บทที่ 4

การทดลองและการประเมินผล

ในการทดลองและประเมินผลนี้ เราจะประเมินการทำงานของโปรแกรมประยุกต์ที่เขียนด้วย DRN ซึ่งทำงานอยู่บนระบบที่เราพัฒนาขึ้นอีกทีหนึ่ง ในที่นี้จะขออธิบายถึงวิธีการทดลองและประเมินผลของเรา รวมทั้งพิจารณาพารามิเตอร์ในการปรับปรุงสมรรถนะของ DRN ว่ามีผลต่อสมรรถนะของโปรแกรมประยุกต์อย่างไร

4.1 วัตถุประสงค์ ตัววัด และ วิธีการ

เราได้พัฒนาโปรแกรมประยุกต์สำหรับการติดตามวัตถุโดยใช้ DRN โปรแกรมประยุกต์นี้ถูกประเมินผลบนเครือข่ายขนาด 20 โหนด แต่ละโหนดถูกจำลองโดยใช้ Smart Message Virtual Machine (SVM) ที่รันอยู่บนเครื่องคอมพิวเตอร์เดียวกัน หากแต่ว่าใช้พอร์ตต่างกัน (เนื่องจาก SMVM สามารถรันได้บนเครื่อง HP iPAQ [3] โปรแกรมของเราจึงสามารถทำงานได้บนเครื่อง iPAQ โดยไม่ต้องมีการแก้ไขใดใด)

วัตถุประสงค์ของเราในการศึกษาและประเมินผลนั้นมีอยู่ 2 ส่วนคือ ข้อแรก ทดสอบว่าโมเดลของ DRN สำหรับการโปรแกรมเชิงมหภาคเครือข่ายไร้สายของระบบฝังตัวเป็นโมเดลที่ใช้งานได้จริง ข้อสองคือ ทำความเข้าใจกับค่าอายุขัยของการผูกทรัพยากรว่ามีผลกระทบต่อโปรแกรมอย่างไร

เราเลือกตัววัดในการวิเคราะห์สมรรถนะของโปรแกรมเราดังนี้: จำนวนไบต์ที่โปรแกรมประยุกต์ส่งและค่าผิดพลาดเฉลี่ยของระยะห่าง จำนวนไบต์ที่โปรแกรมประยุกต์ส่งนั้นนับทั้งเครือข่าย ตัววัดนี้สามารถบ่งบอกถึงพลังงานที่ต้องสูญเสียไปได้อย่างหยาบๆ และยังสามารถอนุมานถึงอายุขัยโดยรวมของเครือข่ายอีกด้วย ค่าผิดพลาดเฉลี่ยของระยะห่างเป็นตัววัดระยะห่างระหว่างตำแหน่งจริงของวัตถุและตำแหน่งที่รายงานจากโปรแกรม ตัววัดนี้เป็นตัวบ่งบอกถึงความเที่ยงตรงของโปรแกรมติดตาม ตัววัดที่คล้ายกันนี้ถูกนำเสนอในงานอื่นเช่นกัน [23] เราศึกษาตัววัดดังกล่าวโดยใช้ค่าอายุขัยของการผูกต่างๆกัน เพื่อดูผลกระทบที่เกิดขึ้น

ในการทดลองของเรา เราจำลองพื้นที่ซึ่งติดตั้งเครือข่ายเซนเซอร์ไร้สายแบบหลายฮอบ อันประกอบไปด้วยโหนด 20 โหนด ที่ติดตั้งในตำแหน่งที่สุ่มมาในพื้นที่ขนาด 20m x 40m โหนดแต่ละตัวมีรัศมีการส่ง

10m ส่วนรัศมีในการรับรู้คือ 5m ระยะดังกล่าวทำให้โหนดสองโหนดที่ตรวจจับวัตถุเดียวกันได้สามารถสื่อสารตรงถึงกันได้ในลิงค์เดียว รัศมีการส่งยังเป็นตัวกำหนดเพื่อนบ้านของโหนดแต่ละตัวในการจำลองของเรา

วัตถุมีการเคลื่อนที่ที่ความเร็ว 0.25m/s โดยมีการเคลื่อนที่ตามเข็มนาฬิกาบนขอบของสี่เหลี่ยมขนาด 10m x 30m ซึ่งอยู่ตรงกลางพื้นที่ตรวจจับของเรา การเคลื่อนที่ตามเข็มนาฬิกานี้ทำให้โหนดในบริเวณต่างกันมีการตรวจพบและติดตามวัตถุโดยทั่วถึงกัน โปรแกรมจะประมาณตำแหน่งของวัตถุ 25 ครั้งตลอดการทดลอง หรือ หนึ่งครั้งทุกสี่วินาที นอกจากนี้การทดลองของเราได้กำจัดผลกระทบที่อาจเกิดจากลำดับชั้น MAC ออกไป โดยกำหนดให้ไม่มีการชนกันหรือการสูญหายของ packet ในการทดลอง

