**北京邮电大学**

**物联网技术课程期末技术报告**



**题 目： 智能联网定向灭火消防系统**

**学 院： 计算机学院**

**专 业： 计算机科学与技术**

**班 级： 2020211312**

**姓 名： 肖宇童**

**学 号： 2020212242**

日期:2023.5

# 应用系统创意

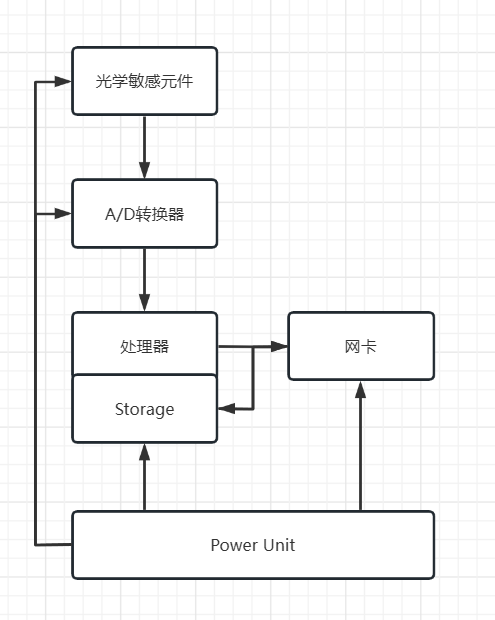
传统室内消防设施一般为烟雾报警器和消防喷淋头，在火灾初期能够起到一定效果，但并不能很好达到彻底消除火源的灭火效果。其中烟雾报警器要在室内烟雾达到一定浓度后才能检测报警并启动喷淋头，此时往往火势已经较大，而喷淋头的喷淋范围也比较有限且不能定位火源灭火，灭火效率较低，往往无法控制火势蔓延。

本系统基于物联网技术设计了一个智能联网定向灭火消防系统，并对系统进行了仿真模拟。系统采用基于光学传感器的火焰传感器作为检测火源的设备；使用具有联网功能的能够控制喷头方向的喷水器作为定向灭火装置；可使用多个处理机对整个系统进行信息处理和控制。

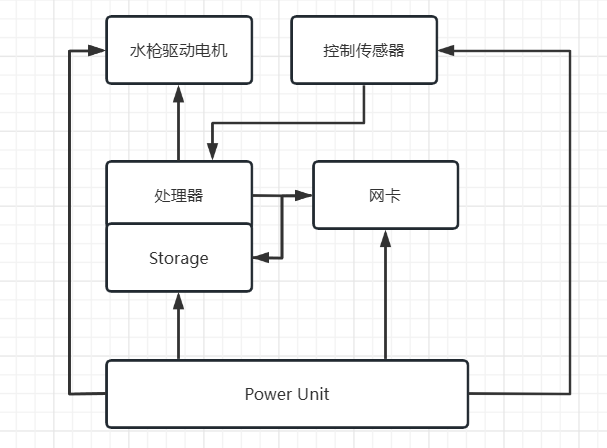
火焰传感器能够通过红外线、紫外线等作为输入物理量，对较大范围内进行火焰检测，能在火灾初期烟雾浓度不大时就检测并精确定位火源，并将信息上报处理机。喷水器内置网卡能通过Zigbee等无线通信协议自动接受和转发数据包，从而实现远程接收指令、转发信息和自由组网，从而在很大范围内组建灭火系统，可以适用于商场、酒店等大型建筑设施。处理机获取火焰传感器上报的火源信息，能够智能调度各喷水器向每个火源处精确喷水灭火。相比与传统消防设施，该系统应该能具备更好的消防能力，在火灾初期就能达到彻底消灭火源的灭火效果。

# 相关技术

火焰传感器：该系统使用光学火焰传感器作为检测火源的设备。光学火焰传感器利用火焰的可见光和红外辐射特性进行检测。其构成包括一个光敏元件(敏感元件)，如光敏二极管或光电二极管，和一个光学过滤器。可以以红外光作为输入物理量检测波长在700纳米～1000纳米范围内的火源发出的红外光，以紫外光作为输入物理量检测火源发出的波长在400纳米以下的热辐射。之后将这些接受到的光强通过A/D转换器(即模数转换器)转换为电流的强度，从而检测火焰。之后内置微型处理机对这些数字信号进行处理整合出火源强度、位置等信息，并使用无线通信技术将检测到的火源信息上报给远端处理机。

