|  |
| --- |
| <시뮬레이션 진행 순서>  1. 시작  2. 프로토콜 선택  3. 노드 추가/제거  4. 클러스터 재구성  5. 연관 노드 설정  6. 키 그래프 (재) 구성  7. leaf 노드 – 실제 노드 대응 및 물리적 그룹화  8. 그룹 키 갱신  9. 이벤트 발생  10. 이벤트 보고서 생성  11. 이벤트 보고서 전달 및 필터링  12. 실험 종료 조건 확인  14. 통계 지표 도출, 기록  15. 종료 |

< 시뮬레이션 소스 –구조 정리 >

<Routing 관련>

Rtn class: 라우팅 프로토콜 클래스

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // cluster 구성 관련 |  |  |
| // 클러스터 구성 관련  void setNodes(map<int, Node\*> nodes); // 노드 리스트 초기화 함수  void setBS( Node\* bs); // 기지 노드 객체 주소 설정  virtual void findNbrs(); // 이웃 노드 탐색 함수  virtual void setCHs() = 0; // CH 선택 함수  Node\* getCH (int clstrID); // CH 획득 함수  set<int> getClstr (int clstrID); //(2014.05.21)입력: 클러스터 ID (ch's ID), 출력: 클러스터 소속 노드들의 iD 집합  map<int, set<int>> getClusters(); // 클러스터 정보 반환  int getClstrID (int nodeID); // (2014.04.10) 입력: 노드 id, 출력: 클러스터 ID (= CH's ID)  void changeCluster (int nodeID, int clstrID); // 클러스터 변경 함수  bool hasUpdatedCH (int nodeID);// (2014.05.19) 주어진 nodeID에 대응되는 노드의 클러스터 (CH' ID)가 변경되었는지 검사 |  |  |
|  |  |  |
| // 다음 노드 선택 관련  void addParent (Node\* cNode, Node\* prevNode);  // (2013.10.11 추가) 부모 노드 추가 (한 노드가 여러 개의 부모 노드를 가질 수 있음)  std::list<Node\*> getParents(Node\* cNode); // // (2013.10.11 추가) 부모 노드 리스트 반환  Node\* getSPParent (Node\* cNode); // (2013.09.09) 부모 노드 반환  void setKmr( Kmr\* kmr); // (2012.11.10 추가) kmr 객체 멤버변수 설정  void setFuzzy (Fuzzy\* fuzzy); // fuzzy 객체 멤버변수 설정  virtual list<Node\*> selNext (struct \_tagCntrlMsg\* cntrlMsg, BS\* bs) = 0;  // (2014.05.20) 제어 메시지 전달 경로 상의 다음 노드 선택  virtual list<Node\*> selNext (struct \_tagCntrlMsg\* cntrlMsg, Node\* cNode) = 0;  // (2013.08.06) 제어 메시지 전달 경로 상의 다음 노드 선택  virtual Node\* selNext (struct \_tagEvntMsg\* evntMsg, Node\* cNode) = 0;  // 데이터 메시지 전달 경로 상의 다음 노드 선택  virtual list<Node\*> selNext (struct \_tagTREQ\* treq, Node\* cNode) = 0;  // (2014.05.10) TREQ 메시지 다음 전달 노드 결정 함수  virtual list<Node\*> selNext (struct \_tagTRES\* tres, Node\* cNode) = 0;  // (2014.05.10) TRES 메시지 다음 전달 노드 결정 함수    virtual double eval (struct \_tagCntrlMsg\* cntrlMsg, Node\* candi) = 0;  // (2013.08.06) 제어 메시지 전달을 위한 후보 노드 평가  virtual double eval (struct \_tagEvntMsg\* evntMsg, Node\* candi) = 0;  // 데이터 메시지 전달을 위한 후보 노드 평가  virtual double eval(Node\* recipent, Node\* candi) = 0; // 키 재분배 메시지 전달을 위한 후보 노드 평가 |  |  |
|  |  |  |
| // 키 재분배 메시지 처리 관련  virtual list<Node\*> selNext (struct \_tagKeyMsg\* keyMsg, BS\* bs) = 0;  virtual list<Node\*> selNext (struct \_tagKeyMsg\* keyMsg, Node\* cNode) = 0;  void setRecipent (struct \_tagKeyMsg\* keyMsg, Node\* next);  // (2014.05.12) 키 재분배 메시지의 목적 노드 설정 |  |  |
|  |  |  |
| // topology 관련  void resetTopology(); // topology 재 초기화 |  |  |
|  |  |  |
| // TREQ, TRES 메시지 처리 관련  void genTREQMsg (int round); // bs에서 TREQ 메시지 생성  struct \_tagTREQ\* getTREQMsg(); // TREQ 메시지 획득  void updatTREQMsg (Node\* cNode); // 중간 노드에서 TREQ 메시지 필드 업데이트  void genTRESMsg (Node\* cNode);  struct \_tagTRES\* getTRESMsg(); // TRES 메시지 획득  void handleTREQ(struct \_tagTREQ\* treq, Node\* cNode); // 수신된 TREQ 메시지 처리  int getTRESMsgLen(struct \_tagTRES\* tres); // TRES 메시지 길이 반환  void resetNxtNds(); // 다음 노드 목록 초기화 |  |  |
|  |  |  |
|  |  |  |
| // sorting 관련  void quicksort( int start, int finish);  int partition( int start, int finish); |  |  |
|  |  |  |
| // test 함수들  void testNbrs(); // 각 노드의 이웃 노드 정보 파일에 출력  void testCHs(); // 각 클러스터 CH 정보 파일에 출력  void testCandis(); // 후보 부모 노드 정보 테스트 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
| NodesMap nodes; // 노드 목록 ( {node ID, node pointer) } | | |
| std::map<int, int> ndsClstrMap; // (2014.04.10) <노드 ID - 클러스터 ID > ( 클러스터 ID = CH's ID) | | |
| std::map<int, std::set<int> > clusters; // (2014.04.10) <클러스터 ID - {노드 ID} > (클러스터에 속한 노드들의 ID 집합 관리) | | |
|  | | |
| NodesMap CHs; // CH 목록 ( {cluster ID, CH pointer) } | | |
| BS\* bs; // 기지 노드 객체 주소 | | |
| map<int, NodesLst> nbrLsts; // 각 노드의 이웃 노드 목록 | | |
| map<int, NodesLst> parentsLsts; // 각 노드의 후보 부모 노드 목록 | | |
| double eval\_result[MAX\_NUM\_PARENTS]; // 후보 부모 노드 평가 결과 | | |
| NodesLst nxtNds; // 키 분배를 위한 다음 전달 노드 목록 | | |
| Kmr\* kmr; // 키 관리자 객체 | | |
| Fuzzy\* fuzzy; // fuzzy 객체 | | |
| RAND gen1; // (2014.03.24) CH 선택을 위한 난수 생성기 | | |
| unsigned int seed1; // CH 선택을 위한 seed | | |