4.2 โปรแกรมติดตามวัตถุ

รูปที่ 1 แสดงรหัสเทียม DRN อย่างง่ายสำหรับโปรแกรมประยุกต์ติดตามวัตถุของเรา โดยหลักๆ แล้ว โปรแกรมจะติดตามวัตถุโดยการอ่านค่าตำแหน่งของอุปกรณ์หรือโหนดที่ตรวจพบวัตถุภายในบริเวณที่สนใจ ค่าเฉลี่ยตำแหน่งของอุปกรณ์ดังกล่าวจะถูกใช้เป็นค่าประมาณตำแหน่งของวัตถุ ในตอนแรก จะไม่มีค่าประมาณตำแหน่งของวัตถุ โปรแกรมจะต้องหาวัตถุให้ทั่วพื้นที่การทดลอง เมื่อพบวัตถุแล้ว เราสามารถใช้ตำแหน่งของวัตถุมากำหนดบริเวณที่สนใจในการค้นหาครั้งต่อไปได้โดยไม่ต้องค้นหาทั้งพื้นที่อีกต่อไป ในที่นี้กำหนดให้บริเวณที่สนใจสำหรับการค้นหาครั้งต่อไปเป็นพื้นที่ในระยะ 10 m รอบตำแหน่งที่ประมาณได้ครั้งล่าสุด แนวทางนี้มีข้อดีในแง่ของการจำกัดพื้นที่ในการค้นหา ยังผลให้มีประสิทธิภาพในแง่พลังงานดีขึ้น โดยเฉพาะอย่างยิ่งเมื่อมีการใช้การหาเส้นทางเชิงภูมิศาสตร์ในระบบ หลังจากนั้น หากเราไม่สามารถหาวัตถุในพื้นที่วงกลมพลวัตนี้ได้ บริเวณที่สนใจจะถูกเซตใหม่ให้เป็นพื้นที่ทั้งหมด

```
1: Space sp=UNIVERSE;
2: Resource R1=< (within(Sp)==TRUE) & (motion>0) >;
3: Location AverageLoc;
4:
5: for (int i=1; i<= 25; i++) {
6: AverageLoc = average(R1->Location);
```

```
7: if (AverageLoc != NULL) {
8: System.out.println("Average("+i+")="+AverageLoc);
9: sp.updateRegion(AverageLoc, 10);
10: } else {
11: System.out.println("Average("+i+")=NOT FOUND");
12: sp = UNIVERSE;
13: }
14: sleep(4000);
15: }
```

รูปที่ 1. รหัสเทียมสำหรับโปรแกรมประยุกต์ติดตามวัตถุ

ส่วนโปรแกรมภาษาจาวาสำหรับติดตามวัตถุจะแสดงไว้ในรูปที่ 2 ซึ่งมีอัลกอริธึ่มการทำงานเหมือนรหัสเทียมในรูปที่ 1 จะเห็นได้ว่ามีความเป็นไปได้ที่เราจะแปลงรหัสเทียมของ DRN เป็นโปรแกรมในภาษาจาวาได้โดยไม่ยากนัก สำหรับโปรแกรมในภาษาจาวานี้ คลาส TrackingApp เป็นเพียงคลาสที่เพิ่มขยายจาก SmWrapper ซึ่งเป็นคลาสที่ซ่อนรายละเอียดเกี่ยวกับ SM จากโปรแกรมเมอร์ เพื่อที่จะทำให้เข้ากันได้กับ Syntax ของภาษาจาวา เราได้สร้างส่วนการประกาศเงื่อนไขของทรัพยากรเป็นคลาสชื่อว่า TrackingExpression ดังแสดงไว้ในรูปที่ 3 ส่วนการสร้างคลาสเงื่อนไขทรัพยากรแบบอัตโนมัติจากรหัสเทียมอาจทำได้ แต่ไม่อยู่ในขอบเขตของโครงงานนี้ คลาส Expression แต่ละคลาสจะมี evaluate () method ที่จำเป็นต้องรันบนอุปกรณ์เพื่อดูว่าอุปกรณ์ตรงตามเงื่อนไขของ Expression หรือไม่

```
1: public class TrackingApp extends SmWrapper{
2:
3: private final static int timeout = 24000; // Binding lifetime
4: private Space sp;
5: private TrackingExpression tExp;
6: private Resource resource;
7: private LocationAverage agg;
```

```
8:
9: public TrackingApp(){
10:   super("TrackingApp");
.1: }
.2:
.3: public void run() {
.4:   try {
.5:         sp = new Space(null,-1); // sp = UNIVERSE
.6:         tExp = new TrackingExpression(sp, "motion");
.7:         resource = new Resource(tExp, timeout);
.8:         agg = new LocationAverage();
.9:         for (int i=1; i<= 25; i++) {
20:               agg = (LocationAverage)resource.access(agg, 4000);
21:               System.out.println("agg = "+agg);
22:               Location average = (Location)agg.evaluator();
23:               if (average != null) {
24:                     System.out.println("Average("+i+") = "+average);
25:                     sp.updateRegion(average, 10);
26:               } else {
27:                     System.out.println("Average("+i+") = NOT FOUND");
28:                     sp.updateRegion(null, -1); // sp = UNIVERSE
29:               }
30:               sleep(4000);
31:         }
32: } catch(Exception e) {}
33: }
```

```
34:
35: public static void main(String []args) {
36:   TrackingApp trackingApp = new TrackingApp();
37:   String []types;
38:   types = new String[3];
39:   types[0] = "TrackingApp";
40:   types[1] = "TrackingExpression";
41:   types[2] = "LocationAverage";
42:   trackingApp.initSM(types, trackingApp);
43:   trackingApp.run();
44: }
45: }
```

รูปที่ 2 โปรแกรมภาษาจาวาแบบ DRN สำหรับติดตามวัตถุ

ในส่วนโปรแกรมประยุกต์ติดตามวัตถุนี้ ทรัพยากรจะถูกเข้าถึงแบบขนาน ซึ่งเปิดโอกาสให้มีการประมวลผลในเครือข่ายได้ (อาทิเช่น การรวมข้อมูล) อันจะนำไปสู่การลดการใช้พลังงานโดยรวมของระบบ [10, 11, 13, 14, 17, 19] ตามปกติแล้ว ในระบบอื่นๆ โปรแกรมสำหรับการรวมข้อมูลในเครือข่ายไม่สามารถปรับแต่งหรือติดตั้งแบบพลวัตในขณะทำงาน หลังจากเริ่มติดตั้งใช้งานเครือข่ายไปแล้วได้ [19] ในบางระบบ API อาจไม่สนับสนุนให้เขียนหรือแก้โค้ดการรวมข้อมูลในเครือข่ายขึ้นมาใหม่ ซึ่งต่างจากจากงานของเรา DRN ได้สร้าง Class Aggregation ไว้ให้ ที่สามารถเพิ่มขยายให้เป็นการรวมข้อมูลแบบใหม่ได้ตามที่ต้องการ อีกทั้งยังสามารถติดตั้งได้แบบพลวัตในขณะทำงาน

```
1: public class TrackingExpression extends Expression{
2:
3: private Space sp_;
4: private String moTag_;
5:
```

```
6:public TrackingExpression(Space sp, String moTag) throws
7:                          BadSMApiUsageException {
8:    sp_=sp;
9:    moTag_=moTag;
10: }
11:
12: public boolean evaluate() {
13:   try{
14:         Integer moInt = (Integer)TagSpace.readTag(moTag_);
15:         GPSData gps = (GPSData)TagSpace.readTag("gps");
16:         if (!sp_.outside(new Location(gps.latitude, gps.longitude))
17:               && (moInt.intValue()>0) ) {
18:               return true;
19:         }
20:   }catch(Exception e) {}
21:   return false;
22: }
23: }
```