传感器节点结构主要分为数据采集模块、数据处理模块、无线通信模块、供电模块，结构图如下所示：

定向喷水器：使用低功耗电机作为水枪驱动源，以减少能耗和成本。可采用传感器如陀螺仪或加速度计等来检测和控制喷水器朝向和角度，以精确控制喷水方向。在未接收到传感器指令时进入待机睡眠模式，从而极大程度减少系统在平时的能量消耗。当从网卡接收到来自远端处理机的命令时，进入工作模式向目标位置喷水。

定向喷水器节点结构主要分为传感和控制模块、数据处理模块、无线通信模块、供电模块，结构图如下所示：

无线通信技术：本系统拟采用Zigbee作为消防设施间的无线通信技术。Zigbee使用低功耗设计，能够适用于需要长时间运行的消防设备，若传感器和喷水设备使用电池供电，则Zigbee能够降低设备的电量消耗，保证更长的工作时间。Zigbee支持各种网络拓扑结构，且具有自组网能力，能够在复杂的、大型的建筑物中进行消防设备的自由组网，并且Zigbee支持数据加密和认证机制，能够确保系统设备间通信安全稳定。

路由协议：由于Zigbee通信距离一般为10~100m范围，若系统需要适配的建筑物体积、范围较大，该通信范围一般不能覆盖系统的使用范围。本系统拟采用GPSR协议作为设备间通信的路由协议，由于系统中设备位置较为固定，可事先确定设备位置信息及其邻居节点的位置等信息，故可以使用GPSR协议更快捷有效地确定最短路由，提高系统处理效率和响应速度。当信息需要多跳才能到达目标节点时，可以使用如喷水器节点的路由功能，从而使其作为中间节点进行数据转发。

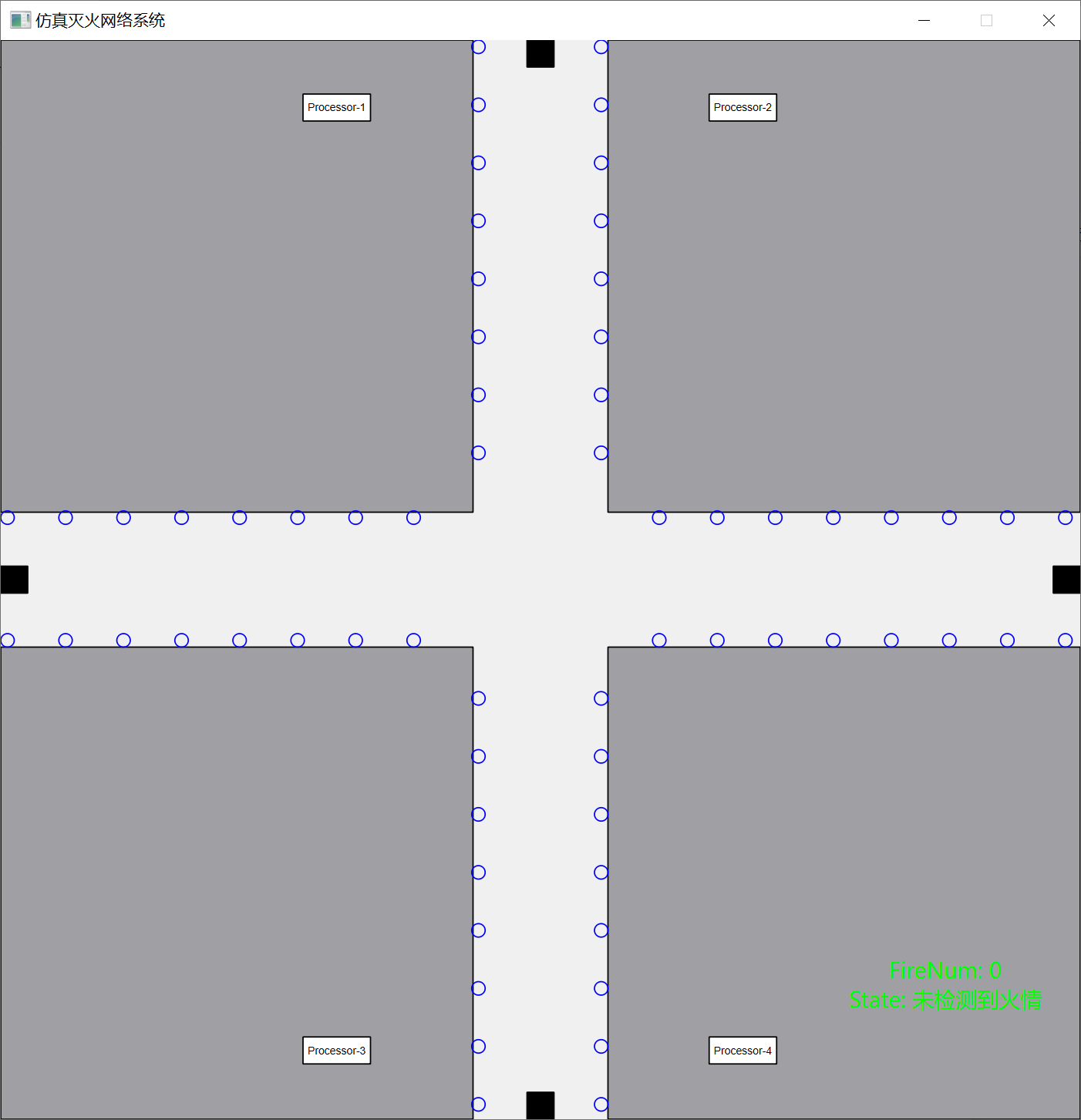
处理机：处理机放置于防火的安全位置，负责接收火焰传感器上报信息，如火源数量、位置、火势大小等，并通过调度算法对水枪进行调度和发送指令，对火焰实现压制、扑灭等操作。在系统规模较大时，一个处理机负责控制调度一定区域内的水枪。