Rtn\_IHA class: IHA에서 사용되는 라우팅 프로토콜 (최단 경로 라우팅)

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // cluster 구성 관련 |  |  |
| void findNbrs(); // 이웃 노드 탐색 함수 |  |  |
| void setCHs() = 0; // CH 선택 함수 |  |  |
|  |  |  |
| Node\*\* selNext (struct \_tagCntrlMsg\* cntrlMsg, Node\* cNode);  // 제어 메시지 전달 경로 상의 다음 노드 선택 |  |  |
| Node\* selNext (struct \_tagEvntMsg\* evntMsg, Node\* cNode);  // 데이터 메시지 전달 경로 상의 다음 노드 선택 |  |  |
| virtual double eval (struct \_tagCntrlMsg\* cntrlMsg, Node\* candi) = 0;  // 제어 메시지 전달을 위한 후보 노드 평가 |  |  |
| virtual double eval (struct \_tagEvntMsg\* evntMsg, Node\* candi) = 0;  // 데이터 메시지 전달을 위한 후보 노드 평가 |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |

Rtn\_DIHA class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // cluster 구성 관련 |  |  |
| void findNbrs(); // 이웃 노드 탐색 함수 |  |  |
| void setCHs() = 0; // CH 선택 함수 |  |  |
|  |  |  |
| Node\*\* selNext (struct \_tagCntrlMsg\* cntrlMsg, Node\* cNode);  // 제어 메시지 전달 경로 상의 다음 노드 선택 |  |  |
| Node\* selNext (struct \_tagEvntMsg\* evntMsg, Node\* cNode);  // 데이터 메시지 전달 경로 상의 다음 노드 선택 |  |  |
| virtual double eval (struct \_tagCntrlMsg\* cntrlMsg, Node\* candi) = 0;  // 제어 메시지 전달을 위한 후보 노드 평가 |  |  |
| virtual double eval (struct \_tagEvntMsg\* evntMsg, Node\* candi) = 0;  // 데이터 메시지 전달을 위한 후보 노드 평가 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |

<Node 관련>

Node class: 센서 네트워크를 구성하는 노드의 기본적인 속성과 기능 정의. 특정 프로토콜에 종속되지 않는 기능만을 포함

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
|  |  |  |
| void setKmr(Kmr\* kmr); // 키 관리 객체 설정 |  |  |
| void setRtn(Rtn\* rtn); // 라우팅 객체 설정 |  |  |
| void setSec(Sec\* sec); // 보안 객체 설정 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
| int id; | | |
| int x, y; | | |
| double distance; // sink 노드까지의 거리 | | |
| double remEnrgy; // 노드의 잔여 에너지량 | | |
| double eCnsmptn; // 노드의 에너지 소모량 | | |
| static double totEnrgCnsmptn; // 노드들의 전체 에너지 소모량 | | |
| bool cmprmsd; // 노드가 훼손된 경우 1, 정상인 경우 0 | | |
|  | | |
| Rtn\* rtn; // 라우팅 객체 | | |
| Sec\* sec; // 보안 객체 | | |
| Kmr\* kmr; // 키 관리 객체 | | |

