Handover Optimisation in 4G Systems

Iain Cuthbertson
201015895
Computer & Electronic Systems
University of Strathclyde

April 2014

Abstract

This is the paper's abstract ...

Acknowledgements

Thanks Mum!

Contents

Nomenclature

3G Third Generation

4G Fourth Generation

eNodeB Evolved Node B

LTE Long Term Evolution

UMTS Universal Mobile Telecommunications System

List of Figures

Chapter 1

Introduction

This is time for all good men to come to the aid of their party!

Chapter 2

LTE

2.1 Overview

Mobile communications is on to its fourth generation of network infrastructure with LTE (Long Term Evolution) (4G). This network infrastructure is an improvement upon Universal Mobile Telecommunications System (UMTS), which is a third generation network (3G).

Chapter 3

Machine Learning

3.1 Overview

3.2 Q-Learning

Chapter 4

Future Work

Chapter 5

Conclusion

We worked hard, and achieved very little.

Bibliography

[1] Joseph (Yossi) Gil. LaTeX 2_ε for graduate students. manuscript, Haifa, Israel, 2002.

[2] Nobody Jr. My article, 2006.

[3] R.R. Roy. *Handbook of Mobile Ad Hoc Networks for Mobility Models*. Springer, 2010.

Appendices

Appendix A

System Design and Evaluation

The contents...

```
#include "mobile.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "event_definitions.h"
#include "bstations.h"
#include "prop.h"
#include "q.h"
// #include "opt.h"

/* Constructor
 ****************************
 * Return Type: N/A
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Basic class constructor that create an instance of
    basestation
 * with all parameters set to zero.
 */
mobile::mobile(scheduler* gs) : event_handler(gs) {
   id = 0;
 x_co = 0;
 y_co = 0;
 connected = 1;
 h = 2.0;
 count = 0;
 wall = 0;
```

```
 minusX = 1;
 minusY = 1;
}
/* Constructor
 ****************************
 * Return Type: N/A
 ****************************
 * Parameters Passed in:
 * int idNum
 * int x
 * int y
 * double r
 * int con
 ****************************
 * Description: Class constructor that create an instance of
     basestation
 * with all parameters are passed in.
 */
mobile::mobile(scheduler* gs, int num, int x, int y, int con, double
    height) : event_handler(gs) {
    id = num;
    x_co = x;
    y_co = y;
    connected = con;
    h = height;
    count = 0;
    wall = 0;
    minusX = 1;
 minusY = 1;
 send_delay(new event(MOVE),0.0);
}
/* Destructor
 ****************************
 * Return Type: N/A
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Class destrcutor that removes an instance of this class
 * from the schedular and any messages still waiting to be passed.
 */
mobile::~mobile() {
```

```
 globalScheduler->remove_from(this);
 globalScheduler->remove_to(this);
}
/* Method
 ****************************
 * Return Type: N/A
 ****************************
 * Parameters Passed in:
 * const event* received
 ****************************
 * Description: Class hanlder that takes in a event and does the
 * required action depending on the event that was received.
 */
void mobile::handler(const event* received)
{
 switch(received->label) {
  case MOVE:
   moveRandom();
   print();
   count++;
   break;
  case STEP:
   moveMobile();
   send_now(new event(POLL));
   break;
  case POLL:
   double dist;
   propRequestPacket* sendPacket;
   for(int i=0; i<9;i++) {
    dist = sqrt(pow((bStations[i]->getX()-getX()),2.0) +
        pow((bStations[i]->getY()-getY()),2.0));
       sendPacket = new propRequestPacket(dist,h);
       send_now(new event(PROP,reinterpret_cast<payloadType<class
           T>*>(sendPacket),bStations[i]));
   }
   break;
  case PROP:
   propSendPacket* recPacket;
   recPacket = reinterpret_cast<propSendPacket*>
       (received->getAttachment());
      current_prop[recPacket->id] = recPacket->prop;
```

```
    //printf("Current: id:%d Rx:%f dB\nPrevious: id:%d Rx:%f