รูปที่ 3 Class TrackingExpression สำหรับโหนดที่ตรงตามเงื่อนไข

โดยทั่วไปแล้ว เทคนิคการรวมข้อมูลจะถูกสร้างโดยใช้ฟังก์ชัน 3 ฟังก์ชันได้แก่ initializer i(), merger m(), และ evaluator e() โดยที่ initialize i() จะระบุว่าจะทำการสร้างบันทึกสถานะข้อมูลสำหรับค่าของตัวรับรู้ค่าหนึ่งได้อย่างไร DRN จะเรียกฟังก์ชันนี้บนอุปกรณ์ที่มีคุณสมบัติตรงตามเงื่อนไขที่ประกาศไว้ บันทึกสถานะข้อมูลนี้จะถูกส่งกลับไปยังโหนดของผู้ใช้ ระหว่างการเดินทาง บันทึกดังกล่าวอาจไปพบเจอกับบันทึกอื่นจากเซตของโหนดที่ตรงตามเงื่อนไขเดียวกัน ในกรณีเช่นนี้ DRN จะเรียกฟังก์ชัน merger m() เพื่อรวมข้อมูลเหล่านี้ให้เป็นบันทึกเดียว เมื่อบันทึกได้มาถึงโหนดของผู้ใช้ DRN จะทำการเรียกฟังก์ชัน evaluator e() เพื่อคำนวณค่าที่แท้จริงของข้อมูลที่รวมได้

ในโปรแกรมประยุกต์ของเรา เราได้แสดงการสร้างเทคนิครวมข้อมูลใหม่ชื่อว่า LocationAverage (ในรูปที่ 4) โดยสามารถทำได้อย่างง่ายดายเพียงแค่การเพิ่มขยาย Class Aggregation และ Overload ฟังก์ชันทั้งสามที่กล่าวไปแล้วข้างต้น เราใช้ <sum_x, count_x, sum_y, count_y> แทนสถานะของข้อมูล สมมุติว่าอุปกรณ์ที่ตรงตามเงื่อนไขอยู่ที่ตำแหน่ง (x1, y1) ตัว initializer จะทำการสร้างบันทึกสถานะข้อมูล เป็น <x1, 1, y1, 1> ส่วน merger จะรวมสถานะ <x1, cx1, y1, cy1> และ <x2, cx2, y2, cy2> เป็นสถานะ เดียวคือ <x1+x2, cx1+cx2, y1+y2, cy1+cy2> สำหรับ evaluator จะประเมินผลลัพธ์ที่ให้ค่าเป็น <sum_x/count_x, sum_y/count_y> ซึ่งเป็นค่าตำแหน่งเฉลี่ย

```
1: public class LocationAverage extends Aggregate{
2:
3: private GPSData gps;
4: double sum_x, sum_y;
5: int count_x, count_y;
6:
7: public void initializer() {
8:    try {
9:          gps = (GPSData)TagSpace.readTag("gps");
10:         sum_x = gps.latitude;
11:         sum_y = gps.longitude;
12:         count_x = 1;
13:         count_y = 1;
14:   } catch (Exception e) {}
15: }
16:
17: public void merger(Aggregate agg) {
18:   sum_x = sum_x+agg.sum_x;
```

```
19:   sum_y = sum_y+agg.sum_y;
20:   count_x = count_x+agg.count_x;
21:   count_y = count_y+agg.count_y;
22: }
23:
24: public Object evaluator() {
25:   try {
26:         return new Location(sum_x/count_x, sum_y/count_y);
27:   } catch (Exception e) {
28:         return null;
29:   }
30: }
31: }
```

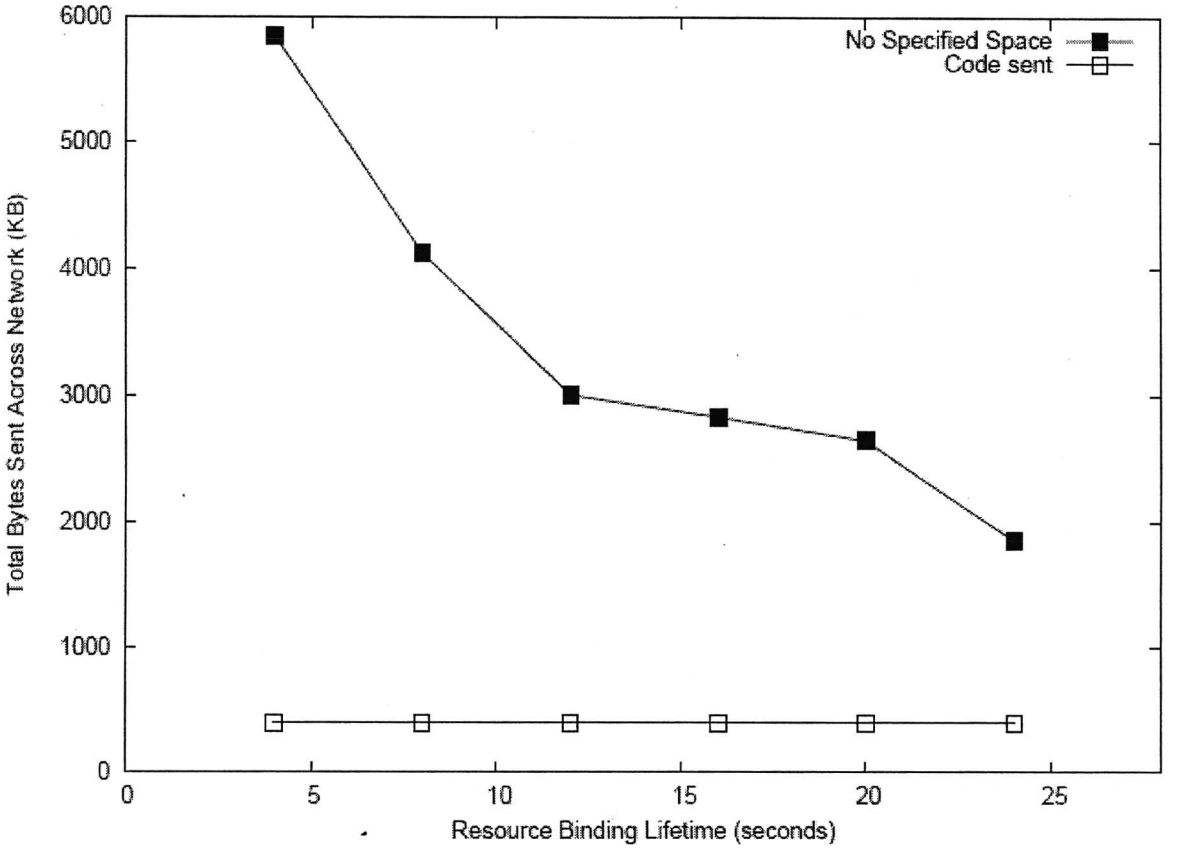
รูปที่ 4 แสดง Class LocationAverage สำหรับการประมวลผลในเครือข่าย