BS class:

|  |
| --- |
|  |

<Attack 관련>

Atk class

|  |
| --- |
|  |

Atk\_FRA class

|  |
| --- |
|  |

<Security 관련>

Sec class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // 멤버 변수 값 설정 |  |  |
| void setNodes(NodesMap nodes); // 노드 목록 설정 |  |  |
| void setRtn (Rtn\* rtn); // 라우팅 객체 주소 설정 |  |  |
| void setKmr( Kmr\* kmr); // 키 관리자 객체 주소 설정 |  |  |
|  |  |  |
| // 이벤트 보고서 처리 관련 |  |  |
| **virtual void setMAC (struct \_tagEvntMsg\* msg) = 0; // MAC 초기 생성** |  |  |
| **virtual void setMAC (struct \_tagEvntMsg\* msg, int clstrID) = 0; // MAC 초기 생성 (클러스터 기반)** |  |  |
| **virtual bool verMAC (struct \_tagEvntMsg\* msg) = 0; // MAC 검증 (성공 시 true, 실패 시 false 반환)** |  |  |
| **virtual void updateMAC(struct \_tagEvntMsg\* msg, int clstrID) = 0; // MAC 갱신** |  |  |
| **virtual void doEventMsgProc(struct \_tagEvntMsg\* msg); // 수신된 이벤트 보고서 처리** |  |  |
|  |  |  |
| // 제어 메시지 처리 관련 |  |  |
| **virtual void genCntrlMsg(enum ctrlMsgType type, Node\* cNode)=0; // 제어 메시지 생성** |  |  |
| **virtual void testCntrlMsg(); // 제어 메시지 테스트** |  |  |
| CntrlMsg\* getCntrlMsg(); // 제어 메시지 획득 |  |  |
| void setCntrlMsg(int clstrID); // 제어 메시지 내 클러스터 ID 설정 |  |  |
| **virtual void doCntlOp (struct \_tagCntrlMsg\* msg, Node\* cNode) = 0; // 수신된 제어 메시지 처리** |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **// 에너지 소모량 계산 관련** |  |  |
| **virtual int getEventMsgLen(struct \_tagEvntMsg\* msg) = 0; // 이벤트 보고서 길이 반환** |  |  |
| **virtual int getCntrlMsgLen(struct \_tagCntrlMsg\* cm) = 0; // 제어 메시지 길이 반환** |  |  |
|  |  |  |
| **// 공격 탐지 관련** |  |  |
| **virtual bool detect (struct \_tagEvntMsg\* msg) = 0; // 공격 검출** |  |  |
| int getNumDtctn(); // 공격 탐지 횟수 반환 |  |  |
| int getNumVer(); // MAC 검증 횟수 반환 |  |  |
| // 연관 노드 갱신 |  |  |
| **void updateAssocNodes(int nodeID, int assocType, set<int> assocNodes) = 0** |  |  |
| 변수 | | |
| // 객체 멤버 변수  NodesMap nodes; // 노드 목록 ( {node ID, node pointer) } | | |
| Rtn\* rtn; // 라우팅 객체 | | |
| Kmr\* kmr; // 키 관리자 객체 | | |
| struct \_tagCntrlMsg cntlMsg; // 제어 메시지 | | |
|  | | |
|  | | |
| // 공격 탐지 관련  bool atkDtctd; // 공격 탐지 여부 (예: 허위 보고서 공격) | | |
| int numDtctn; // // 공격 탐지 횟수 (예: 허위 보고서 공격) | | |
| int numVer; // 보고서 검증 횟수 | | |
| // MAC 연산 관련  std::list<struct \_tagMAC\*> tmplst;// (2013.11.19) 임시 MAC 목록 (보고서 내 MAC 갱신 과정에서 사용) | | |
|  | | |
| // | | |
|  | | |

Sec\_IHA class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |

Sec\_DIHA class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |
|  | | |

<Key Mngt. 관련>

Kmr class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // 멤버 변수 값 설정 |  |  |
| void setNodes(map<int, Node\*> nodes); // 노드 리스트 초기화 함수 | main |  |
| void setRtn (Rtn\* rtn); // rtn 객체 주소 저장 | main |  |
| void setSec (Sec\* sec); // sec 객체 주소 저장 |  |  |
|  |  |  |
| // 그룹 키 재분배 관련 함수 |  |  |
| virtual void updateKey(struct \_tagKNode\* kNode)=0; // 키 트리 상의 그룹 키 갱신 |  |  |
| virtual void genKeyMsg()=0; // 키 재분배 메시지 생성 |  |  |
| void initKeyMsg( struct \_tagKeyMsg\* kMsg); // 키 재분배 메시지 초기화 |  |  |
| map<int, struct \_tagKeyMsg\*> getKeyMsgs( ); // 키 재분배 메시지 획득 |  |  |
| void delKeyMsg(); // 키 재분배 메시지 제거 |  |  |
| void delRecvdKMsg(Node\* cNode); // (2014.05.15)수신된 키 재분배 메시지 제거 |  |  |
| virtual void testKeyMsg()=0; // 키 재분배 메시지 테스트 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| **// 키 재분배 메시지 처리 관련** |  |  |
|  |  |  |
| virtual void genKeyMsg()=0; // 키 재분배 메시지 생성 |  |  |
| virtual void testKeyMsg()=0; // 키 재분배 메시지 테스트 |  |  |
| **void handleKeyMsg(struct \_tagKeyMsg\* keyMsg, Node\* cNode); // 키 메시지 처리 함수** |  |  |
| // 필터링 키 관리 |  |  |
| virtual void addPairKey (int nID, int nID2) = 0;  // (2013.08.29) 두 노드의 iD를 입력으로 pairwise key를 추가 // pair key 추가 |  |  |
| PairKey\* getPairKey (int nID1, int nID2);  // (2013.08.19) 두 노드가 공유하는 pairwise key 계산 및 반환 // pair key 반환 |  |  |
| GroupKeyLst getKeySet(NodesLst nodes); // 주어진 노드(들)의 그룹 키 목록 반환 |  |  |
| NodesLst getUserSet(GroupKeyLst keys); // 주어진 키(들)의 노드 목록 반환 |  |  |
| // key tree 관리 |  |  |
| virtual void cnstrctKTree() = 0; // key tree 구축 |  |  |
| virtual void dstrctKTree() = 0 // key tree 제거 |  |  |
| void addKTreeNode( KNode\* joinPoint, KNode\* newNode); // key tree node 추가 |  |  |
| void deleteKTreeNode( KNode\* leavePoint, KNode\* delNode); // key tree node 제거 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
| NodesMap nodes; // 노드 목록 ( {node ID, node pointer) } | | |
| NodesMap CHs; // CH 목록 ( {cluster ID, CH pointer) } | | |
| Sec\* sec; | | |
| // 키 재분배 관련 | | |
| **map<int, struct \_tagKeyMsg\*> keyMsgs; // 키 재분배 메시지** | | |
| KNode\* root;// 키 트리의 root | | |
| AStar astar; | | |
| GA ga; | | |