       dB\n",recPacket->id,current_prop[recPacket->id],
    //recPacket->id,previous_prop[recPacket->id]);
    checkProp(recPacket->id);
    delete recPacket;
    break;
  //   case TTTCHECK:
  //      reportPacket* tttPacket;
  // tttPacket = reinterpret_cast<reportPacket*>
      (received->getAttachment());
  //    if(current_prop[tttPacket->id] >= current_prop[connected]+hys)
      {
  //     //send measurement report
  //  reportPacket* sendPacket;
  //  sendPacket = new reportPacket(tttPacket->id);
  //  send_delay(new event(REPORT,reinterpret_cast<payloadType<class
      T>*>(sendPacket),bStations[connected]), HANDOVER_TIME);
  //  fprintf(stderr, "Sim Time: %f - Switch to basestation: %d\n",
      simTime,id);
  //  handingOver = true;
  // }
  // delete tttPacket;
  //    break;
 case PRINT:
  print();
  break;
 default:
  // program should not reach here
  break;
} // end switch statement
if(count > 100) {
 fprintf(stdout, "\nFinal Report\nHandovers: %d\nDropped:
    %d\nPing-Pong: %d\nHandover Failures: %d\n",
    handovers,drop,pingpongCount,drop);
 fprintf(stdout, "Final TTT: %f Final hys: %f\n", TTT,hys);
 // learning->print();
 globalScheduler->stop();
}
}

/* Method
```

```
 ****************************
 * Return Type: void
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Method that does a formetted print out of all the
 * parameters of the class.
 */
void mobile::print() {
    printf("Sim Time: %f - Mobile %d\nX Co-ordinate: %f\nY Co-ordinate:
        %f\nConnected To Basestation: %d\n", simTime,id, x_co, y_co,
        connected);
}
/* Method
 ****************************
 * Return Type: void
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Method that does a formetted print out of the
 * x and y co-ordinates of the mobile.
 */
void mobile::printCos() {
    printf("X Co-ordinate: %f\nY Co-ordinate: %f\n\n", x_co, y_co);
}
/* Method
 ****************************
 * Return Type: void
 ****************************
 * Parameters Passed in:
 * int newBasestation
 ****************************
 * Description: Method that changes the basestation id that this
 * instance of the class is "connected" to.
 */
void mobile::switchBasestation(int newBasestation) {
    connected = newBasestation;
}
/* Method
 ****************************
 * Return Type: void
```

```
 ****************************
 * Parameters Passed in:
 * int x
 * int y
 ****************************
 * Description: Method that "moves" this instance of the class
 * by changing it's x_co and y_co variables to the values
 * passed in.
 */
void mobile::moveMobile() {
 simTime += STEPTIME;
 if(duration>0) {
  if((x_co+(minusX*speed*STEPTIME*sin(angle*PI/180)))>1500) {
   minusX = -1;
  } else if((x_co+(minusX*speed*STEPTIME*sin(angle*PI/180)))<0) {
   minusX = -1;
  } else {
   x_co = x_co+(minusX*speed*STEPTIME*sin(angle*PI/180));
  }
  if(wall==0) {
   if((y_co+(minusY*speed*STEPTIME*cos(angle*PI/180)))>1500) {
    minusY = -1;
   } else if((y_co+(minusY*speed*STEPTIME*cos(angle*PI/180)))<0) {
    minusY = -1;
   } else {
    y_co = y_co+(minusY*speed*STEPTIME*cos(angle*PI/180));
   }
  }
  if(duration==0) {
   send_now(new event(MOVE));
  } else {
   duration -= STEPTIME;
   send_delay(new event(STEP),STEPTIME);
  }
  //fprintf(stderr, "x_co:%f y_co:%f\n", x_co,y_co);
 } else {
  send_delay(new event(MOVE),1.0);
 }
}
/* Method
 ****************************
```

```
 * Return Type: double
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Method that returns the value of the y co-ordinate.
 */
double mobile::getX() {
 return x_co;
}
/* Method
 ****************************
 * Return Type: double
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Method that returns the value of the y co-ordinate.
 */
double mobile::getY() {
 return y_co;
}
/* Method
 ****************************
 * Return Type: int