# 系统的功能描述

1. 火源检测和定位：系统中的红外线热源检测传感器可以实时监测周围环境的热辐射，并识别火源或其他热源。传感器能够快速准确地检测到热源的存在，并确定其位置。
2. 实时数据传输：传感器模块通过无线通信技术如Zigbee将检测到的热源信息实时传输给控制系统。这种实时数据传输确保了控制系统能够及时获得最新的火源信息，以便做出相应的控制决策。
3. 智能化控制系统：处理机接收传感器上传的热源信息，并利用调度算法进行处理和分析。基于实时的热源位置和火势情况，控制系统能够智能地确定喷水器的转向和喷水策略。这使得系统能够根据实际情况做出快速、准确的灭火决策。
4. 水枪控制：控制系统通过系统网络将控制信号传输给喷水装置模块，实现对喷水器的转向和喷水操作的控制。根据控制系统的指令，喷水装置模块能够迅速调整喷水器的方向，并控制喷水器按照特定的喷水策略进行喷水，将水流集中于火源位置，实现灭火效果。
5. 远程监控和控制：该系统支持远程监控和控制功能，可以通过互联网连接进行远程访问。消防中心可以使用终端设备，远程监视系统状态、获取火源信息，并进行人为远程控制操作，例如手动控制喷水器转向或设置系统的工作模式。
6. 智能化决策和预测：控制系统可以利用机器学习或深度学习算法对火灾数据进行分析和预测。使用对应调度算法和调度策略对水枪资源进行调度，实现压制和扑灭火焰。