Kmr\_IHA class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| void genKeyMsg(); // 키 재분배 메시지 생성 |  |  |
| KeyMsg\* getKeyMsg(); // 키 재분배 메시지 반환 |  |  |
| // 필터링 키 관리 |  |  |
| void addPairKey (int nID, int nID2) = 0;  // (2013.08.29) 두 노드의 iD를 입력으로 pairwise key를 추가 // pair key 추가 |  |  |
| // key tree 관리 |  |  |
| void cnstrctKTree(); // key tree 구축 |  |  |
| void dstrctKTree(); // key tree 제거 |  |  |
|  |  |  |
| 변수 | | |
|  | | |
|  | | |
|  | | |

Kmr\_NEWLKH class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| void genKeyMsg(); // 키 재분배 메시지 생성 |  |  |
| KeyMsg\* getKeyMsg(); // 키 재분배 메시지 반환 |  |  |
| // 필터링 키 관리 |  |  |
| void addPairKey (int nID, int nID2) = 0;  // (2013.08.29) 두 노드의 iD를 입력으로 pairwise key를 추가 // pair key 추가 |  |  |
| // key tree 관리 |  |  |
| void cnstrctKTree(); // key tree 구축 |  |  |
| void dstrctKTree(); // key tree 제거 |  |  |
| void expandLevel(int degree);  //key tree의 현재 level 확장 (현재 level의 각 노드마다 degree 만큼의 자식 노드 생성) | Kmr\_NEWLKH |  |
| 변수 | | |
|  | | |
|  | | |
|  | | |

<Fuzzy logic 관련>

fuzzy class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
|  | | |
|  | | |
|  | | |

<GA 관련>

GA class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // 멤버 변수 설정 관련 |  |  |
| void setNodes(map<int, Node\*> nodes); // 노드 리스트 초기화 함수 |  |  |
| void setKmr(Kmr\* kmr); // 키 관리자 객체 설정 |  |  |
| void setIndiLen(int indiLen);// 개체 길이 설정 |  |  |
| void setRoot(struct \_tagKNode\* root); // key tree root 설정 |  |  |
|  |  |  |
| // 세대의 생성, 평가 |  |  |
| void create\_population(); // 세대 생성 |  |  |
| void writeGroupIDs(struct \_tagKNode\* currentNode, int individual, int gene );  // 그룹 id 기록 (어느 개체의 몇 번째 유전자에 기록?) |  |  |
| void writeNodeIDs(int index); // 노드 id 기록 |  |  |
| void evaluate(); // 세대 평가 |  |  |
| double f(int individual, int level, int index);  // 주어진 개체에서 특정 그룹에 속한 노드들의 무게중심(COG)으로부터 각 노드까지의 평균 거리 반환 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
|  | | |
| map<int, Node\*> nodes; // 노드 목록 ( {node ID, node pointer) } | | |
| Kmr\* kmr; // 키 관리자 객체 | | |
| int indiLen; // 염색체의 길이 (가변적) | | |
| map<int, pair<int, int>> individual; // 개체 (염색체) - (인덱스, 값) | | |
| int geneIdx; // 유전자의 인덱스 (다음에 기록될 유전자의 위치) | | |
| map<int, pair<int, int>> population[Population\_Size];// 세대 | | |
| map<int, pair<int, int>> temp\_population[Population\_Size];// 임시 세대 (연산을 위해 사용) | | |
| map<int, char> geneTypes[Population\_Size]; // 유전자의 타입 ( 'G': group, 'N': node) | | |
| double fitness[Population\_Size]; // 세대 평가 결과 | | |
| int bestIndi; // the best individual | | |
| struct \_tagKNode\* root; // 키 트리의 root | | |
|  | | |
| // 난수 생성 관련  RAND gen; // 초기 세대 생성에 사용  //int seed; | | |
| RAND gen2; // GA-select() 함수 - tournament selection에 사용  //int seed2; | | |
| RAND gen3; // GA-crossover() 함수 - crossing point 결정에 사용  //int seed3; | | |
| RAND gen4; // GA-mutate() 함수 - mutation point 결정에 사용  // int seed4; | | |
| RAND gen5; // GA-mutate() 함수 - mutation 발생 여부 결정에 사용  // int seed5; | | |

<A\* 알고리즘 관련>

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| std::list<SrchNode\*> findSolution(); // 문제의 해 발견 (i, i-th degree) | Kmr |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
| SrchGrph\* graph; // 탐색 그래프 | | |
| SrchNode\* solution; // 문제의 해 | | |
|  | | |
|  | | |