4.3 การปรับแต่ง

ในโมเดลของเรา หากใช้ความหมายอย่างเข้มงวด การเข้าถึงทรัพยากรจะต้องกระทำการบนโหนดที่ตรงตามเงื่อนไขเชิงประกาศในขณะที่เข้าถึงเท่านั้น การเปลี่ยนแปลงใดใดในเซตของโหนดที่ตรงตามเงื่อนไขไม่จำเป็นต้องให้โปรแกรมเมอร์มาเสียเวลาเขียนโปรแกรมรองรับไว้เอง ดังนั้น DRN จะต้องทำการผูกทรัพยากรได้ใหม่อย่างพลวัตและอัตโนมัติ ความหมายที่เข้มงวดนี้อาจนำไปสู่โอเวอร์เฮดและการใช้พลังงานอย่างมากเพื่อให้มั่นใจได้ว่าการผูกทรัพยากรนั้นถูกต้องอยู่เสมอ จึงไม่น่าประหลาดใจนักที่เรานำเสนอวิธีที่สามารถใช้ในการปรับแต่งระดับความเข้มงวดดังกล่าวเพื่อแลกกับการประหยัดพลังงานที่เพิ่มขึ้น หนึ่งในตัวปรับแต่งของเราคือ อายุขัยของการผูกทรัพยากร ยกตัวอย่างเช่น การใช้อายุขัยของการผูกทรัพยากรค่า t ถึงแม้ว่าจะลดความเข้มงวดของโมเดลลงแต่จะอนุญาตให้เข้าถึงทรัพยากรที่ถูกผูกไว้ภายใน t วินาทีที่ผ่านมา

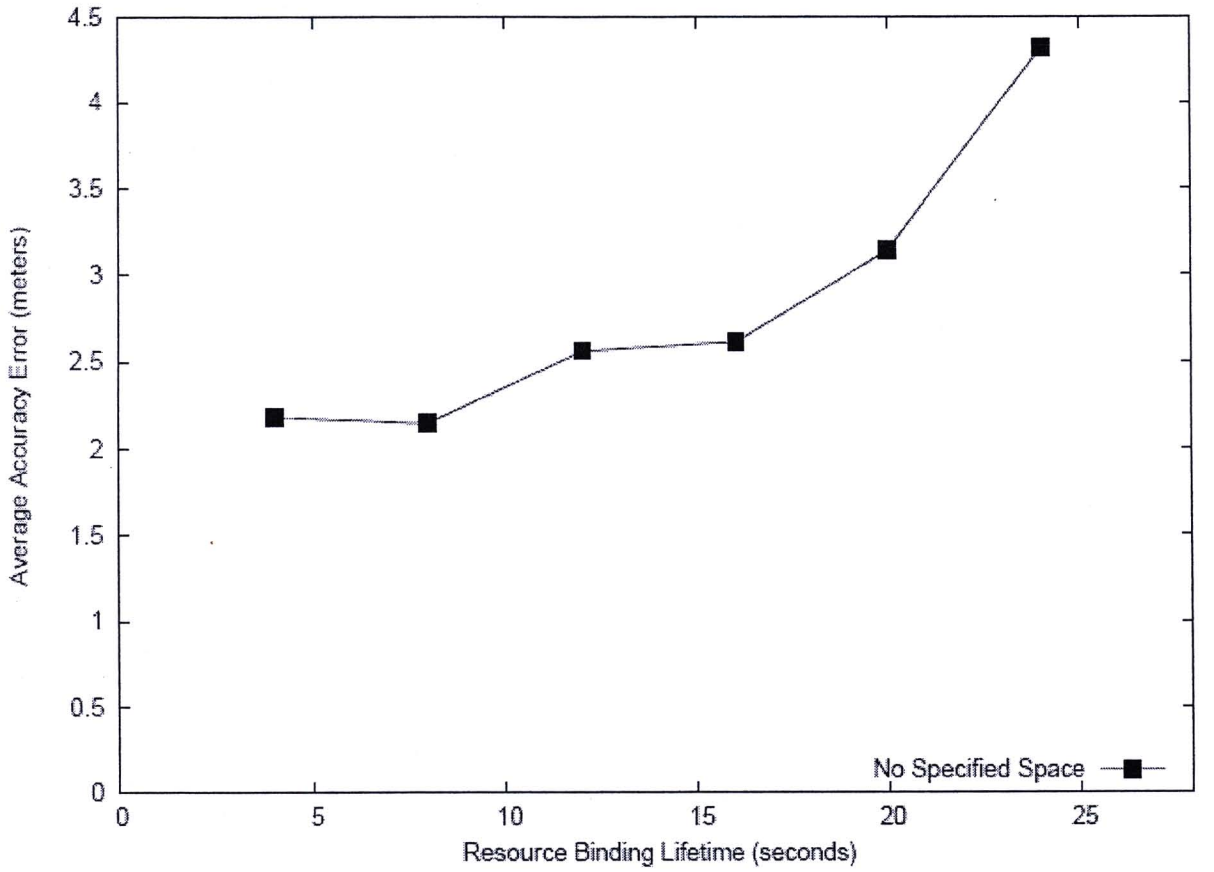
ในการทดลองนี้ เราศึกษาผลกระทบของอายุขัยการผูก ต่อการใช้พลังงานและความแม่นยำของการติดตามวัตถุของโปรแกรมประยุกต์เวอร์ชันที่ยังไม่ได้ปรับปรุงประสิทธิภาพ กล่าวคือ บรรทัดที่ 25 ในรูปที่ 2 นั้นถูกลบออกไป ดังนั้นการค้นหาวัตถุจะทำโดยทั่วเครือข่าย นอกจากนี้ เรายังมุ่งหมายที่จะแสดงให้เห็นว่าถึงแม้ว่า เงื่อนไขเชิงประกาศและตัวแปรที่เกี่ยวข้องไม่ได้ถูกแก้ไขเลย ทรัพยากรยังคงสามารถถูกผูกใหม่ได้อย่างเหมาะสมและเป็นพลวัต

