# ns-2.31의 IEEE 802.11 모듈 버그

정낙천, 안종석
동국대학교 컴퓨터공학과
e-mail:{op1000, jahn}@dgu.edu

# Bugs on the IEEE 802.11 Module of ns-2.31

Nak-Cheon Jung, Jong-Suk Ahn
Dept of Computer Engineering, Dong-Guk University

요 약

본 논문은 공개 네트워크 시뮬레이터 ns-2.31(Network Simulator 2.31)의 802.11 DCF 모듈에서 버그(bug)를 소개하고 이의 영향 결과를 분석한다. ns의 802.11 DCF 모듈은 다음과 같은 문제점을 가지고 있다. 첫째, 백오프(backoff) 알고리즘은 표준안에서 명시한 알고리즘대로 작성되지 않았다. 둘째, 특정 조건에 해당되는 충돌에 대하여 트레이스 파일에 출력하지 않는다. 셋째, 전파 오류 모듈을 삽입하여도 전파 오류 결과를 트레이스 파일에 출력하지 않는다. 넷째, MAC(Medium Access Control) 알고리즘만을 평가할 수 있는 기법을 제공하지 않는다. 이러한 문제점을 수정한 ns-2와 수정전의 ns-2와 평균 4.6%의 충돌률 차이를 보인다.

## 1. 서론

ns-2[1]는 네트워크 연구자에 많이 사용되는 시뮬레이터이다. 그러나 ns에 구현된 802.11의 DCF 모듈은 버그를 비롯한 몇 가지 문제가 있다. ns의 구체적인 문제점은 다음과 같다.

첫 번째, 시뮬레이터는 알고리즘의 성능 검증을 위해 만들어진 소프트웨어이므로 표준안[2]에서 명시한 것과 같은 의미로 작성되어 있어야 한다. 그러나 ns의 백오프(backoff) 알고리즘은 표준안에서 명시한 알고리즘대로 작성되지 않았다. 이를 해결하기 위해서 802.11 표준안에서 제시된 알고리즘에 맞게 코드를 수정하였다.

두 번째, 시뮬레이터는 예외적인 이벤트를 출력하여 실험자가 혼동하지 않도록 해야 한다. 그러나 ns는 충돌의 특별한 상황에 대하여 무시하여 시뮬레이션 분석에 혼동을 주고 있다. 본 논문은 플래그(flag)를 정의하여 트레이스 파일에 출력하도록 하는 방안을 제안한다.

세 번째, 시뮬레이터는 환경설정에 맞는 결과를 출력해야 한다. 그러나 ns는 전파 오류 모듈을 삽입하여도 패킷의 오류 결과를 출력하지 않아 전파 오류를 고려한 실험을 수행할 때 어려움을 주고 있다. 본 논문은 전파 오류로 버려진 패킷을 출력하도록 하였다.

마지막으로, 범용 시뮬레이터는 MAC과 같은 단일 계층만을 테스트 할 수 있도록 설계되어야 한다. 그러나 ns는 ARP(Address Resolution Protocol)의 동작을 끄는 기능을 제공하지 않아 MAC의 성능 분석에 영향을 준다. 이를 위하여 [3]에서 제안하는 방안을 확장하였다.

패치를 적용하여 실험한 결과 백오프 알고리즘의 수정은 성능 척도 분석에 변화를 주지 않았다. 그러나 충돌과 오류 이벤트 출력 알고리즘은 평균 4.6%의 오차로 더 높은 충돌률을 보인다.

논문 구성은 다음과 같다. 2장에서 ns의 버그와 문제점을 기술하고 해결방법을 제안한다. 3장은 패치를 적용한 실험 결과를 보여준다. 그리고 4장에서 결론을 맺는다.