SrchGrph class: 휴리스틱 탐색을 위한 탐색 그래프 클래스

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| SrchNode\* doSearch(); // // 초기 상태에서 목적지 노드를 향한 탐색 시작 | AStar |  |
| std::list<SrchNode\*> getPath(SrchNode\* n); // 주어진 상태로부터 초기 상태까지의 경로 반환 | SrchGrph |  |
| // 멤버 변수 값 설정 및 반환 |  |  |
| void setRoot (SrchNode\* root); |  |  |
| SrchNode\* getRoot (); |  |  |
|  |  |  |
| // 탐색에 필요한 내부 연산 |  |  |
| void expand(SrchNode\* n); // expand the given node |  |  |
| void updateCost (SrchNode\* newParent, SrchNode\* current); // update the cost of current node and (possibly) change its parent node |  |  |
| void updateCostRec (SrchNode\* newParent, SrchNode\* n); // (Recuresive version for its descendants) update the cost of current node |  |  |
| bool isInOpenLst (SrchNode\* s); // 노드 s가 open list에 존재하는 지 여부 반환 |  |  |
| bool isInClsdLst (SrchNode\* s); // 노드 s가 closed list에 존재하는 지 여부 반환 |  |  |
| void sortOpenLst(); // OPEN 목록의 노드들을 f 값을 기준으로 오름차순 정렬 |  |  |
|  |  |  |
| // 테스트 함수 |  |  |
| void printPath(SrchNode\* n); // 현재 노드에서 root 노드까지의 경로를 출력 |  |  |
| void prntOpenLst(); // open list 내용을 화면에 출력 |  |  |
| void prntClsdLst(); // close list 내용을 화면에 출력 |  |  |
| 변수 | | |
| std::list<SrchNode\*> solutionPath; // 초기 상태에서 최종 상태까지의 경로 | | |
| SrchNode\* root; | | |
| list<SrchNode\*> openList; // open node list (nodes not yet expanded) | | |
| list<SrchNode\*> closedList; // closed node list (nodes already expanded) | | |

SrchNode class: 휴리스틱 탐색을 위한 탐색 노드 클래스

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // 탐색 노드의 비용 관련 함수 |  |  |
| double getG() const; | SrchGrph |  |
| double getH() const; | SrchGrph |  |
| double getF() const; | SrchGrph |  |
| void setG(); | SrchGrph |  |
| void setG( double g); | SrchGrph |  |
| **virtual void setH() = 0; // pure virtual function (problem-dependent function)** | SrchGrph |  |
| void setF(); | SrchGrph |  |
| // 탐색 관련 함수 |  |  |
| SrchNode\* getParent(); | SrchGrph |  |
| void setParent (SrchNode\* newParent); | SrchGrph |  |
| **virtual double getLinkCost (SrchNode\* parent) = 0; // get the link cost between the parent and this node (pure virtual function)** | SrchGrph |  |
| void addChild (SrchNode\* s); // add child of the current node | SrchGrph |  |
| int getNumChld(); | SrchGrph |  |
| list<SrchNode\*>::iterator getChildren(); | SrchGrph |  |
| bool isAncestorOf (SrchNode\* n); // return true if the current node is an ancestor of the given node n | SrchGrph |  |
| **virtual bool isSameState( SrchNode\* n) =0; // return true if this node has the same state as node n** | SrchGrph |  |
|  |  |  |
| // 다음 상태 계산 관련 함수 |  |  |
| **virtual void calNxtStates() = 0; // calculate possible next states (pure virtual function)** | SrchGrph |  |
| SrchNode\*\* getNxtStates(); // return iterator for possible next states list | SrchGrph |  |
| int getNumNxtSts(); // return number of possible next states | SrchGrph |  |
|  |  |  |
| // 종료 조건 검사 |  |  |
| **virtual bool isGoalNode() = 0; // return true if the current state is the goal state** |  |  |
| // 테스트 함수 |  |  |
| **virtual void prntStat() = 0; // print the current state** |  |  |
| 변수 | | |
|  | | |
| double g; // lowest cost found so far from node n0 to this node | | |
| double h; // est. cost from this node to goal node | | |
| double f; // est. cost of 'minimal cost path' (from n0 to goal node) through the current node (i.e., g + h) | | |
| SrchNode\* parent; // the parent node of this node on the least cost path found so far from n0 to this node | | |
| list<SrchNode\*> children; // child nodes of this node on the search graph | | |
| SrchNode\*\* nxtStates; // possible next states list | | |
| int numNxtSts; // number of possible next states | | |