รูปที่ 5 แสดงจำนวนไบต์ที่ส่งเมื่อเทียบว่าอายุขัยการผูกทรัพยากร ผลลัพธ์เป็นไปดังคาดคือ จำนวนไบต์ที่ใช้ลดลง เมื่อเราเพิ่มอายุขัยการผูกทรัพยากรขึ้น เพราะเป็นการลดจำนวนการค้นหาทรัพยากรที่ตรงตามเงื่อนไข ผลลัพธ์ชี้ให้เห็นว่ามันเป็นไปได้ที่จะประหยัดพลังงานได้อย่างมากโดยที่ไม่ทำให้ความแม่นยำลดลงอย่างมีนัยสำคัญ กล่าวคือ เราสามารถลดจำนวนไบต์ที่ต้องส่งลงได้ถึง 51.5% โดยที่ความแม่นยำลดลงเพียงเล็กน้อยหากเราเพิ่มอายุขัยการผูกทรัพยากรจาก 4 เป็น 16 วินาที การประหยัดนี้จะมีค่ามากขึ้นอีกเมื่อ โปรแกรมได้ถูกแคช หรือถูกติดตั้งในเครือข่ายเรียบร้อยแล้ว หากเราแยกจำนวนไบต์ที่ต้องใช้ในส่งโปรแกรมไปติดตั้งในเครือข่ายออกไป การประหยัดของเราจะเพิ่มขึ้นเป็น 55.2%



รูปที่ 5 แสดงจำนวนไบต์ที่โปรแกรมประยุกต์ต้องส่ง

ความผิดพลาดในการติดตามเฉลี่ยไม่ได้เพิ่มขึ้นอย่างมีนัยสำคัญ จนกว่าอายุขัยการผูกจะมีค่ามากกว่า 16 วินาที (รูป 6) ผลลัพธ์นี้ไม่น่าแปลกใจ ถ้าวัตถุเคลื่อนที่ด้วยความเร็ว 0.25 m/s ออกจากตัวรับรู้ที่ผูกไว้ มันจะใช้เวลาอย่างมากที่สุด 20 วินาที ที่จะออกจากรัศมีการรับรู้ของตัวรับรู้ ในทางกลับกัน ถ้าวัตถุเคลื่อนที่เข้าหาตัวรับรู้โดยไม่เปลี่ยนทิศทาง มันจะใช้เวลาอย่างมากที่สุด 40 วินาทีในการออกจากรัศมีการรับรู้ สำหรับการเคลื่อนที่ในการทดลองนี้ หากต้องการความแม่นยำที่พอใช้ได้ เราไม่จำเป็นต้องค้นหาทรัพยากรใหม่ภายใน 20 วินาที อย่างไรก็ตาม หลังจาก 20 วินาที ความแม่นยำจะเริ่มแย่ลงอย่างมีนัยสำคัญ ถ้าเราไม่ค้นหาทรัพยากรใหม่หลังจาก 40 วินาที เราจะไม่สามารถติดตามวัตถุได้