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Method that returns the id of the mobile that
 * the mobile is currently connected to.
 */
int mobile::getConnectedTo() {
 return connected;
}
/* Method
 ****************************
 * Return Type: double
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Method that returns the height of the mobile.
 */
double mobile::getHeight() {
```

```
 return h;
}
/* Method
 ****************************
 * Return Type: int
 ****************************
 * Parameters Passed in: N/A
 ****************************
 * Description: Method that returns a number that will define the
 * random movement the mobile will make.
 */
void mobile::moveRandom() {
 angle = rand()%360; //0 to 359 degrees
 speed = (rand()%4)+2; //1 to 4 m/s
 duration = (rand()%100)+50; //50 to 100s

 double deltaX = duration*speed*sin(angle*PI/180);
 double deltaY = duration*speed*cos(angle*PI/180);
 fprintf(stderr, "\nSim Time: %f - X_Co:%f Y_Co:%f deltaX:%f
     deltaY:%f\nspeed:%f duration:%f\n",
     simTime,x_co,y_co,deltaX,deltaY,speed,duration);

 minusX = 1;
 minusY = 1;

 moveMobile();
}

// void mobile::checkProp(int id) {
//  if(deadzone) {
//   if(current_prop[id]>THRESHOLD) {
//    fprintf(stderr, "Sim Time: %f - DEADZONE RECOVER\n",simTime);
//    deadzoneRecovers++;
//    reportPacket* reconPacket;
//    reconPacket = new reportPacket(id);
//    send_now(new event(REPORT,reinterpret_cast<payloadType<class
   T>*>(reconPacket),bStations[connected]));
//   }
//  } else if(!deadzone) {
//   if(id==connected) {
//    if(current_prop[id] < THRESHOLD) {
```

```
//     //called dropped!
//     int thresCount = 0;
//     for(int k=0; k<9; k++) {
//      if(current_prop[k] < THRESHOLD) {
//       thresCount++;
//      }
//     }
//     if(thresCount == 9) {
//      deadzone = true;
//     } else {
//      drop++;
//      double highest = -300.0;
//      int highestid = 0;
//      for(int j=0; j<9; j++) {
//       if(current_prop[j] > highest) {
//        highest = current_prop[j];
//        highestid = j;
//       }
//      }
//      reportPacket* sendPacket;
//      sendPacket = new reportPacket(highestid);
//      send_now(new event(REPORT,reinterpret_cast<payloadType<class
   T>*>(sendPacket),bStations[connected]));
//      fprintf(stderr, "Should switch to basestation: %d\n", id);
//      for(int i=0; i<9; i++) {
//       TTTtest[i] = TTT;
//       globalScheduler->remove_from(bStations[i]);
//      }
//      fprintf(stderr, "Sim Time: %f - DROPPED - Basestation:
   %d\n",simTime,connected);
//     }
//    }
//   } else if(!handingOver) {
//    if(current_prop[id] >= current_prop[connected]+hys) {
//     reportPacket* tttPacket;
//     tttPacket = new reportPacket(id);
//     send_delay(new
   event(TTTCHECK,reinterpret_cast<payloadType<class T>*>(tttPacket)),
   TTT);
//    }
//   }
```

```
//  }
// }

void mobile::checkProp(int id) {
 if(deadzone) {
  if(current_prop[id]>THRESHOLD) {
   // fprintf(stderr, "Sim Time: %f - DEADZONE RECOVER\n",simTime);
   deadzoneRecovers++;
   reportPacket* reconPacket;
   reconPacket = new reportPacket(id);
   send_now(new event(REPORT,reinterpret_cast<payloadType<class
       T>*>(reconPacket),bStations[connected]));
  }
 } else if(!deadzone) {
  if(id==connected) {
   if(current_prop[id] < THRESHOLD) {
    //called dropped!
    // TTT_weighting[TTTindex] -= 1;
    // hys_weighting[hysindex] -= 1;
    // learning->learn(); //call machine learning
    int thresCount = 0;
    if(handingOver) {
     handoverFailures++;
     for(int l=0; l<9; l++) {
      globalScheduler->remove_to(bStations[l]);
     }
     handingOver = false;
    }
    for(int k=0; k<9; k++) {
     if(current_prop[k] < THRESHOLD) {
      thresCount++;
     }
    }
    if(thresCount == 9) {
     deadzone = true;
    } else {
     drop++;
     rewardDrop++;
     if(function == 2) {//runnning policy
      send_now(new event(POLICY,q));