## 2. 버그 수정

### 2.1 백오프 알고리즘

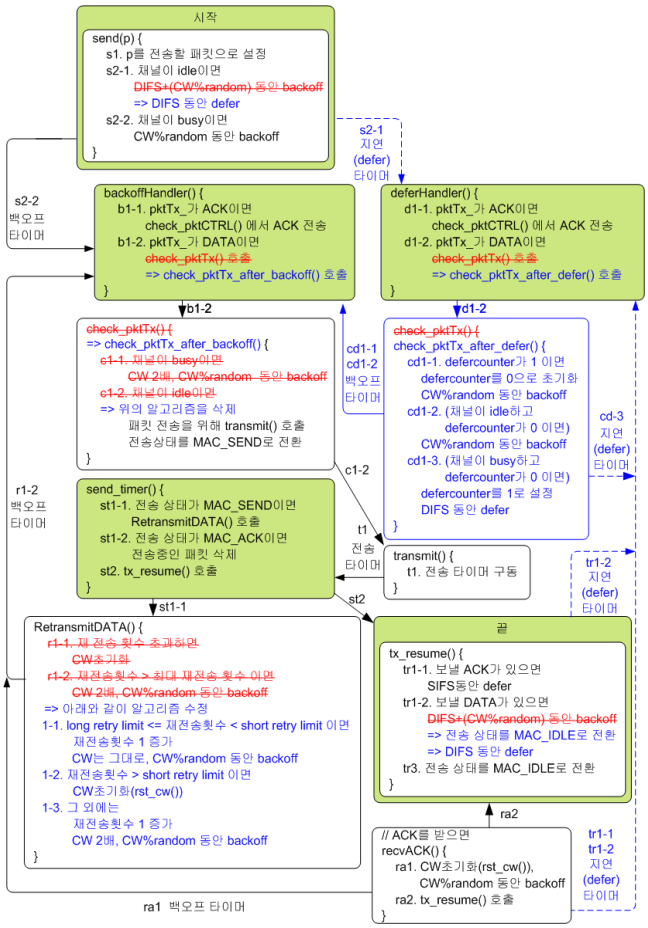
(그림 1)은 제안하는 ns의 백오프 알고리즘을 도식화한 것이다. 상자는 하나의 함수의 내용을 담고 있으며, 취소줄은 802.11의 표준 명세와 맞지 않는 부분이다. ns의 백오프 알고리즘은 *send()*부터 시작하여 *tx_resume()*에서 끝난다. 회색 바탕의 상자는 타이머가 만료되면 호출되는 함수이다. 즉, *backoffHandler()*는 백오프 타이머 만료 시, *deferHandler()*는 지연 타이머 만료 시, *send_timer()*는 전송 타이머 만료시 호출된다. *CW*는 경쟁 윈도우(Contention Window), *random*은 랜덤 함수로 생성된 임의의 수이다. (그림 1)에서 수정될 코드의 알고리즘 부분을 단일 취소선으로 나타내었으며, 이를 대체하는 제안하는 알고리즘의 시작을 화살표('=>')로 표시하였다.

#### 2.1.1 DIFS 지연과 백오프의 혼용

ns의 802.11 모듈은 DIFS 지연과 백오프 지연을 같은 기법으로 가정하여 사용한다. 그러나 이 두 가지의 지연은 서로 분리되어 구현되어야 한다. DIFS는 일정 시간동안 대기하는 기법이며, 백오프는 채널이 사용 중 일 때, 백오프 타이머를 중지시키는 기법이다. 그러나 ns는 백오프와 DIFS를 혼용하여 사용하였다. 이러한 오류는 (그림 1)의 *send()*의 s2-1과 *tx_resume()*의 tr1-2에서 확인할 수 있다.

이 문제를 해결하기 위해서는 (그림 1)의 s2-1과 tr1-2 부분을 다음과 같이 고쳐야한다. 첫째, 지연 타이머를 이용하여 DIFS동안만 대기하도록 한다. 둘째, DIFS와 백오프 후에 다르게 동작하도록 해야 한다. 전자의 해결은 *send()*와 *tx_resume()*의 수정으로 가능하다. 후자는

*backoffHandler()*가 *check_ptkTx()*를 호출하는 대신 *check_pktTx_after_backoff()*를 호출하도록 하였으며, *deferHandler()*는 *check_ptkTx()* 호출하는 대신 *check_pktTx_after_defer()*를 호출하도록 하였다.




(그림 1) 수정된 ns의 백오프 알고리즘

### 2.1.2 DIFS 지연 이후의 동작

표준안 [2]의 9.2.3절과 9.2.5절의 명세에 의하면 스테이션은 DIFS 지연 이후에 바로 채널을 감지할 수 있다. 채널이 사용 중이면 채널이 사용되지 않을 때까지 기다렸다가 DIFS 동안 한 번 더 대기하고 백오프 지연을 수행한다. 즉, 첫 번째 DIFS 지연 이후에는 채널의 사용여부를 검사하고 두 번째 DIFS 지연 이후에는 채널의 사용여부를 검사하지 않는다. 그러나 ns에서는 이에 대한 구현 부분을 찾아 볼 수 없다.

DIFS의 구현은 (그림 1)의 *check_pktTx_after_defer()*에서 찾을 수 있다. *defercounter*라는 변수를 새로 정의하여 이전에 DIFS가 수행 되었는지 확인할 수 있도록 하였으며, 채널을 검사하는 코드와 *defercounter*의 조합으로 조건문을 만들어 구현하였다.