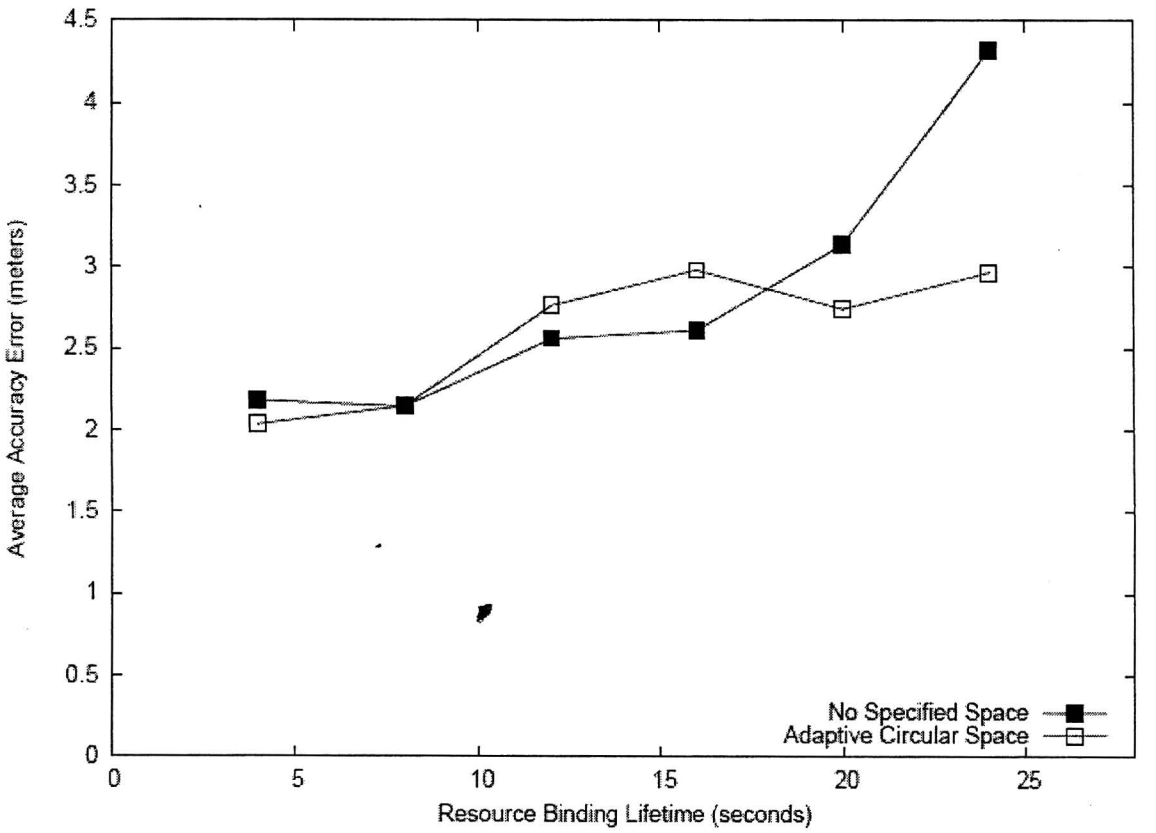
รูปที่ 6 แสดงระยะผิดพลาดเฉลี่ย

ความแม่นยำในการติดตามขึ้นอยู่กับหลายปัจจัย เช่น เทคนิคในการประมาณตำแหน่ง ความหนาแน่นของเครือข่าย และ รัศมีการรับรู้ ความผิดพลาดในการประมาณระดับ 2-3 m ในการทดลองนี้นับว่าใช้ได้ สำหรับวิธีการประมาณตำแหน่งที่เรียบง่ายเช่นนี้ ในเครือข่ายที่มีความหนาแน่นต่ำ และ มีรัศมีการรับรู้เพียง 5 m

4.4 การเพิ่มประสิทธิภาพโดยการระบุบริเวณ

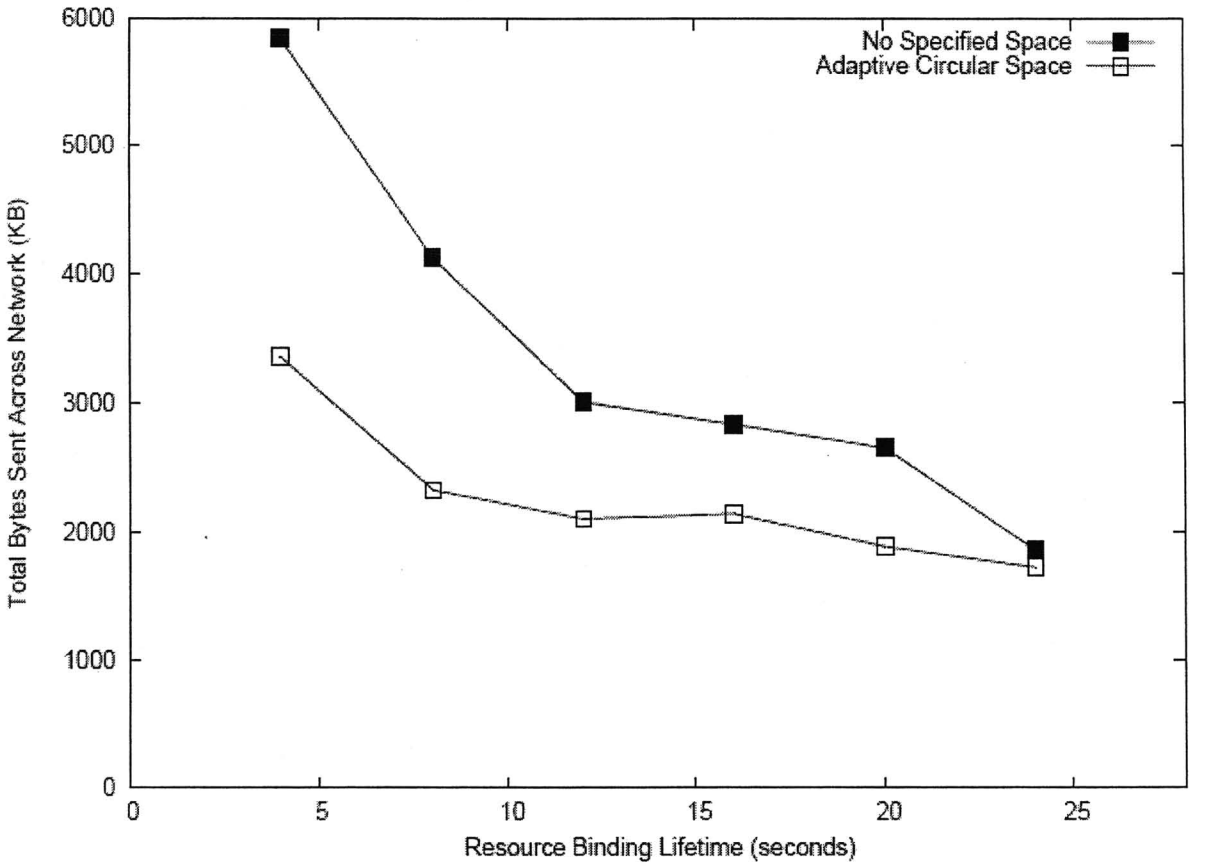
เหมือนดังเช่นวิธีการเขียนโปรแกรมอื่นๆ การเขียนโปรแกรมให้มีประสิทธิภาพจำเป็นที่จะต้องเข้าใจระบบที่ทำงานอยู่ด้วย ยกตัวอย่างเช่น ในระบบหน่วยความจำเสมือน เราควรเขียนโปรแกรมที่ทำให้เกิดจำนวน Page Faults น้อยที่สุด หากจะเขียนโปรแกรมให้กระทำการบนข้อมูลทุกตัวในอาร์เรย์สองมิติ ซึ่งรัน