# 系统模块设计与实现

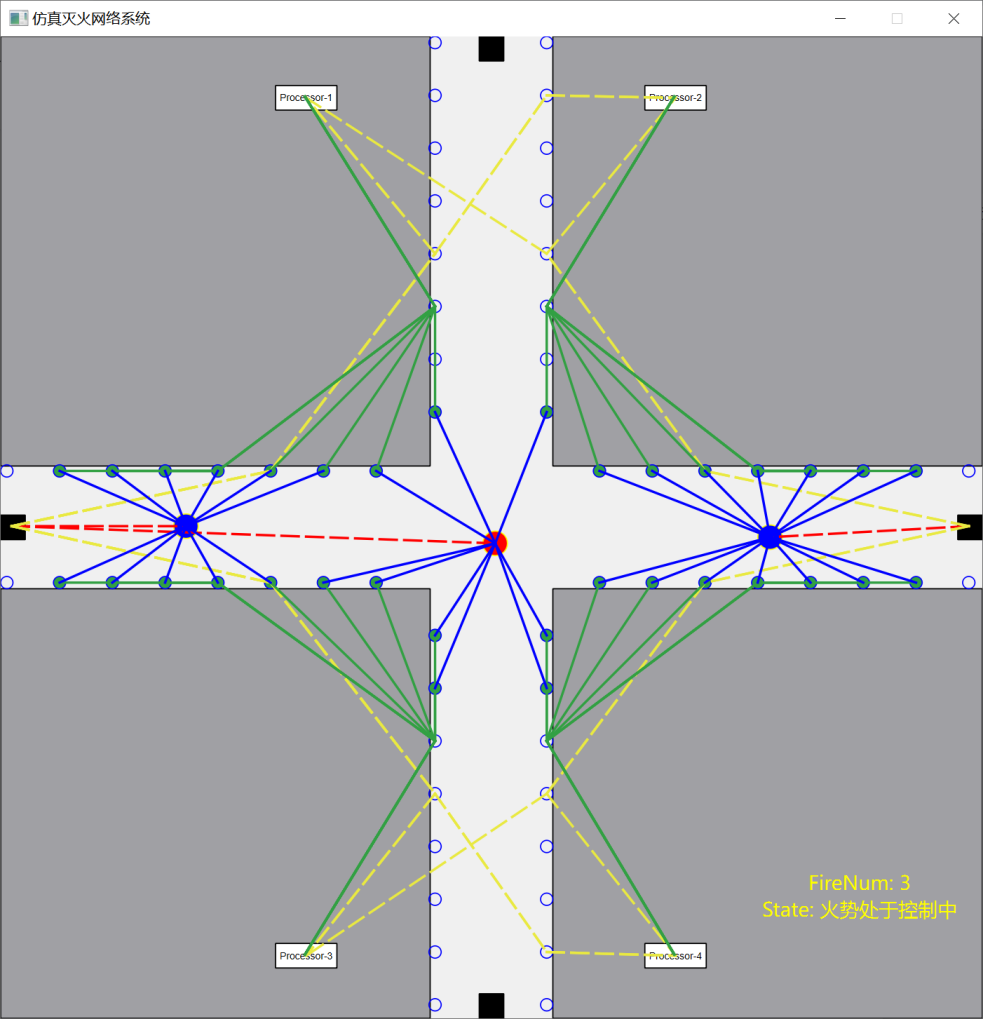
总体设计：本系统基于物联网技术设计了一个智能联网定向灭火消防系统，并对系统进行了仿真模拟。系统采用基于光学传感器的火焰传感器作为检测火源的设备；使用具有联网功能的能够控制喷头方向的喷水器作为定向灭火装置；可使用多个处理机对整个系统进行信息处理和控制，并调度水枪资源灭火。

仿真系统界面如图所示：

仿真系统模拟了在如大型商场、酒店等复杂建筑中交错的走廊安装的智能联网定向灭火消防系统。其中蓝色小圈代表水枪、黑色矩形代表火焰传感器、白色矩形代表远端处理机，中间白色区域为走廊。右下角显示系统检测到的火情状态。