### 2.1.3 백오프 지연 이후의 동작

표준안 [2]의 9.2.5.2절의 명세에 의하면 백오프 타이머가 0에 도달하면 스테이션은 패킷 전송을 바로 시작한다. 그러나 (그림 1)의 *check_pktTx()*에서 보듯이 ns의 구현은 백오프 타이머가 만료한 후 채널의 상태를 감지하고 그에 따라 다른 동작을 한다.

이에 대한 해결은 *check_pktTx_after_backoff()*에서 확인할 수 있다. 표준안에 맞도록 채널 감지부분을 삭제하고 패킷을 전송하는 부분만을 남겨놓으면 된다.

### 2.1.4 백오프 지연 시간

표준안 [2]의 9.2.4절의 명세에 의하면 백오프 지연 시간의 기준이 되는 *CW*를 계산하는 방법은 다음과 같다. 첫째, 재전송 횟수가 *long retry limit* 보다 작으면 *CW*를 두 배 늘린다. 둘째, 재전송 횟수가 *long retry limit*와 *short retry limit*사이에 있으면 *CW*를 늘리지 않는다. 셋째, 재전송 횟수가 *short retry limit*보다 크면 전송을 포기하고 *CW*를 초기화한다. 그러나 ns의 구현은 표준안과 전혀 다르다. 이에 대한 수정은 (그림 1)의 *RetransmitDATA()*에서 확인할 수 있다.

## 2.2 오류 이벤트 비출력

802.11 무선랜에서 발생할 수 있는 오류 이벤트는 충돌, 전파오류, 신호 감쇠로 인한 패킷 무시(Capture)가 있다. 이 상황들은 수신 스테이션이 패킷을 수신 할 수 없는 오류 상황이다. 즉, 데이터 수신 스테이션이 ACK 패킷을 전송하지 않는 상태이다. 시뮬레이터는 트레이스의 분석을 원활하게 하기 위해서 ACK 패킷이 전송되지 않은 이유를 출력해야 한다. 그렇지만 ns는 일부의 이벤트만 충돌상황을 출력한다.

### 2.2.1 충돌 이벤트 비출력

802.11 무선 랜에서는 다음과 같은 경우를 충돌로 판단한다. 첫째, 두 개 이상의 스테이션이 패킷을 동시에 전송할 경우. 둘째, 패킷을 전송중인 스테이션에 다른 스테이션으로 부터 패킷을 받을 경우. 셋째, 패킷을 다른 스테이션으로부터 패킷을 받고 있을 때 자신이 패킷을 전송한 경우이다. 저자는 두 번째와 세 번째 상황을 '전송 모드에서의 수신 패킷 무시(TGN:TX mode ignore)'로 부르겠다.

```
inline void Mac802_11::transmit(Packet *p, double timeout) {
    if(rx_state_ != MAC_IDLE) { ch->error() = 1; }}

void Mac802_11::recv(Packet *p, Handler *h) {
    if(tx_active_ && hdr->error() == 0) { hdr->error() = 1; }}

void Mac802_11::recv_timer() {
    if(tx_active_) { Packet::free(pktRx_); }}
```

[코드 1] 충돌 이벤트를 무시한 ns의 코드

ns는 전송 모드에서의 수신 패킷 무시 상황을 무시한다. 그 결과 트레이스 파일에 충돌 여부가 출력되지 않는다. 이를 입증하는 코드는 [코드 1]에 있다. 또한, [코드 1]의 함수에서 사용하는 *error()*는 *Packet* 클래스의 메서드로 패킷에 전파 오류가 발생했을 경우 1로 설정하도록 설계된 것이다. 그러나 [코드 1]에서 보듯이 ns는 충돌 상황에서 *error()*를 사용하여 설계 의도를 무시하고 있다.

### 2.2.2 패킷의 전파 오류 이벤트 비출력

전파 오류 이벤트는 errmodel.h와 cc 파일에 정의된 오

류 모델을 사용함으로써 발생시킬 수 있다. 오류 모델은 확률적으로 오류 모델을 계산하여 Packet 클래스에 정의된 *error()* 메서드를 사용하여 *error_* 필드에 1을 대입한다. 그러나 ns는 [코드 2]에서 보는 바와 같이 *discard()*에서 *error_*에 1이 대입된 패킷을 삭제하여 트레이스 파일에이벤트를 출력하지 않는다.

```
void Mac802_11::discard(Packet *p, const char* why) {
    if(ch->error() != 0) { Packet::free(p); }}
```

