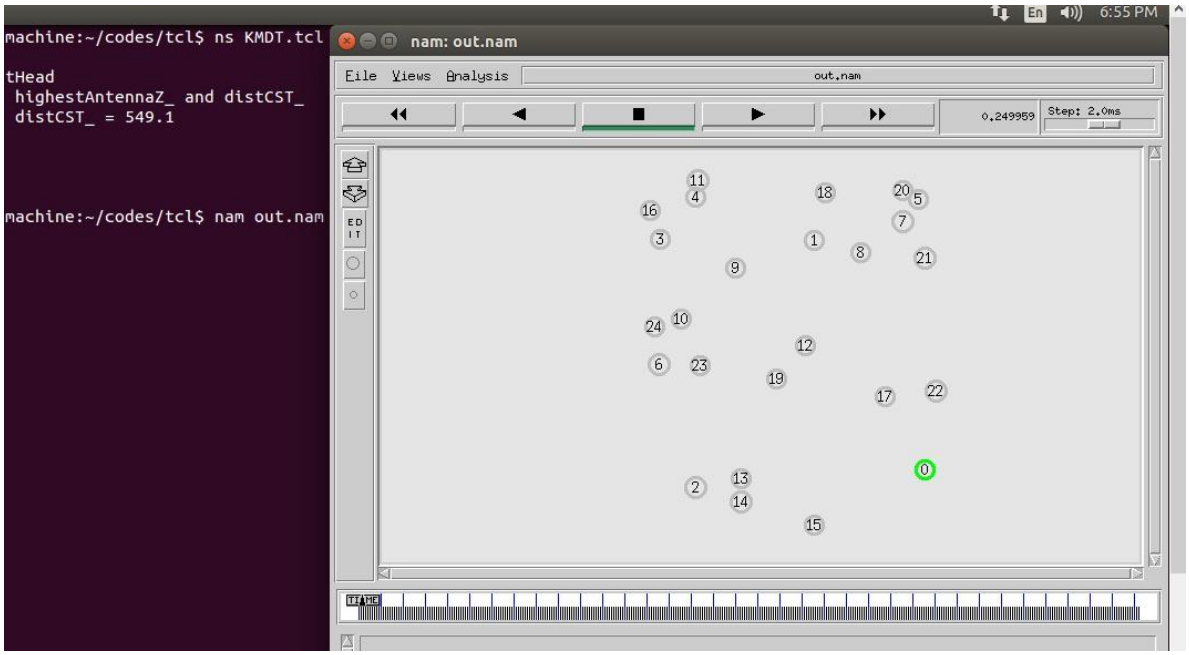


No packet drop during data transmission

A6. A screenshot of KMDT Simulation

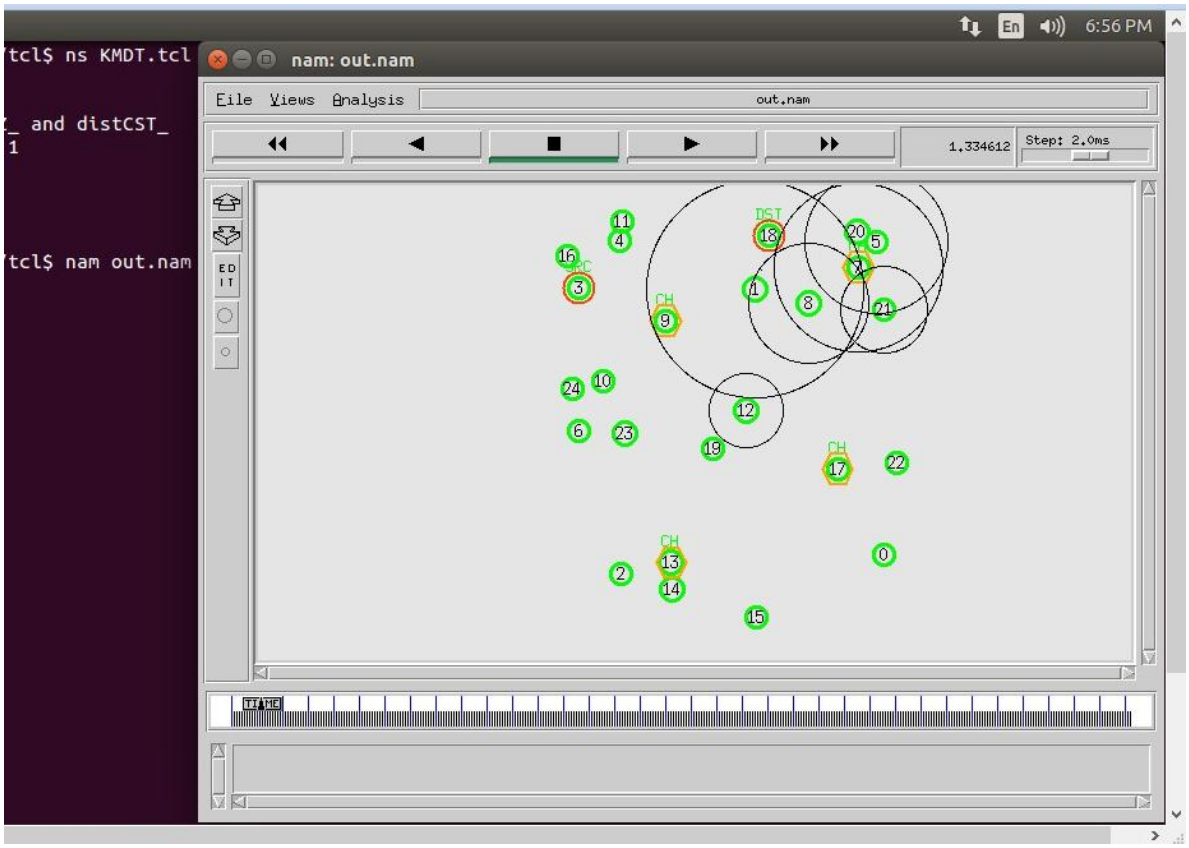


A7. A screenshot of the initial node set up in KMDT.