系统主要分为物理模型绘制模块、传感器检测模块、通信路由模块、处理机调度模块。

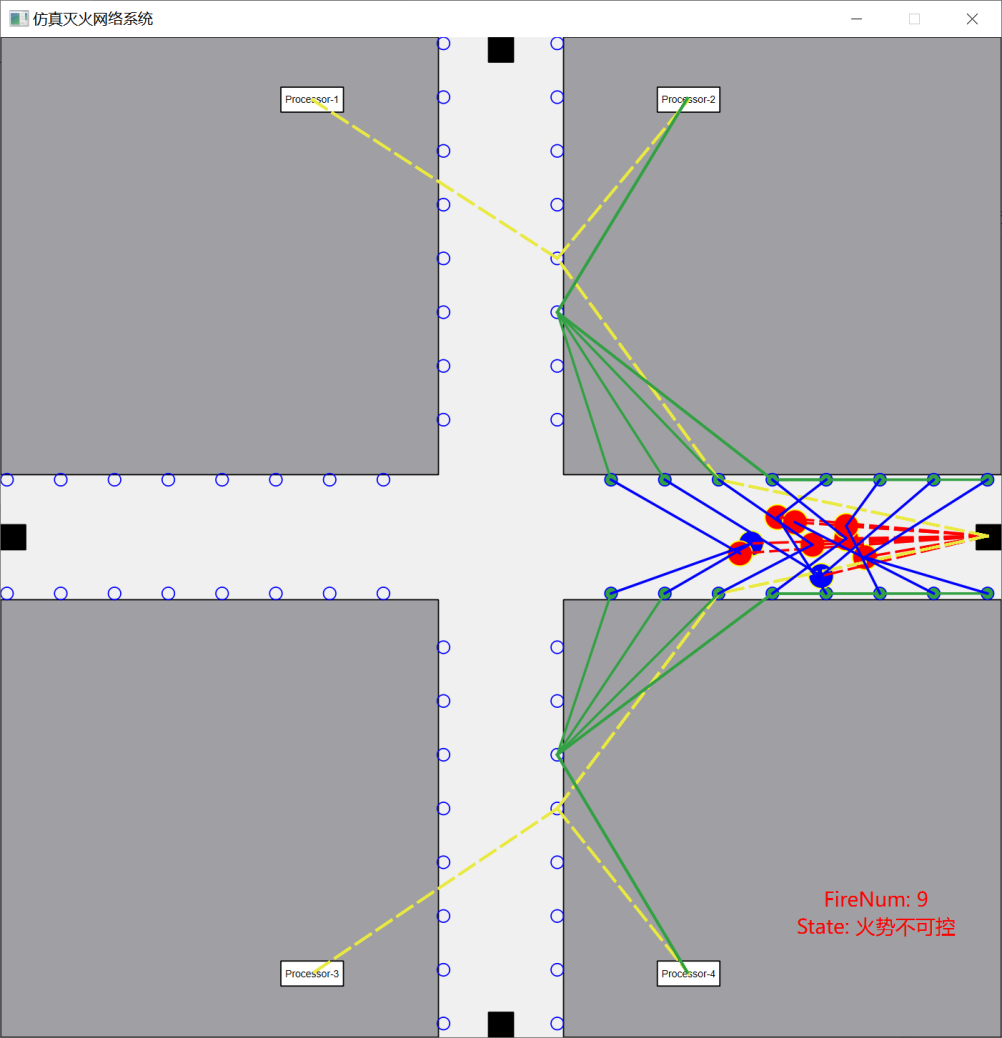
物理模型绘制模块负责场景绘制和火焰对象放置与销毁，以及数据流向绘制；传感器检测模块负责检测火焰对象并向处理机发送消息；通信路由模块负责对系统中设备间发送的消息进行基于GPSR的路由转发；处理机模块负责根据火源信息对水枪进行调度，对各火源进行合理的水枪资源分配从而实现压制火源和扑灭火源。

系统运行截图：

其中红点表示火源(可通过鼠标点击放置)、红蓝闪烁的点表示正被压制扑灭的火源，各色线条表示系统中的信息流向。其中红色虚线为火源信息被火焰传感器检测，黄色虚线表示火焰传感器的上报消息如何被路由转发到各处理机，绿色实线表示处理机的控制指令如何被路由转发到受调度的水枪处，蓝色实线表示水枪正在对对应火源进行灭火。

当每个火源都分配到了一定数量的水枪资源，则火源处于被压制状态，火情处于控制中，在此基础上火源分配到的水枪资源越多，其被扑灭速度就越快；若存在火源未分配到足够的水枪资源，则火情处于不可控状态，系统发出告警。