     }
```

```
    double highest = -300.0;
    int highestid = 0;
    for(int j=0; j<9; j++) {
     if(current_prop[j] > highest) {
      highest = current_prop[j];
      highestid = j;
     }
    }
    reportPacket* sendPacket;
    sendPacket = new reportPacket(highestid);
    send_now(new event(REPORT,reinterpret_cast<payloadType<class
        T>*>(sendPacket),bStations[connected]));
    // fprintf(stderr, "Should switch to basestation: %d\n", id);
    for(int i=0; i<9; i++) {
     TTTtest[i] = TTT;
     globalScheduler->remove_from(bStations[i]);
    }
    // fprintf(stderr, "Sim Time: %f - DROPPED - Basestation:
        %d\n",simTime,connected);
   }
  }
 } else if(!handingOver && id !=connected) {
  TTTtest[id] -= STEPTIME;
  if((TTTtest[id] <= 0) && (current_prop[id] >=
      current_prop[connected]+hys)) {
   if(TTTtest[id] <= 0) {
    //send measurement report
    reportPacket* sendPacket;
    sendPacket = new reportPacket(id);
    send_delay(new event(REPORT,reinterpret_cast<payloadType<class
        T>*>(sendPacket),bStations[connected]), HANDOVER_TIME);
    fprintf(stderr, "Sim Time: %f - Switch to basestation: %d\n",
        simTime,id);
    for(int i=0; i<9; i++) {
     TTTtest[i] = TTT;
    }
    handingOver = true;
   }
  }
 }
}
```

```
}

// void mobile::checkProp(int id) {
//  if(deadzone) {
//   if(current_prop[id]>THRESHOLD) {
//    // fprintf(stderr, "Sim Time: %f - DEADZONE RECOVER\n",simTime);
//    deadzoneRecovers++;
//    reportPacket* reconPacket;
//    reconPacket = new reportPacket(id);
//    send_now(new event(REPORT,reinterpret_cast<payloadType<class
   T>*>(reconPacket),bStations[connected]));
//   }
//  } else if(!deadzone) {
//   if(id==connected) {
//    if(current_prop[id] < THRESHOLD) {
//     //called dropped!
//     // TTT_weighting[TTTindex] -= 1;
//     // hys_weighting[hysindex] -= 1;
//     // learning->learn(); //call machine learning
//     int thresCount = 0;
//     if(handingOver) {
//      handoverFailures++;
//      for(int l=0; l<9; l++) {
//       globalScheduler->remove_to(bStations[l]);
//      }
//      handingOver = false;
//     }
//     for(int k=0; k<9; k++) {
//      if(current_prop[k] < THRESHOLD) {
//       thresCount++;
//      }
//     }
//     if(thresCount == 9) {
//      deadzone = true;
//     } else {
//      drop++;
//      rewardDrop++;
//      if(function == 2) {//runnning policy
//       send_now(new event(POLICY,q));
//      }
//      double highest = -300.0;
```

```
//      int highestid = 0;
//      for(int j=0; j<9; j++) {
//       if(current_prop[j] > highest) {
//        highest = current_prop[j];
//        highestid = j;
//       }
//      }
//      reportPacket* sendPacket;
//      sendPacket = new reportPacket(highestid);
//      send_now(new event(REPORT,reinterpret_cast<payloadType<class
   T>*>(sendPacket),bStations[connected]));
//      // fprintf(stderr, "Should switch to basestation: %d\n", id);
//      for(int i=0; i<9; i++) {
//       TTTtest[i] = TTT;
//       globalScheduler->remove_from(bStations[i]);
//      }
//      // fprintf(stderr, "Sim Time: %f - DROPPED - Basestation:
   %d\n",simTime,connected);
//     }
//    }
//   } else if(!handingOver && !deadzone) {
//    if(current_prop[id] >= current_prop[connected]+hys) {
//     TTTtest[id] -= STEPTIME;
//     if(TTTtest[id] <= 0) {
//      //send measurement report
//      reportPacket* sendPacket;
//      sendPacket = new reportPacket(id);
//      send_delay(new event(REPORT,reinterpret_cast<payloadType<class
   T>*>(sendPacket),bStations[connected]), HANDOVER_TIME);
//      // fprintf(stderr, "Sim Time: %f - Should switch to
   basestation: %d\n", simTime,id);
//      for(int i=0; i<9; i++) {
//       TTTtest[i] = TTT;
//      }
//      handingOver = true;
//     }
//    } else {
//     TTTtest[id] = TTT;
//    }
//   }
//  }
```

```
// }
```