บนระบบที่ใช้หน่วยความจำเสมือน เราควรเข้าถึงข้อมูลของอาร์เรย์ทีละแถว แทนที่จะเข้าถึงทีละคอลัมน์ในทางที่คล้ายกัน โปรแกรมประยุกต์ติดตามวัตถุของเราจะมีประสิทธิภาพที่มากขึ้นเมื่อมีการระบุพื้นที่ที่ต้องการค้นหา เพราะว่าไลบรารี่ของเราสนับสนุนการทำ Geographic Routing หากมีการระบุพื้นที่ คำร้องขอในการค้นหาทรัพยากรจะถูกส่งต่อตามตำแหน่งทางภูมิศาสตร์ไปยังพื้นที่เป้าหมายได้ แทนที่จะต้อง Flood คำร้องขอนี้ไปทั่วเครือข่าย



รูปที่ 7 ผลกระทบของการระบุพื้นที่ต่อระยะทางความผิดพลาดเฉลี่ย

เพื่อที่จะศึกษา ผลกระทบของการระบุตำแหน่งต่อโปรแกรมประยุกต์ติดตามวัตถุของเรา เราจึงได้ทำการทดสองคล้ายกับการทดลองที่ผ่านมา หากแต่ว่าครั้งนี้ โปรแกรมของเรามีบรรทัดที่ 25 ของรูปที่ 2 อยู่ด้วย เมื่ออายุขัยการผูกเพิ่มขึ้น เราจะประหยัดมากขึ้นเนื่องจาก เป็นการลดจำนวนครั้งในการค้นหาทรัพยากรลง นอกจากนี้ ความแม่นยำในการติดตามไม่ได้แย่ลงเมื่อมีการระบุพื้นที่ (รูปที่ 7)



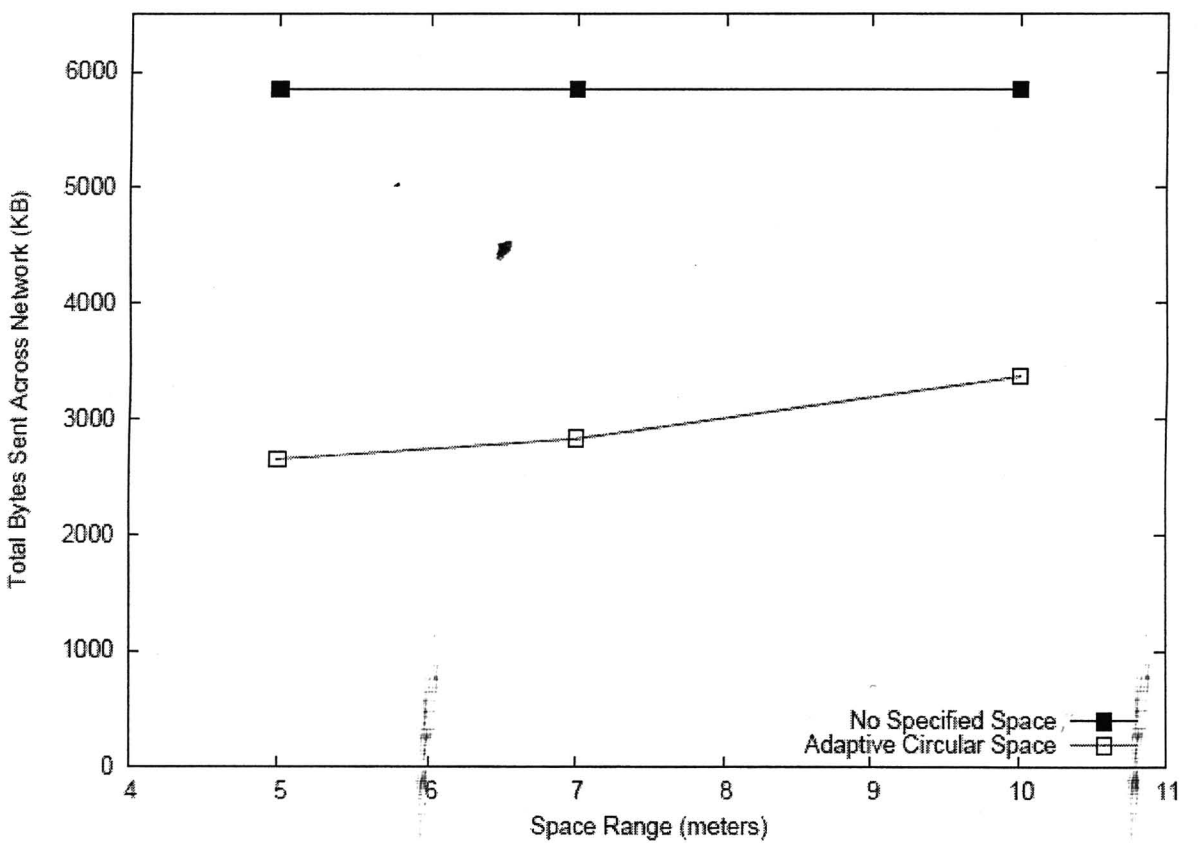
รูปที่ 8 แสดงผลกระทบของการระบุพื้นที่ต่อจำนวนไบต์ที่ต้องส่ง

ผลลัพธ์ของเราบ่งชี้ว่า การระบุพื้นที่เป้าหมายแบบพลวัตสามารถประหยัดจำนวนไบต์ในการส่งได้ถึง 42.5% เมื่อเทียบกับการไม่ระบุพื้นที่ (รูปที่ 8) ถึงแม้ว่าการประหยัดนี้จะมีนัยสำคัญ อาจมีผู้ที่คาดหวังว่าจะประหยัดได้มากกว่านี้ เพราะว่า Geographic Routing มีประสิทธิภาพมากกว่าการ Flooding มาก อย่างไรก็ตาม เมื่อคำขอค้นหาทรัพยากรได้ถูกส่งต่อเชิงภูมิศาสตร์ไปยังพื้นที่เป้าหมายแล้ว คำร้องขอนั้นจะถูก Flood ในพื้นที่จำกัดดังกล่าวอยู่ดี เพื่อค้นหาทรัพยากรทั้งหมดที่ตรงตามเงื่อนไข การ Flood แบบจำกัดพื้นที่นี้ มีโอเวอร์เฮด จึงยังผลให้ประหยัดน้อยกว่าที่คาดไว้บ้าง