下图中，局部火源过多，超出系统水枪资源调度能力，火情处于不可控状态：



**关键代码：**

SPSR路由算法实现：

1. double dis(double x1, double y1, double x2, double y2) {
2. return sqrt((x1 - x2) \* (x1 - x2) + (y1 - y2) \* (y1 - y2));
3. }
4. void sendInfo(QRect \*sensor, QRect \*processor, Fire \*fire) {
5. double x1 = sensor->center().x(), y1 = sensor->center().y();
6. while(dis(x1, y1, processor->x()+ ProcessorWidth, processor->y()+ ProcessorHeight) > SendRange) {
7. double d2p = dis(x1, y1, processor->x()+ ProcessorWidth, processor->y()+ ProcessorHeight);
8. double x = x1, y = y1;
9. for(WaterGun \* waterGun : waterGuns) {
10. double x2 = waterGun->X + GunSize / 2, y2 = waterGun->Y + GunSize / 2;
11. double d = dis(x1, y1, x2, y2);
12. if(d < SendRange && dis(x2, y2, processor->x()+ ProcessorWidth, processor->y()+ ProcessorHeight) < d2p)  {
13. d2p = dis(x2, y2, processor->x()+ ProcessorWidth, processor->y()+ ProcessorHeight);
14. x = x2, y = y2;
15. }
16. }
17. if(x1 == x && y1 == y) { *// 不可达*
18. return;
19. }
20. QLine \*line = new QLine(x1, y1, x, y);
21. fire->infoLines.push\_back(line);
22. x1 = x, y1 = y;
23. }
24. QLine \*line = new QLine(x1, y1, processor->x() + ProcessorWidth / 2, processor->y() + ProcessorHeight / 2);
25. fire->infoLines.push\_back(line);
26. }
27. *// 处理机向水枪发送指令*
28. bool sendCommand(QRect \*Processor, WaterGun \*tarWaterGun, Fire \*fire){
29. double x1 = Processor->center().x(), y1 = Processor->center().y();
30. while(dis(x1, y1, tarWaterGun->X + GunSize / 2, tarWaterGun->Y + GunSize / 2) > SendRange) {
31. double d2g = dis(x1, y1, tarWaterGun->X + GunSize / 2, tarWaterGun->Y + GunSize / 2);
32. double x = x1, y = y1;
33. for(WaterGun \* waterGun : waterGuns) {
34. double x2 = waterGun->X + GunSize / 2, y2 = waterGun->Y + GunSize / 2;
35. double d = dis(x1, y1, x2, y2);
36. if(d < SendRange && dis(x2, y2, tarWaterGun->X + GunSize / 2, tarWaterGun->Y + GunSize / 2) < d2g)  {
37. d2g = dis(x2, y2, tarWaterGun->X + GunSize / 2, tarWaterGun->Y + GunSize / 2);
38. x = x2, y = y2;
39. }
40. }
41. if(x1 == x && y1 == y) { *// 不可达*
42. return false;
43. }
44. QLine \*line = new QLine(x1, y1, x, y);
45. fire->commandLines.push\_back(line);
46. x1 = x, y1 = y;
47. }
48. QLine \*line = new QLine(x1, y1, tarWaterGun->X + GunSize / 2, tarWaterGun->Y + GunSize / 2);
49. fire->commandLines.push\_back(line);
50. return true;
51. }

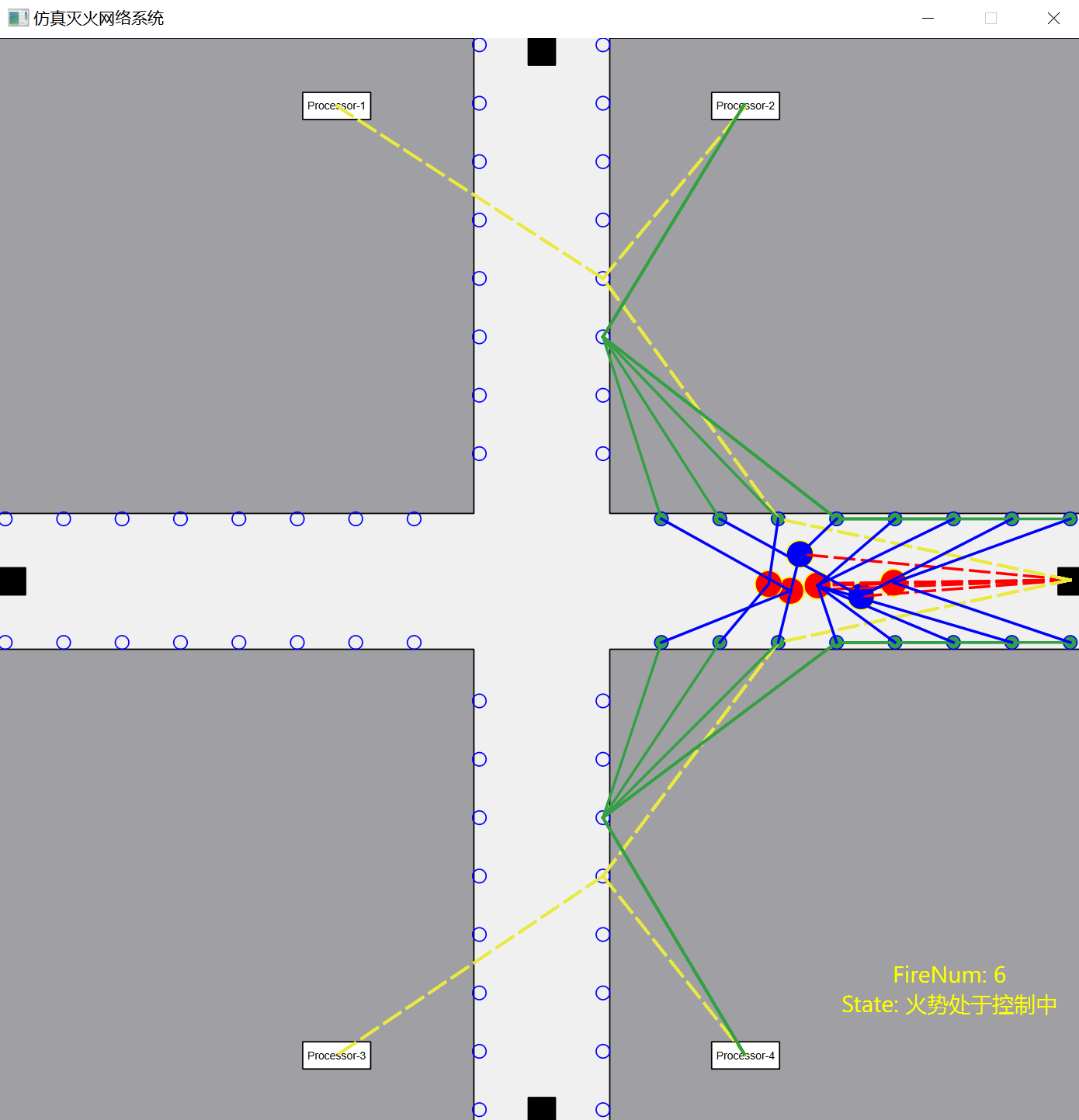
SendInfo()从传感器将火源信息传送给处理机，通过在每个途径节点的通信范围内搜寻距离目标处理机最近的下一个邻居节点，将该信息发送给该节点。若处理机处于节点通信范围内，则节点将信息直接发送给处理机，若不能则重复上一步骤。

