NS2 AODV 관련 클래스의 리팩토링

신경호*, 이민순*, 김준환*, 이병수*
*인천대학교 컴퓨터공학과
e-mail:skh426@incheon.ac.kr

Refactoring for Classes Related with NS2 AODV

Kyoung-Ho Shin*, Min-Soon Lee*, Jun-Hwan Kim*, Byoung-Soo Lee*
*Dept of Computer Engineering, University of Incheon

요 약

AODV 프로토콜은 최단경로 라우팅 기법, 에너지 효율, 빠른 지역 경로 복구 등을 위한 많은 알고리즘이 연구되어 왔다. AODV 프로토콜의 성능평가는 대부분 NS2 시뮬레이터를 통해 이루어지고 있으며, AODV의 성능개선을 위한 알고리즘의 적용은 NS2의 AODV 소스파일 변경을 통해 이루어진다. AODV 소스파일의 AODV 클래스가 아주 많은 역할을 하는 구조로 작성되어 있어 알고리즘의 적용이 쉽지 않고, 이로 인해 성능평가를 위한 시뮬레이션에 시간과 노력이 많이 들게 된다. 이를 위해 기존 소프트웨어를 재사용하고 유연성을 높일 수 있도록 소프트웨어의 리팩토링을 해주어야 한다. 리팩토링은 소프트웨어를 효율적이고 유지보수가 쉽도록 전환하는 과정이며, 소프트웨어의 재사용성을 높여주고 유연성을 제공해 줄 수 있다. 본 연구에서는 AODV의 성능 개선을 위한 알고리즘 적용이 보다 유연해지도록 AODV 관련 클래스들에 대해 리팩토링을 수행한다.

1. 서론

이동 Ad-hoc 네트워크에서 데이터 교환을 가능하게 하는 프로토콜로 2003년 RFC 3561 문서로 채택된 AODV(Ad-hoc On-demand Distance Vector) 라우팅 접근 방식은 최단 경로를 통한 라우팅 방식이다[1]. 현재의 AODV 라우팅 프로토콜에 관한 연구는 에너지 효율을 고려한 네트워크의 Lifetime의 향상, 링크 단절시 빠른 지역 경로 복구방법, 제어 메시지의 발생 횟수 감소 등, 성능향상을 위해 다양하게 이루어진다. 이러한 AODV 라우팅 프로토콜의 성능평가는 대부분 NS2 네트워크 시뮬레이터를 통해 이루어지고 있다.

NS2에서 제공되는 AODV 소스는 카네기멜론 대학에서 만들어졌으며, 새로운 알고리즘의 적용은 이 AODV의 파일 수정을 통해 이루어진다. 반영된 알고리즘은 NS2 시뮬레이터의 시뮬레이션 결과를 토대로 성능평가가 이루어진다. 그러나 AODV의 클래스 내에 많은 메소드와 데이터 필드로 인해 새로운 알고리즘의 반영이 쉽지 않다. 새로운 알고리즘의 반영은 기존 프로그램의 기능 추가, 변경, 삭제 등을 발생시킨다. 이러한 구조는 성능평가를 위한 AODV 알고리즘 반영에 많은 비용을 발생시킨다.

위와 같은 소프트웨어의 비효율적인 구조와 복잡함을 막기 위해 소프트웨어를 재작성해야 한다. 리팩토링은 소프트웨어의 재작성을 통해 소프트웨어의 재사용성과 유연성을 고양하고 유지보수의 비용을 절감하는데 그 목적이 있다. 또한 소프트웨어의 구조를 단순화하고 개선함과 동시에 이전의 행동에 변화가 없도록 보장해주어야 한다[3].

본 연구에서는 기존의 AODV보다 알고리즘 적용이 쉬운 클래스 구조로 개선하기 위하여, NS2 AODV 관련 클래스를 기능별로 분해, 합성하는 리팩토링 방법을 제안하고자 한다. AODV 클래스와 관련된 aodv_rt_entry, AODV_Precursor 등의 클래스들은 friend 클래스의 사용으로 인해 객체지향의 관점을 해치고, 코드의 가독성을 떨어뜨리는 등, 새로운 알고리즘 적용에 유연하지 못하게 설계 되어있다. Encapsulate Field는 클래스의 friend 관계를 제거하기 위한 리팩토링 기법이다. 또한 AODV 클래스가 너무 많은 기능을 하는 구조로 되어있어 Extract Class와 Move Method 등의 리팩토링 기법을 이용하여, 클래스를 기능별로 나눌 필요가 있다.