Gray nodes are inactive. (sleep state). Green node is active.

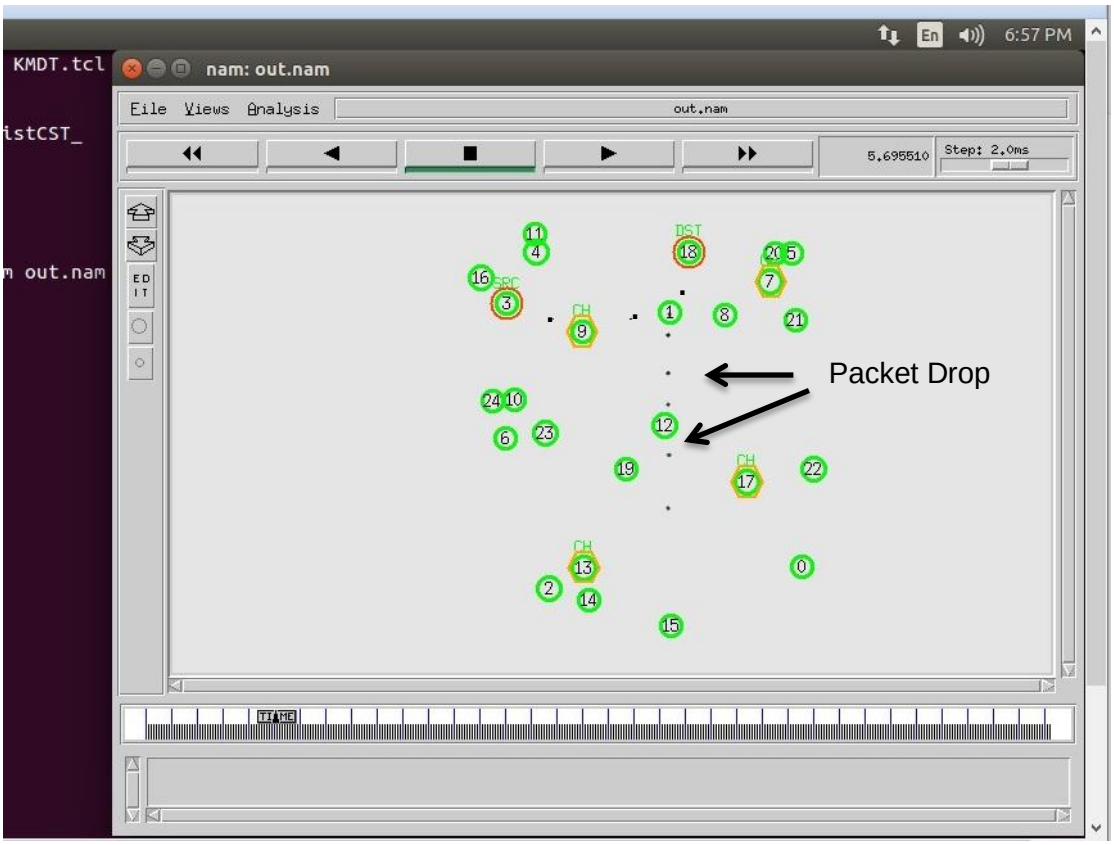
A8. A screenshot of the number of nodes for transmisssion in KMDT.