SendCommand()从处理机处将命令发送给应参与扑灭或压制该处火源的水枪，如对应水枪处于通信范围之外无法直接将命令送达，则通过与SendInfo()类似算法流程，在每个途径节点的通信范围内搜寻距离目标水枪最近的下一个邻居节点，将该信息发送给该节点，最终将命令转发给目标水枪。

水枪灭火调度程序：

1. void putFire(Fire \*fire) {
2. double x1 = fire->X + FireSize / 2, y1 = fire->Y + FireSize / 2;
3. for(WaterGun \*waterGun : waterGuns) {
4. double x2 = waterGun->X + GunSize / 2, y2 = waterGun->Y + GunSize / 2;
5. if(dis(x1, y1, x2, y2) < AttackRange && waterGun->state == Idle) {
6. *// 检查水枪是否被墙壁挡住*
7. QLineF lineF = QLineF(x1, y1, x2, y2);
8. QRectF topLeftRectF = QRectF(topLeftRect->x(), topLeftRect->y(), topLeftRect->width(), topLeftRect->height());
9. QRectF topRightRectF = QRectF(topRightRect->x(), topRightRect->y(), topRightRect->width(), topRightRect->height());
10. QRectF bottomLeftRectF = QRectF(bottomLeftRect->x(), bottomLeftRect->y(), bottomLeftRect->width(), bottomLeftRect->height());
11. QRectF bottomRightRectF = QRectF(bottomRightRect->x(), bottomRightRect->y(), bottomRightRect->width(), bottomRightRect->height());
12. if(isLineRectOverlap(lineF, topLeftRectF) || isLineRectOverlap(lineF, topRightRectF) || \
13. isLineRectOverlap(lineF, bottomLeftRectF) || isLineRectOverlap(lineF, bottomRightRectF)) {
14. continue;
15. }
16. int flag = false; *// 指令是否能够送达*
17. if(waterGun->belongProcessor == 1) {
18. flag = sendCommand(topLeftProcessor, waterGun, fire);
19. } else if(waterGun->belongProcessor == 2) {
20. flag = sendCommand(topRightProcessor, waterGun, fire);
21. } else if(waterGun->belongProcessor == 3) {
22. flag = sendCommand(bottomLeftProcessor, waterGun, fire);
23. } else if(waterGun->belongProcessor == 4) {
24. flag = sendCommand(bottomRightProcessor, waterGun, fire);
25. }
26. if(flag) {
27. QLine \*line = new QLine(x1, y1, x2, y2);
28. waterGun->state = Attack;
29. fire->attackLines.push\_back(line);
30. }
31. }
32. }
33. *// 强制调度水枪压制*
34. while(fire->attackLines.size() < SuppressNeed) {
35. *// 按attackLines的长度排序从大到小*
36. bool breakFlag = false;
37. sort(fires.begin(), fires.end(), [](Fire \*a, Fire \*b) {
38. return a->attackLines.size() > b->attackLines.size();
39. });
40. *// 从占用最多资源的火焰中分取一个可达水枪*
41. for(vector<Fire\*>::iterator it\_fire = fires.begin(); it\_fire != fires.end(); it\_fire ++) {
42. if((\*it\_fire)->attackLines.size() <= SuppressNeed) return; *// 没有资源压制*
43. for(vector<QLine\*>::iterator it\_line = (\*it\_fire)->attackLines.begin(); it\_line != (\*it\_fire)->attackLines.end(); it\_line ++) {
44. double x2 = (\*it\_line)->x2(), y2 = (\*it\_line)->y2();
45. if(dis(x1, y1, x2, y2) < AttackRange){
46. *// 检查水枪是否被墙壁挡住*
47. QLineF lineF = QLineF(x1, y1, x2, y2);
48. QRectF topLeftRectF = QRectF(topLeftRect->x(), topLeftRect->y(), topLeftRect->width(), topLeftRect->height());
49. QRectF topRightRectF = QRectF(topRightRect->x(), topRightRect->y(), topRightRect->width(), topRightRect->height());
50. QRectF bottomLeftRectF = QRectF(bottomLeftRect->x(), bottomLeftRect->y(), bottomLeftRect->width(), bottomLeftRect->height());
51. QRectF bottomRightRectF = QRectF(bottomRightRect->x(), bottomRightRect->y(), bottomRightRect->width(), bottomRightRect->height());
52. if(isLineRectOverlap(lineF, topLeftRectF) || isLineRectOverlap(lineF, topRightRectF) || \
53. isLineRectOverlap(lineF, bottomLeftRectF) || isLineRectOverlap(lineF, bottomRightRectF)) {
54. continue;
55. }
56. QLine \*line = new QLine(x1, y1, x2, y2);
57. (\*it\_fire)->attackLines.erase(it\_line);
58. fire->attackLines.push\_back(line);
59. *// 取到资源后退出内层循环*
60. breakFlag = true;
61. break;
62. }
63. }
64. if(breakFlag) break;
65. }
66. *//*
67. }
68. }