[코드 2] 전파 오류 이벤트를 무시한 ns의 코드

### 2.2.3 신호 감쇠로 인한 패킷 무시 이벤트 비출력

신호 감쇠로 인한 패킷 무시 현상은 캡쳐(Capture) 현상이라 불리며 [4]에서 보고된 바 있다. ns에서도 [4]의 제안한 방법을 적용하여 캡쳐 현상을 구현하였다. 그러나 [코드 3]의 *capture()*에서 보듯이 트레이스 파일에 출력하지 않는다.

```
void Mac802_11::capture(Packet *p) { Packet::free(p); }
```

[코드 3] 캡쳐 현상을 무시한 ns의 코드

### 2.2.4 오류 이벤트 비출력 현상 해결

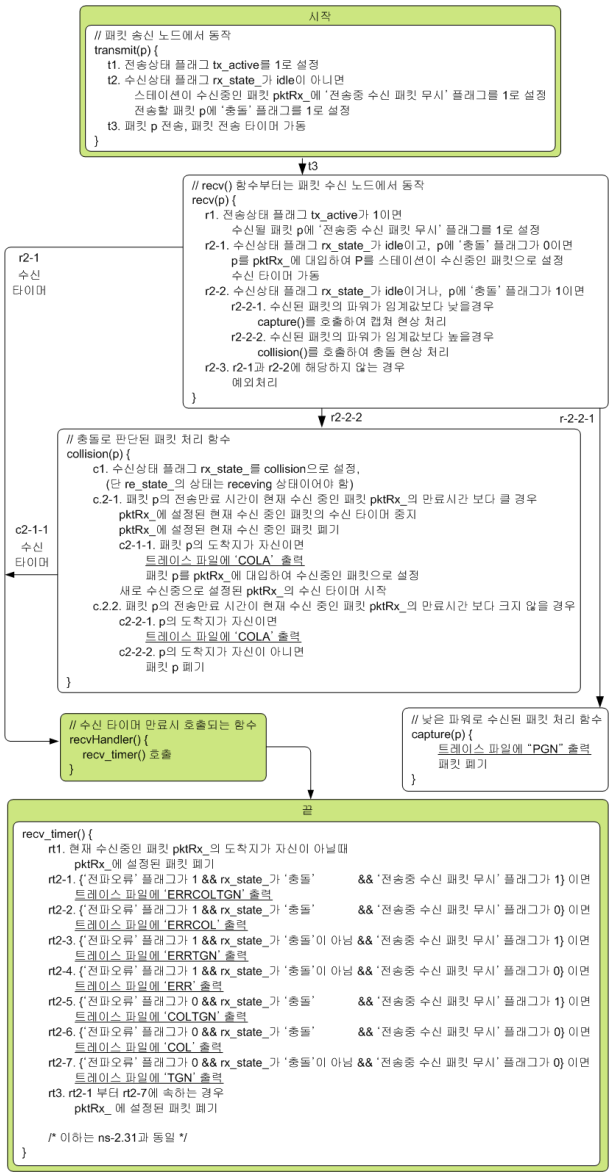
현재까지의 오류 이벤트 비출력 현상 분석으로 다음과 같은 결론을 낼 수 있다. ns의 구현은 충돌, 오류, 캡쳐 현상이 섞여 부정확한 성능 측정을 할 수 밖에 없다. 본 논문은 다음과 같이 해결방법을 제시한다.

<표 1> 패킷에 발생 가능한 오류 이벤트의 조합

| 전파오류 (ERR) | 충돌 (COL) | 전송 모드에서의 수신 패킷 무시 (TGN) | 트레이스 파일에 출력되는 문자열 |
|---|---|---|---|
| 1 | 1 | 1 | ERRCOLTGN |
| 1 | 1 | 0 | ERRCOL |
| 1 | 0 | 1 | ERRTGN |
| 1 | 0 | 0 | ERR |
| 0 | 1 | 1 | COLTGN |
| 0 | 1 | 0 | COL |
| 0 | 0 | 1 | TGN |
| 0 | 0 | 0 | 문자열 없음 |

첫째, 충돌, 전송 모드 패킷 무시, 오류 이벤트가 하나의 패킷에 동시에 발생할 수 있다는 점에 착안하였다. 전송된 패킷은 <표 1>과 같이 8개의 상태를 가질 수 있다. 패킷의 상태에 따라서 <표 1>의 오른쪽에 명시한 문자열이 트레이스 파일이 출력된다. 세부적인 오류 이벤트 출력하는 것이 분석 시 혼동을 주지 않다.