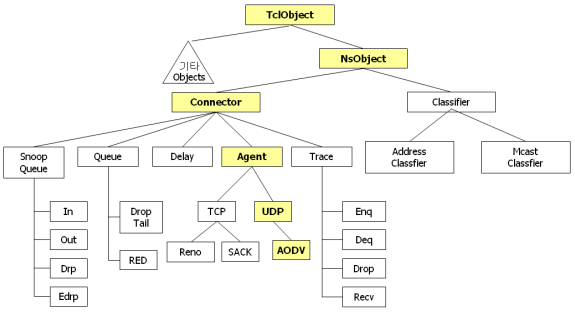
본 연구의 구성은 2장에서 NS2의 특징과 클래스의 계층구조, AODV 메커니즘, UML, 리팩토링과 디자인패턴에 대하여 살펴본다. 3장에서는 NS2 AODV 관련 클래스들의 클래스다이어그램을 통해 문제점을 살펴본다. 4장에서는 문제 해결을 위한 리팩토링이 적용된 클래스 설계를 제안하고, 5장에서는 결론과 향후과제에 대해 논의한다.

2. 관련 연구

2.1 NS2

NS2는 TCP, UDP, FTP, HTTP 등과 같은 TCP/IP 프로토콜 패밀리와 라우팅 프로토콜, 멀티캐스팅 프로토콜, SRN, RTP 등과 같은 다양한 인터넷 프로토콜은 지원한다. 또한 Ad-hoc 네트워크, 이동 통신망 기지국(Base Station), WLAN, Mobile-IP 관련 프로토콜도 지원하는

등 매우 광범위하게 사용되고 있는 네트워크 시뮬레이션 소프트웨어이다. NS2 시뮬레이터는 네트워크 컴포넌트가 거의 대부분을 차지하고 있으며, 컴포넌트에는 노드(Node), 링크(Link), 큐(Queue), 에이전트(Agent) 등이 포함되어있다. (그림 1)에 NS2 시뮬레이터에 있는 OTCL (Object Tool Command Language) 클래스 계층 구조(Class Hierarchical Structure)를 나타내었으며, 이 클래스 계층 구조에 있는 NsObject 부분에 네트워크 컴포넌트가 포함되어 있다[5].



(그림 1) NS2 클래스 계층 구조

2.2 AODV 메커니즘

AODV 라우팅 프로토콜은 1999년 C.Perkins에 의해 제안된 것으로 경로 획득 절차에 의해 얻어진 라우팅 정보를 일정시간 유지하는 방법이다. AODV 라우팅 프로토콜은 라우팅을 위해 RREQ, RREP, RERR 메시지를 사용한다. AODV 라우팅 프로토콜은 각 중간 노드에 라우팅 테이블에 대한 정보를 도착한 패킷의 목적지에 따라 다음 노드로 전송하는 방법이다[2]. 만약 소스노드가 이동하여 링크 손실이 발생하면, 소스는 경로탐색을 다시 시작한다. 중간노드가 이동하여 링크 손실이 발생하면, 손실된 링크의 업 스트림(up-stream) 노드가 RERR 메시지를 액티브 이웃노드(active neighbor)에 전파하여 모든 소스노드가 알게 된다. 여전히 목적지에 대한 경로가 필요하다면, 소스노드는 경로 탐색과정을 다시 시작하게 된다.

2.3 UML

UML(Unified Modeling Language)은 1996년 6월 Grady Booch의 Booch 방법론, Ivar Jacobson의 OOSE (Object Oriented Software Engineering) 방법론, James Rumbaugh의 OMT(Object Modeling Technique) 방법론을 통합하여 개발된 소프트웨어 모델링의 표준 언어이다[6]. 객체지향 분석과 설계를 위한 모델링 언어로써 표기하려는 대상을 다이어그램을 사용하여 나타내고 그 대상에 의미를 부여한다. 방법론이 아닌 방법론에 따른 설계에서 사용되는 언어이며 공식적인 표기법으로 사용된다. UML은 그래픽언어로써 소프트웨어 개발과정에서 나오는 산출물들을 가시화, 명세화, 구축, 문서화하는 역할을 수행할 뿐만 아니라 컴포넌트 기반 개발을 지원할 수 있도록 언어 확장 메커니즘인 스테레오 타입(Stereo types, <<>>기호)과 Tagged Value를 지원한다.