调度算法思想：先进行火源位置分析，判断水枪是否能够作用于火源，即火源是否处于射击范围内、是否被障碍物挡住。再检查目标水枪连接是否正常，即指令是否能够成功到达水枪。之后调动尽可能多的能够正常接收指令且能够作用于火源位置(未被障碍物挡住，且火源在水枪攻击范围之内)的水枪资源进行灭火。若出现新的火源，则在有足够多作用于其它火源的水枪资源的前提下，仅调用足够多的能够压制新火源的水枪数目来压制该火源。在其它火源被集中扑灭后，空闲且能够作用于该火源的水枪会集中来扑灭该火源。若有火源无法得到足够压制其火焰的水枪数，则火势处于不可控状态，系统告警。

抢占式调度：如图

(新出现火源抢占足够  
水枪资源，压制火焰防止  
火势扩大)

# 系统可能的扩展

系统可能的改进与扩展方向:

1. 多种类型的传感器：除了火焰传感器，可以考虑集成其他类型的传感器，如温度传感器、气体传感器、化学物质传感器等，以便更全面地监测火灾情况。这样可以提供更多的信息来评估火灾的严重程度和蔓延风险。
2. 结合AI进行数据融合分析以及预测功能：通过使用机器学习和数据分析技术，对传感器数据进行实时分析和处理，可以提前预测火灾蔓延趋势，并采取相应的灭火策略。这样可以更好地控制火势，减少灾害损失。
3. 多通信协议支持：除了使用Zigbee作为无线通信协议，可以考虑支持其他通信协议，如LoRaWAN或NB-IoT，以提供更大的通信范围和更稳定的连接。
4. 更多种类灭火装置：除了喷水器，可以考虑集成其他类型的灭火装置，如灭气剂、灭油剂等，以应对不同类型的火灾场景。
5. 更多远程监控和管理手段：通过云平台和移动应用，实现对消防系统的远程监控和管理。可以远程查看传感器数据、灭火装置状态，并进行远程控制和调度。
6. 智能报警和通知系统：集成声音、光线或手机短信等多种报警方式，能够及时通知相关人员和消防部门，提高火灾应对的速度和效率。

# 对物联网技术的理解分析