SrchNode class: A\* 탐색 노드 클래스

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // 탐색 노드의 비용 관련 함수 |  |  |
| double getG() const; |  |  |
| double getH() const; |  |  |
| double getF() const; |  |  |
| void setG(); |  |  |
| void setG( double g); |  |  |
| virtual void setH() = 0; // pure virtual function (problem-dependent function) |  |  |
| void setF(); |  |  |
| // 탐색 관련 함수 |  |  |
| SrchNode\* getParent(); |  |  |
| void setParent (SrchNode\* newParent); |  |  |
| virtual double getLinkCost (SrchNode\* parent) = 0; // get the link cost between the parent and this |  |  |
| node (pure virtual function) |  |  |
| void addChild (SrchNode\* s); // add child of the current node |  |  |
| int getNumChld(); |  |  |
| list<SrchNode\*>::iterator getChildren(); |  |  |
| bool isAncestorOf (SrchNode\* n); // return true if the current node is an ancestor of the given node n |  |  |
| virtual bool isSameState( SrchNode\* n) =0; // return true if this node has the same state as node n |  |  |
| // 다음 상태 계산 관련 함수 |  |  |
| virtual void calNxtStates() = 0; // calculate possible next states (pure virtual function |  |  |
| SrchNode\*\* getNxtStates(); // return iterator for possible next states list |  |  |
| int getNumNxtSts(); // return number of possible next states |  |  |
| // 종료 조건 검사 |  |  |
| virtual bool isGoalNode() = 0; // return true if the current state is the goal state |  |  |
| // 테스트 함수 |  |  |
| virtual void prntStat() = 0; // print the current state |  |  |
|  |  |  |
| 변수 | | |
|  | | |
| double g; // lowest cost found so far from node n0 to this node | | |
| double h; // est. cost from this node to goal node | | |
| double f; // est. cost of 'minimal cost path' (from n0 to goal node) through the current node (i.e., g + h) | | |
| SrchNode\* parent; // the parent node of this node on the least cost path found so far from n0 to this node | | |
| list<SrchNode\*> children; // child nodes of this node on the search graph | | |
| SrchNode\*\* nxtStates; // possible next states list | | |
| int numNxtSts; // number of possible next states | | |

SrchNode\_KTS: key tree structure (KTS) 구축을 위한 A\* 탐색 노드 클래스

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // 멤버 변수 설정 |  |  |
| void setNumNodes(int numNodes); |  |  |
| void setNumClusters(int numClusters); |  |  |
|  |  |  |
| // 휴리스틱 함수 (예상 비용) |  |  |
| void setH(); // 목적지 노드까지의 예상 비용 계산 |  |  |
| int log2(int n); // 정수 n에 대한 log (base =2) 함수 결과 출력 |  |  |
|  |  |  |
| // 탐색 함수 |  |  |
| double getLinkCost (SrchNode\* parent); // 부모 노드와의 연결 비용 계산 |  |  |
| bool isSameState( SrchNode\* n); // 다른 상태와 비교 |  |  |
|  |  |  |
| // 다음 상태 계산 |  |  |
| void calNxtStates (); // 가능한 다음 상태 목록 계산 |  |  |
|  |  |  |
| // 종료 조건 |  |  |
| bool isGoalNode(); // 목표 상태와 일치하면 true, 아니면 false 반환 |  |  |
|  |  |  |
| // test |  |  |
| void prntStat(); // 현재 상태 출력 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
| // 상태를 구성하는 변수 | | |
| int currentLevel; // 현재까지 구축된 KTS의 높이 (즉, 현재 탐색 노드의 수준- 0, 1, 2, ..., c ) | | |
| map <int, int> degreeSeq; // KTS 구축을 위한 각 level의 degree 열 | | |
| int numLeafs; // 현재까지 구축된 KTS의 leaf node의 수 | | |
|  | | |
| // 상태 구성에 필요한 변수 | | |
| int numNodes;// 현재 센서 노드 수 | | |
| int numClusters; // 요구되는 클러스터 수 | | |

<통계 관련>

EVAL class

|  |  |  |
| --- | --- | --- |
| 함수 | | |
|  | caller | callee |
| // 멤버 변수 설정 |  |  |
| void setNodes(NodesMap nodes); // 노드 목록 설정 | main |  |
| void setEvctdNodes(NodesMap evctdNodes); // 노드 목록 설정 | main |  |
| void setAtk(Atk\* atk); // 공격 기법 객체 설정 | main |  |
| void setSec(Sec\* sec); // 보안 기법 객체 설정 | main |  |
|  |  |  |
| // 성능 평가 |  |  |
| void evalSec(); // 보안성 평가 | main | Sec, Atk |
| void evalEnergy();// 에너지 효율성 평가 | main | Node |
| void evalLifetime(); // 네트워크 수명 평가 | main |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 변수 | | |
|  | | |
| NodesMap nodes; // 노드 목록 ( {node ID, node pointer) } | | |
| NodesMap evctdNodes; // 네트워크에서 제거된 노드들의 목록 | | |
| Atk\* atk; // 공격 기법 객체 | | |
| Sec\* sec; // 보안 기법 객체 | | |
| double avgEnrgCnpmtn; // 노드들의 평균 에너지 소모량 | | |
|  | | |
|  | | |

params.h

|  |
| --- |
|  |

Node class

|  |
| --- |
| <역할>  센서 네트워크를 구성하는 노드의 기본적인 속성과 기능 정의  특정 프로토콜에 종속되지 않는 기능만을 포함 |
| <변수>  int id;  int x;  int y;  double distance; // sink 노드까지의 거리  double eCnsmtn; // 노드의 에너지 소모량  int cmprmsd; // 노드가 훼손된 경우 1, 정상인 경우 0  Rtn\* rtn; // 라우팅 객체  Sec\* sec; // 보안 객체 |
| <함수>  void txMsg(struct \_tagEvntMsg\* evntMsg); // 데이터 메시지 송신 함수  int rxMsg(struct \_tagEvntMsg\* evntMsg, Node nodes[], Sec\* sec); // 데이터 메시지 수신 함수  (정상: 0, 공격탐지: 1반환)  void txMsg(struct \_tagKeyMsg\* km); // 키 메시지 송신 함수  int rxMsg(struct \_tagKeyMsg\* km, Node nodes[], Sec\* sec); // 키 메시지 수신 함수 |