2.4 디자인 패턴과 리팩토링

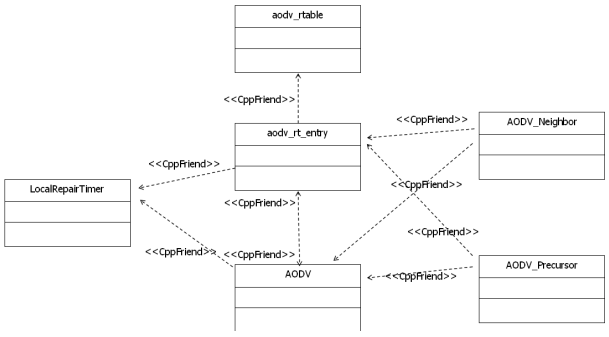
디자인 패턴은 특정 상황에서 발생되는 문제를 해결하기 위한 방법으로 사용되며, 소프트웨어의 구조를 명백하게 하고 구성요소들의 결합을 약하게 함으로써 소프트웨어의 유지보수 비용을 절감할 수 있게 한다. 현재 대중적인 활용성을 제공하는 대표적인 객체지향 디자인 패턴은 GoF 디자인 패턴이다[4]. 그 결과 특정 상황에서의 디자인 패턴 적용은 소프트웨어의 품질 향상과 보다 안정적이고 견고한 소프트웨어를 구축할 수 있게 한다.

TDD(Test Driven Development)는 점진적인 개발 접근 방법으로써 Kent Beck에 의해 소개되었다[8]. TDD에서는 새로운 코드를 추가하기 전에 먼저 테스트를 작성하고, 이러한 테스트를 바탕으로 프로그램을 작성하게 된다. 또한 과감한 리팩토링 과정을 통해 지속적으로 디자인을 정제하게 된다. Martin Fowler에 의해 제안된 리팩토링은 외부 동작을 바꾸지 않으면서 내부 구조를 개선하는 방법으로, 소프트웨어 시스템을 변경하는 프로세스이다.

리팩토링은 다음과 같은 특징을 가진다[7].

- 리팩토링은 소프트웨어의 디자인을 개선시킨다.
- 리팩토링은 소프트웨어를 더 이해하기 쉽게 만든다.
- 리팩토링은 버그를 찾도록 도와준다.
- 리팩토링은 프로그램을 빨리 작성하도록 도와준다.

3. NS2의 AODV 관련 클래스 구성

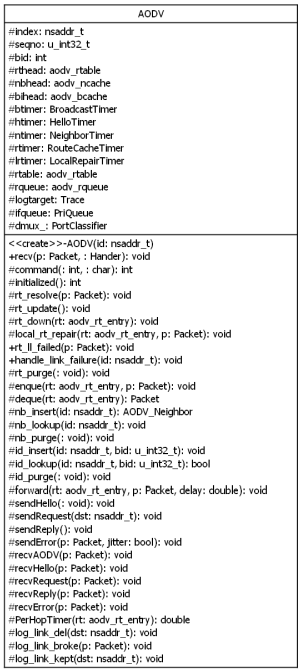


(그림 2) AODV와 관련된 클래스 다이어그램

AODV와 관련된 클래스의 구조는 AODV 클래스, LocalRepairTimer 클래스, aodv_rt_entry 클래스 등의 여러 클래스가 friend 관계로 얽혀있다. NS2의 AODV 부분을 StarUML을 이용하여 클래스 다이어그램을 만들면 (그림 2)와 같다. 이런 구조는 모듈의 응집성을 낮추고, 결합성을 높이게 되어 객체 지향적 관점에 벗어나게 된다. 또한 AODV 클래스는 (그림 3)에서와 같이 너무 많은 기능을 내포하는 문제점을 안고 있다. 이러한 설계는 과거의 설계 정보를 재사용(Reuse)하기 위해 설계 정보의 추상화

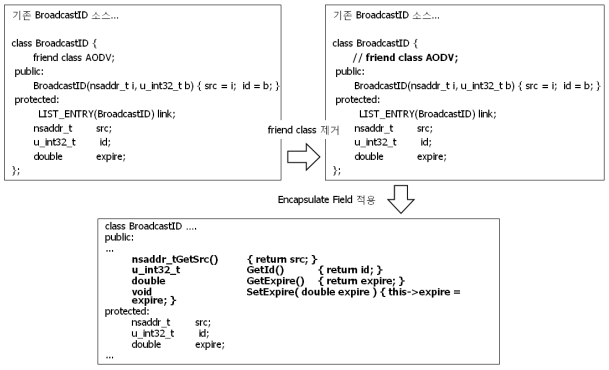
단계가 아닌 구체적인 코드를 읽어야 하므로 개발 환경에 의존적이며 설계 정보의 가독성(Readability)이 떨어지는 문제점을 갖고 있으므로, 리팩토링을 통해 클래스를 기능별로 분리할 필요가 있다[9].