CH – cluster heads in yellow. All nodes take part in transmission (green color).

Source node - 3 , Destination node - 18

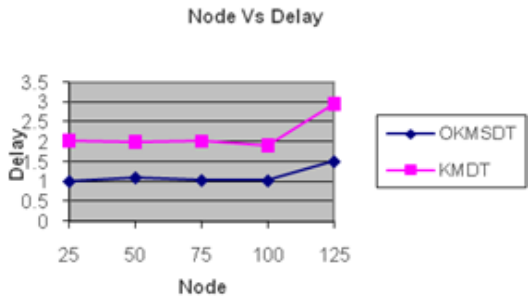
A9. A screenshot of data transmission from source node to destination node in KMDT.



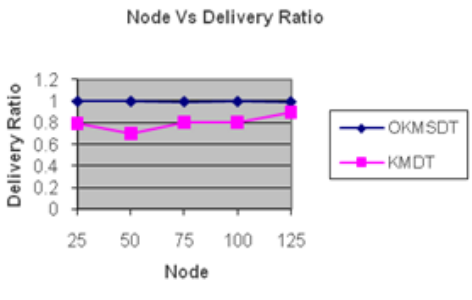
Packet drop is seen from node 1 during transmission of data

A10. Experimental Results and Comparison Graphs

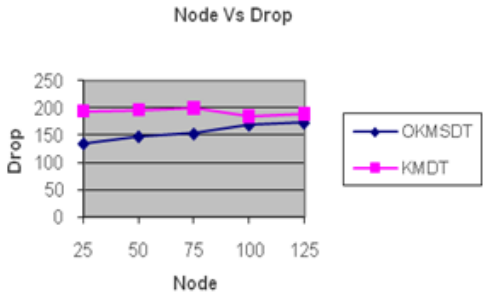
The experimental results and the performance of the proposed method are evaluated by varying the nodes as 25, 50, 75, 100 and 125 and nodal rate as 50, 100, 150, 200 and 250. Comparison analysis for end-to-end delay, packet delivery ratio, packet drop, throughput, and overhead are shown with the help of graphs. The results demonstrate that the proposed key management for secure communication in MANETs (OKMSDT) improves performance compared to the key management data transmission system (KMDT) in which the optimization algorithm EBFO has not been applied.