Kmr class

|  |
| --- |
| <역할>  보안 프로토콜에서 사용되는 비밀 키, 인증 키 등의 각종 키 관리와 키 배포 메시지 관리 |
| <변수>  Rtn\* rtn; // 라우팅 객체  Node\* nodes; // 노드 정보  SecKey gkp[GKP\_SIZE]; // 전체 키 집합 (GLOBAL KEY POOL)  AuthKey aKeys[NUM\_NODES]; // 노드들의 인증 키 정보  SecKey sKeys[NUM\_NODES][SKeysPerNode]; // 노드들의 비밀 키 정보  AuthKey vKeys[NUM\_NODES][VKeysPerNode]; // 노드들의 검증 키 정보 (다른 노드의 인증 키)  KeyMsg keyMsg; // 키 배포 메시지  random\_gen gen; // 키 배포 메시지 구성을 위한 난수 발생기  unsigned int seed; // 난수 생성을 위한 seed |
| <함수>  void setNodes( Node nodes[]); // 노드 객체 포인터 배열 초기화 함수  void setRtn( Rtn\* rtn); // rtn 객체 주소 저장  // 정적 키 배포 관련  virtual void preDistKeys() = 0; // 사전 키 분배 함수  bool isStored(int nodeId, SecKey sKey); // 주어진 sKey가 nodeID에 해당하는 노드에 이미 저장되었는지 확인하여 결과 반환  // 키 동적 배포 관련  virtual void genKeyMsg(int clstrID) = 0; // 키 메시지 생성 함수  KeyMsg\* getKeyMsg(); // 키 메시지 반환 함수  // (2012.11.09) 추가  virtual int getNumDismtdKeys(Node\* candi) = 0; // 후보 노드가 획득할 수 있는 키 개수 반환  int getNextAK(int kVal); // 기존 인증 키로부터 새로운 인증 키 계산 (OHC 상에서의 이전 노드)  // test 함수  virtual void testKeyInfo() = 0; // 키 정보 파일에 출력 (test 목적)  void testKeyMsg(int clstrID); // 키 배포 메시지 테스트 (파일에 출력) |

Kmr\_DEF class

|  |
| --- |
| <변수>  // Kmr 클래스의 멤버 변수 상속 |
| <함수>  void preDistKeys();  void genKeyMsg(int clstrID); // 키 메시지 생성 함수  int getNumDismtdKeys( Node\* candi);  void testKeyInfo(); // 키 정보 출력 |

Kmr\_PRM class

|  |
| --- |
| <변수>  // Kmr 클래스의 멤버 변수 상속 |
| <함수>  void preDistKeys();  void genKeyMsg(int clstrID); // 키 메시지 생성 함수  int getNumDismtdKeys( Node\* candi);  void testKeyInfo(); // 키 정보 출력 |

Rtn class

|  |
| --- |
| <변수>  // 노드 정보  Node\* nodes; // 노드 정보 배열의 시작 주소  Node\* nbrs[NUM\_NODES][NBRS\_PER\_NODE]; // 이웃 노드 정보  Node\* clstr\_hds[NUM\_CHS]; // ch 정보  // 후보 노드 관련  Node\* candis[NUM\_NODES][CANDIS\_PER\_NODE]; // 후보 부모 노드 정보  int numCandis[NUM\_NODES];  double eval\_result[CANDIS\_PER\_NODE]; // 현재 노드의 후보 부모 노드들에 대한 평가 함수의 결과 값 저장  // 다음 전달 노드 관련  Node\* nxtNds[BRANCH\_FACTOR]; // 다음 전달 노드 (키 배포 시) 배열    // 키 관련  Kmr\* kmr; // 키 관리자 객체 (2012.11.09 추가)  // 난수 생성 관련  random\_gen gen1; // CH 선택을 위한 난수 생성기  unsigned int seed1; // CH 선택을 위한 seed |
| <함수>  // 클러스터 구성 관련  void setNodes( Node nodes[]); // 노드 객체 포인터 배열 초기화 함수  void findNbrs(); // 이웃 노드 탐색 함수  virtual void setCHs() = 0; // CH 선택 함수  Node\* getCH (int clstrID); // CH 획득 함수    // 다음 노드 선택 관련  void setKmr( Kmr\* kmr); // (2012.11.10 추가) kmr 객체 멤버변수 설정  virtual void setCandis() = 0; // 후보 노드 설정 함수  virtual Node\*\* selNext (struct \_tagKeyMsg\* keyMsg, Node\* cNode) = 0; // 키 배포 경로 상의 다음 노드 선택  virtual Node\* selNext (struct \_tagEvntMsg\* evntMsg, Node\* cNode) = 0; // 데이터 메시지 전달 경로 상의 다음 노드 선택  virtual double eval (struct \_tagKeyMsg\* keyMsg, Node\* candi) = 0; // 현재 후보 노드의 평가 함수 결과를 계산하여 반환  void resetNxtNds(); // 다음 노드 목록 초기화  // sorting 관련  void quicksort(Node\* candis[], int start, int finish);  int partition(Node\* candis[], int start, int finish);  // test 함수들  void testNbrs(); // 각 노드의 이웃 노드 정보 파일에 출력  void testCHs(); // 각 클러스터 CH 정보 파일에 출력  void testCandis(); // 후보 부모 노드 정보 테스트 |