AODV 클래스는 Martin Fowler가 제안한 Encapsulate Field 기법을 이용하여 friend 관계의 클래스들을 getXxx(), setXxx() 형태의 메소드를 통해 캡슐화 하여 friend 관계를 제거하고, Extract Class를 위한 Move Method, Move Field 리팩토링 기법을 이용하여 클래스를 기능별로 분리할 수 있다.



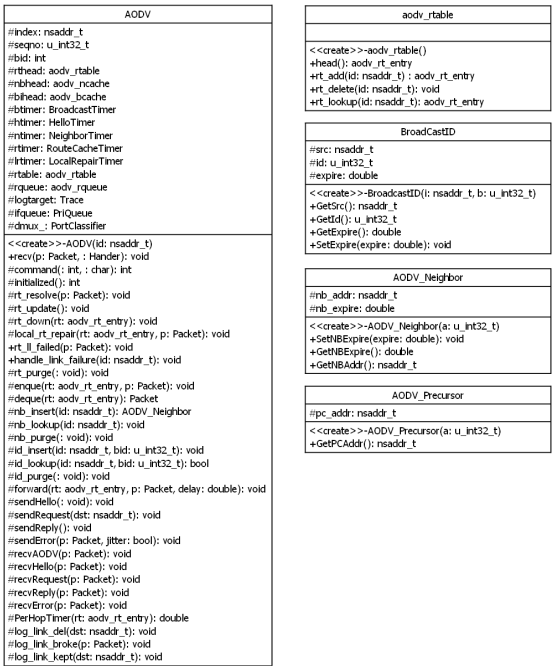
(그림 3) AODV 클래스의 클래스 다이어그램

4. AODV의 리팩토링



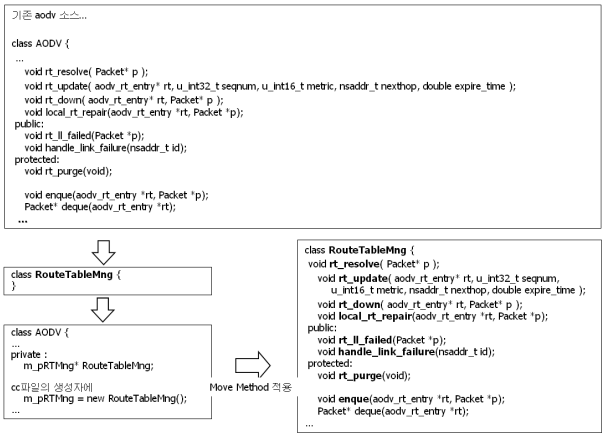
(그림 4) Encapsulate Field를 이용한 BroadcastID 클래스의 리팩토링

(그림 4)는 BroadcastID 클래스 내에 friend 관계에 있는 AODV 클래스의 friend 관계를 제거하기 위해 Encapsulate Field 과정을 보여주고 있다. Encapsulate Field 리팩토링 기법을 통해 AODV 클래스에서 임의로 BroadcastID 객체의 protected한 데이터 값에 접근하는 것을 막고, public한 GetSrc(), GetId(), GetExpire(), SetExpire() 메소드를 통해 데이터를 접근하도록 수정하였다. 이는 데이터와 동작이 분리됨을 뜻한다.



(그림 5) Encapsulate Field 리팩토링 기법이 적용된 AODV 클래스 다이어그램

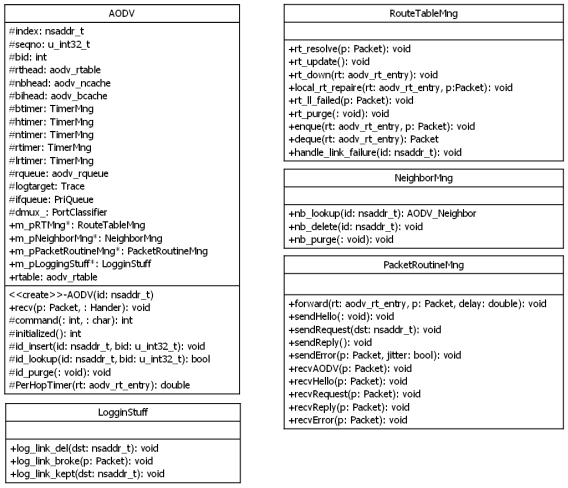
AODV_Neighbor 클래스, aodv_rtable 클래스, AODV_Precursor 클래스도 Encapsulate Field 리팩토링 기법을 적용하여 friend의 관계를 제거할 수 있다. (그림 5)는 Encapsulate Field 리팩토링 기법을 적용하여 완성된 클래스 다이어그램이다.