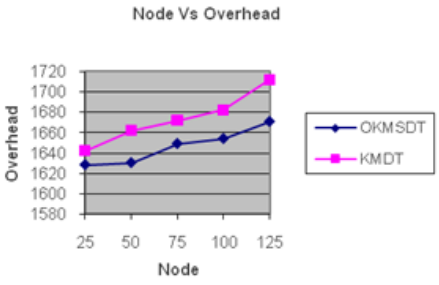
Comparison for Node vs. Delay



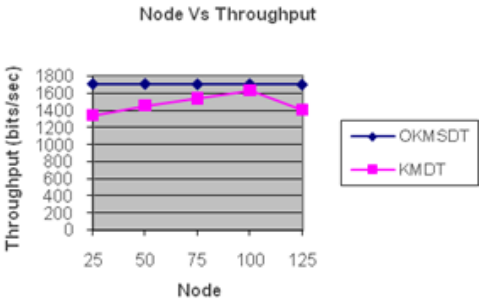
Comparison for Node vs. Delivery Ratio



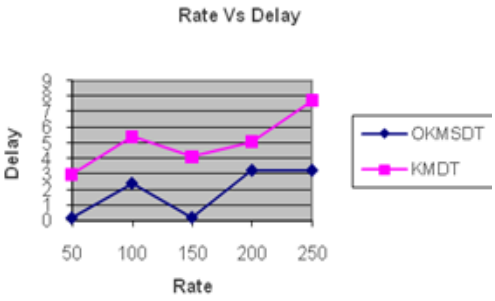
Comparison for Node vs. Drop



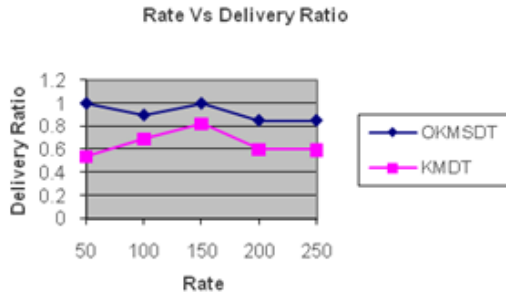
Comparison for Node vs. Overhead



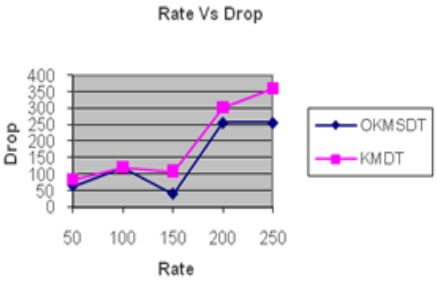
Comparison for Node vs. Throughput



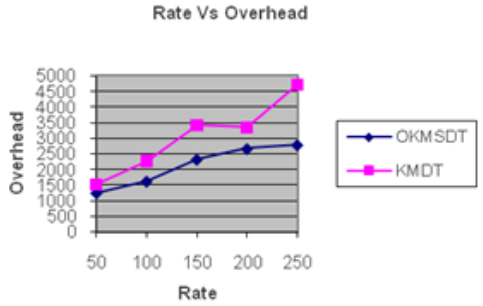
Comparison for Rate vs. Delay



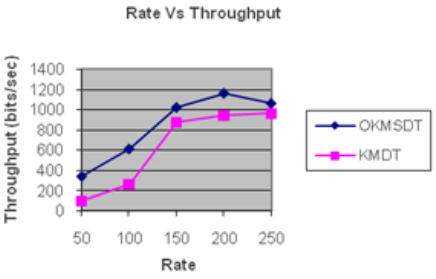
Comparison for Rate vs. Delivery ratio



Comparison for Rate vs. Drop



Comparison for Rate vs. Overhead



Comparison for Rate vs. Throughput

Coordinates Table for Comparison Graphs with respect to **Nodes**										
Nodes	OKMSDT					KMDT				
	Delay	Delivery Ratio	Drop	Overhead	Throughput	Delay	Delivery Ratio	Drop	Overhead	Throughput
25	1.008319	0.998248	140	1625	1709	2.036727	0.792918	194	1640	1339
50	1.02191	0.998248	149	1630	1709	2.104744	0.696486	198	1661	1444
75	1.034365	0.995911	151	1650	1705	2.009205	0.799618	210	1675	1500
100	1.024072	0.997079	160	1655	1707	1.907372	0.80359	189	1680	1601
125	1.50071	0.993575	170	1665	1701	2.957362	0.896745	194	1715	1400

Coordinates Table for Comparison Graphs with respect to **Rate**										
Nodal Rate	OKMSDT					KMDT				
	Delay	Delivery Ratio	Drop	Overhead	Throughput	Delay	Delivery Ratio	Drop	Overhead	Throughput
50	0.18765	0.994186	65	1344	342	3.984488	0.542373	84	1519	96
100	2.399411	0.893586	119	1616	613	5.381335	0.689655	120	2271	260
150	0.219827	0.994163	40	2250	1022	4.598412	0.820069	108	3425	874
200	3.228224	0.848175	257	2550	1162	5.075583	0.601057	302	3337	955
250	3.228224	0.848175	257	2621	1162	8.701532	0.597082	359	4717	1101