Rtn\_GEA class

|  |
| --- |
| <변수>  // Rtn 클래스의 멤버 변수 상속 |
| <함수>  // 클러스터 구성 관련  void setCHs(); // CH 설정 함수  // 다음 노드 선택 관련  void setCandis(); // 후보 노드 설정 함수  Node\*\* selNext (KeyMsg\* keyMsg, Node\* cNode); // 키 배포 경로 상의 다음 노드 선택  Node\* selNext (EvntMsg\* evntMsg, Node\* cNode); // 데이터 메시지 전달 경로 상의 다음 노드 선택  double eval (KeyMsg\* keyMsg, Node\* candi); // 현재 후보 노드의 평가 함수 결과를 계산하여 반환  double eval (double dist, double eCnpmtn ); // eval 함수 오버로딩 ( 각 factor는 0~1 사이의 값으로 정규화) |

Rtn\_PRM class

|  |
| --- |
| <변수>  // Rtn 클래스의 멤버 변수 상속 |
| <함수>  // 클러스터 구성 관련  void setCHs(); // CH 설정 함수  // 다음 노드 선택 관련  void setCandis(); // 후보 노드 설정 함수  Node\*\* selNext (KeyMsg\* keyMsg, Node\* cNode); // 키 배포 경로 상의 다음 노드 선택  Node\* selNext (EvntMsg\* evntMsg, Node\* cNode); // 데이터 메시지 전달 경로 상의 다음 노드 선택  double eval (KeyMsg\* keyMsg, Node\* candi); // 현재 후보 노드의 평가 함수 결과를 계산하여 반환  double eval (double dismKeys, double dist, double eCnpmtn ); // (2012.11.11 추가) eval 함수 오버로딩 ( 각 factor는 0~1 사이의 값으로 정규화) |

Atk class

|  |
| --- |
| <변수>  int numAtk; // 공격 횟수  random\_gen gen1; // 훼손 노드 결정을 위한 난수 생성기  unsigned int seed1; // seed |
| <함수>  void forgeEvntMsg(EvntMsg\* msg); // 허위 보고서 위조  void initCompNodes (Node nodes[]); // 훼손 노드 초기화  int getNumAtk(); // 공격 발생 횟수 반환 |

Atk\_FRA class

|  |
| --- |
| <변수> |
| <함수>  void attack(eventRep\* rep, Node nodes[], Sec\* sec); // 데이터 전달 단계 공격 함수 |

Sec class

|  |
| --- |
| <변수>  // 객체 멤버 변수  map<int, Node\*> nodes; // 노드 목록 ( {node ID, node pointer) }  Rtn\* rtn; // 라우팅 객체  Kmr\* kmr; // 키 관리자 객체  struct \_tagCntrlMsg\* cntlMsg; // 제어 메시지  // 키 재분배 관련  map<int, struct \_tagKeyMsg\*> keyMsgs; // 키 재분배 메시지  // 공격 탐지 관련  bool atkDtctd; // 공격 탐지 여부 (예: 허위 보고서 공격)  int numDtctn; // // 공격 탐지 횟수 (예: 허위 보고서 공격)  int numVer; // 보고서 검증 횟수  // MAC 연산 관련  std::list<struct \_tagMAC\*> tmplst;// (2013.11.19) 임시 MAC 목록 (보고서 내 MAC 갱신 과정에서 사용) |
| <함수>  void setKmr( Kmr\* kmr); // 키 관리자 객체 주소 설정  virtual void setMAC (struct \_tagEvntMsg\* msg) = 0; // MAC 설정  virtual void setMAC (struct \_tagEvntMsg\* msg, int clstrID) = 0; // MAC 설정 (클러스터 기반)  virtual bool verMAC (struct \_tagEvntMsg\* msg) = 0;//MAC검증 (성공 시 true, 실패 시 false 반환)  virtual bool detect (struct \_tagEvntMsg\* msg) = 0;//공격검출(공격 검출 시 true, 아니면 false반환)    int getNumDtctn(); // 공격 탐지 횟수 반환 |

Sec\_PM class

|  |
| --- |
| <변수> |
| <함수>  void setMAC(struct \_tagEvntRep\* rep);  int verMAC(struct \_tagEvntRep\* rep); |

Sec\_DEF class

|  |
| --- |
| <변수> |
| <함수>  void setMAC(struct \_tagEvntRep\* rep);  int verMAC(struct \_tagEvntRep\* rep); |

EvntMsg 구조체

|  |
| --- |
| <변수>  int x;  int y;  int val;  Node\* CoS; // CoS 정보  Node\* sender; // 송신 노드 (매 홉마다 변경)  Node\* receiver; // 수신 노드 ( " ) |
|  |

SecKey 구조체

|  |
| --- |
| <변수>  int kID; // 키 ID  int kVal; // 키 값 |
|  |

AuthKey 구조체

|  |
| --- |
| <변수>  int nID; // 노드 ID  int seq; // 해쉬 체인에서의 순서 (1 -> 2 -> 3 -> … -> m)  int kVal; // 키 값 |
|  |

KeyMsg 구조체

|  |
| --- |
| <변수>  AuthKey keys[NODES\_PER\_CLUSTER]; // 배포 중인 인증 키  int sKeyID[NODES\_PER\_CLUSTER]; // 인증 키 암호화에 사용된 비밀키의 인덱스  Node\* sender;  Node\* receiver;  int numDsm; // 키 배포 횟수 (해당 클러스터의 인증 키가 배포된 노드 수) |
|  |