(그림 6) Extract Class를 이용한 AODV 클래스의 리팩토링

(그림 6)은 AODV 클래스 내에 라우팅 테이블의 정보와 관련된 부분을 Extract Class 리팩토링 기법을 적용하

여 RouteTableMng를 만드는 과정을 보여주고 있다. (그림 6)과 같은 Extract Class 리팩토링 기법을 통하여 Packet의 송·수신을 담당하는 PacketRoutineMng 클래스, 이웃노드의 발견 및 삭제와 관련된 NeighborMng 클래스, 링크의 상태와 관련된 LoggingStuff 클래스로 기능을 분리할 수 있으며, (그림 7)은 AODV 클래스의 Extract Class 리팩토링이 적용된 클래스 다이어그램을 나타낸다. (그림 7)과 같은 기능의 분리는 NS2의 AODV 라우팅 알고리즘에 새로운 알고리즘의 추가나 변경이 특정 클래스의 기능 내에 한정함으로서 코드의 재사용에서 더 유연함을 제공해 준다.



(그림 7) 리팩토링이 적용된 AODV 클래스 다이어그램

5. 결론

AODV 프로토콜의 성능평가는 대부분 NS2 시뮬레이터를 통해 이루어지고 있으며, AODV의 성능 개선을 위한 알고리즘의 적용은 AODV 소스파일의 변경을 통해 이루어진다. AODV 소스파일의 AODV 클래스는 다른 클래스들과 friend 관계로 얽혀 있고, 아주 많은 역할을 하는 구조로 되어 있어 알고리즘의 적용이 쉽지 않고, 이로 인해 성능평가를 위한 시뮬레이션에 시간과 노력이 많이 들게 된다. 이런 점을 감안할 때, 바람직하지 않은 friend 관계와 한 클래스 내에서 다양한 기능이 제공되는 AODV 구조의 리팩토링은 무엇보다 중요하다. 리팩토링은 AODV의 새로운 알고리즘을 적용함에 있어서, 코드의 가독성 및 재사용성을 높여 주는 중요한 수단이다.

앞으로 AODV의 구조 개선을 위해 추가적인 연구가 필요하다. 첫째, AODV 알고리즘은 객체의 타입에 따라 다른 동작을 선택하는 조건문을 가지고 있다. 이런 경우 Replace Conditional with Polymorphism 리팩토링 기법을 적용하여 AODV의 구조를 개선해야 한다. 둘째, 리팩토링 과정 중에 보다 유연한 구조가 될 수 있도록 GoF의 디자인 패턴을 추가적으로 적용하는 연구가 필요하다.

참고문헌

[1] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc On-Demand Distance Vector (AODV) Routing," RFC 3561, July 2003.

[2] C. E. Perkins and E. M. Royer, "Ad Hoc On-Demand Distance Vector Routing," Proceedings of IEEE Workshop on Mobile Computing System and Applications 1999, pp. 90-100, February 1999.

[3] Mel O Cinneide, "Automated Refactoring to Introduce Design Patterns," Proceedings of the 22nd International Conference on Software Engineering, pp. 722-724, 2000.

[4] E. Gamma, R. Helm, R.Johnson, and J.Vissides, "Design Patterns Elements of Reusable Object-Oriented Software," Addison-Wesley, 1995.

[5] The Networks Simulator ns-2, http://www.isi.edu/nsnam/, 2004.

[6] Grady Booch, James Rumbaugh, Ivar Jacobson, "The Unified Modeling Language user Guide," Addison Wesley, 1999.

[7] Martin Fowler, "Refactoring : Improving the Design of Existing Code," Addison-Wesley, 1999.

[8] Kent Beck, "Test Driven Development : By Example," Addison Wesley, 2002.

[9] Software Engineering : A practitioner's approach, McGraw-Hill International Edition, 2005.