4.5 ผลกระทบจากรัศมีของพื้นที่วงกลมที่ระบุ

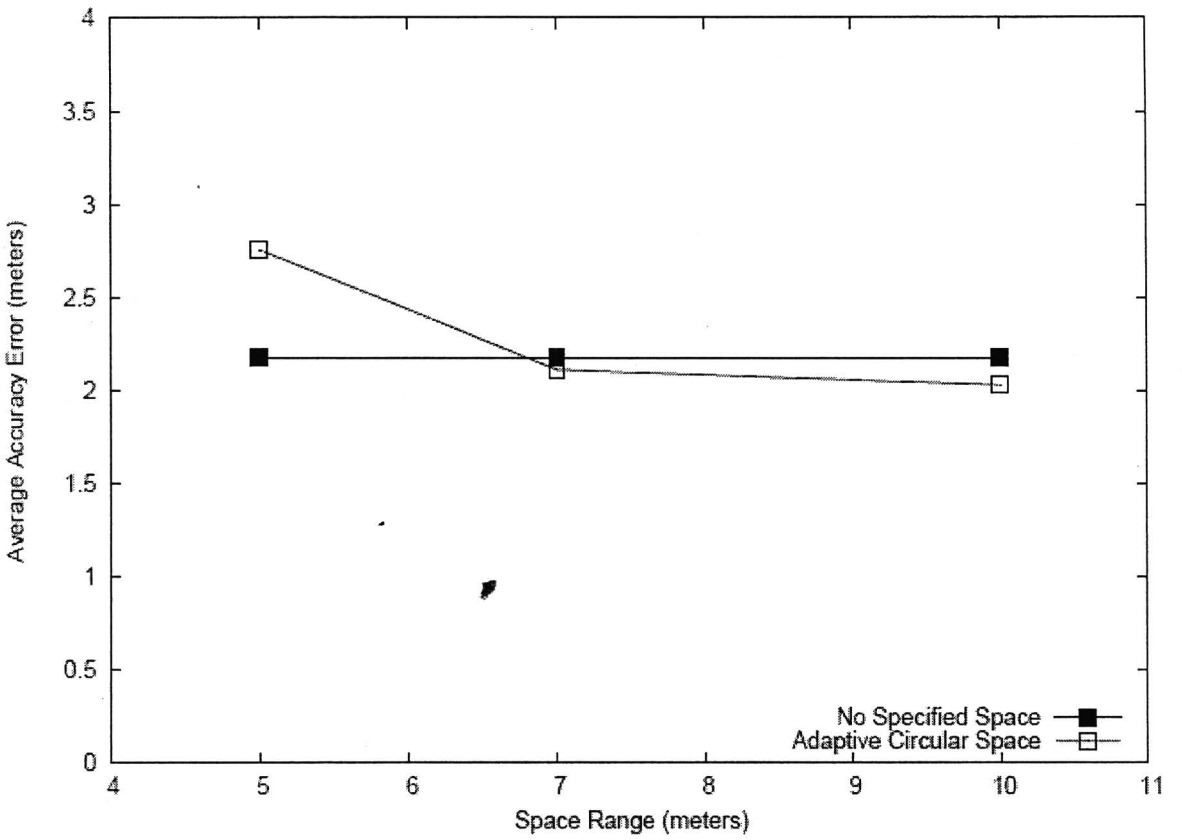
ตามธรรมดาแล้ว หากต้องการจะลดโอเวอร์เฮดให้น้อยที่สุด พื้นที่วงกลมที่ระบุควรมีขนาดเล็กที่สุดเท่าที่จะทำได้ ในการทดลองที่แล้ว เราใช้รัศมี 10 m พื้นที่ดังกล่าวดูเหมือนว่าจะมีขนาดใหญ่เกินความจำเป็น เมื่อเทียบกับรัศมีการรับรู้ 5 m ดังนั้น อาจมีผู้คาดหวังว่ารัศมีของพื้นที่เป้าหมายแค่ 5 m น่าจะเพียงพอแล้วสำหรับตัวรับรู้ทุกตัวที่ตรวจพบวัตถุในขณะหนึ่งๆ อย่างไรก็ตาม ความคาดหวังดังกล่าวค่อนข้างมองโลกในแง่ดีเกินไป เนื่องจากวัตถุมีการเคลื่อนที่อยู่ตลอดเวลา และการประมาณตำแหน่งย่อมมีความผิดพลาดอยู่ (ตำแหน่งที่ประมาณถูกใช้ในการกำหนดพื้นที่เป้าหมายในการค้นหาครั้งถัดไป)

เพื่อที่จะศึกษาผลกระทบของรัศมีพื้นที่เป้าหมายสำหรับโปรแกรมประยุกต์ติดตามวัตถุของเรา เราได้ทำการทดลองคล้ายกับการทดลองที่แล้ว หากแต่แตกต่างจากเดิมตรงที่เราใช้อายุขัยการผูก 4 วินาที และศึกษาสมรรถนะของระบบเมื่อรัศมีเปลี่ยนไป



รูปที่ 9 แสดงผลกระทบของรัศมีพื้นที่เป้าหมายที่ระบุต่อจำนวนไบต์ที่ส่ง

ผลการทดลองเป็นไปตามคาด กล่าวคือ เมื่อรัศมีเพิ่มขึ้น จำนวนไบต์ในการส่งเพิ่มขึ้นตาม (รูปที่ 9) ในขณะที่ความแม่นยำมากขึ้น (รูปที่ 10) ดังนั้น ผลการทดลองนี้บ่งชี้ว่า การใช้รัศมีพื้นที่เป้าหมาย 7 m เราสามารถรักษาความแม่นยำในระดับเดียวกับการค้นหาทั้งเครือข่ายได้ แต่จะประหยัดจำนวนไบต์ในการส่งได้ถึง 51.6% การใช้รัศมีพื้นที่ 7 m (หรือ 2m มากกว่ารัศมีการรับรู้) นั้นไม่น่าแปลกใจเช่นกัน เนื่องจากความผิดพลาดของเราอยู่ที่ประมาณ 2-3 m



รูปที่ 10 แสดงผลกระทบของรัศมีพื้นที่เป้าหมายที่ระบุต่อระยะทางผิดพลาดเฉลี่ย