둘째, 이벤트 출력은 수신 스테이션에서만 처리한다. (그림 2)에서 보듯이 *recv_timer()*를 제외한 함수들은 오류 이벤트 발견 시 해당하는 이벤트 플래그를 1로 대입한다. 그리고 *recv_timer()*에서 이벤트를 출력한다.




(그림 5) 제안하는 오류 이벤트 출력 알고리즘

(그림 2)는 제안하는 오류 이벤트 검사 알고리즘이다. *transmit()*은 송신 스테이션에서 호출되며 나머지 함수들은 수신 스테이션에서 동작한다. *recv()*는 수신 스테이션에 패킷의 첫 비트가 도착했을 때, *recv_timer()*는 패킷 전체가 수신스테이션에 도착했을 때 호출된다. 밑줄은 이벤트 내용을 출력하는 부분을 표시한 것이다.

모아서 출력하는 알고리즘의 예외는 다음과 같다. 첫째, 캡쳐 현상을 처리하는 *capture()*는 첫 비트 수신시 신호의 강도를 알 수 있으므로 바로 이벤트를 출력한다. 둘째, *collision()*에서 스테이션은 하나의 패킷만을 수신할 수 있으므로 수신지가 같은 두 개 이상의 패킷이 수신된 경우는 하나만을 남기고 다른 패킷은 삭제해야 한다.

## 2.3 MAC 알고리즘의 성능만을 측정하기 위한 기법

ns의 ARP 프로토콜을 비 활성화하는 코드를 삽입하는 기법[3]은 MAC 연구자들 사이에서 알려진 기술이다. [코드 4]는 [3]에서 제안한 기법이다.

```
int ARPTable::arpresolve(nsaddr_t dst, Packet *p, LL *ll) {
    hdr_cmn *ch = HDR_CMN(p);
    mac_->hdr_dst( (char*) HDR_MAC(p), ch->next_hop() );
    return 0; }
```

[코드 4] [3]의 ARP 기능 끄는 코드

[코드 4]는 라우팅 프로토콜을 사용하지 않도록 *DumbAgent* 클래스를 같이 사용할 경우에 시뮬레이션 수행 시 오류를 발생한다. 이를 해결하려면 [코드 5]와 같이 확장해야 한다. [코드 5]는 ARP 테이블에 목적지에 대한 엔트리(entry)가 있는지 확인하고 없으면 엔트리를 추가한다. *ARPNextHop*은 tcl 스크립트에서 *true*이면 ARP를 끄고, *false*이면 ARP를 키도록 추가한 변수이다.

```
int ARPTable::arpresolve(nsaddr_t dst, Packet *p, LL *ll) {
    if (ARPNextHop_ == true) {
        ARPEntry *llinfo; llinfo = arplookup(dst);
        if(llinfo == 0) {
            llinfo = new ARPEntry(&arphead_, dst);
            llinfo->macaddr_ = dst; llinfo->up_ = 1;
            llinfo->count_ = 0; llinfo->hold_ = 0; }
        mac_->hdr_dst((char*) HDR_MAC(p), dst);
        return 0 ; }}
```

[코드 5] 제안하는 ARP 기능 끄는 코드

## 3. 실험 및 결과

### 3.1 실험 환경 설정

수정된 모듈을 검증하기 위한 환경설정은 다음과 같다. 33개의 스테이션은 UDP 데이터를 생성하며, ARP와 라우팅 프로토콜을 사용하지 않는다. 즉, 본 논문에서 제안한 방법으로 ARP를 비 활성화하였으며, *DumbAgent*를 사용하여 라우팅 프로토콜의 동작을 비 활성화하였다.

실험은 CBR(Constant Bit Rate) 트래픽을 생성하여 트래픽 부하(offered load)를 높여가면서 수행하였다. 패킷의 데이터 크기는 1000byte 이며, 패킷 전송 속도로 트래픽 부하를 조절하였다. 시뮬레이션은 300초 동안 10번 반복하였으며, 분석은 10번의 평균을 이용하였다. 대역폭은 2Mbps, 물리 계층은 DSSS (Direct-Sequence Spread Spectrum) 송신 기술, 그 외 802.11 프로토콜 인자들은 표준안에서 명시한 값을 사용하였다.

### 3.2 백오프 알고리즘 수정 검증

백오프 알고리즘의 영향만을 고려하기 위해서 세 개의 ns 프로그램을 사용하였다. ns-A에는 본 논문에서 제안하는 모든 패치를 적용하였으며, ns-B에는 백오프 패치만 제외, ns-C는 ARP 패치만을 적용한 것이다. 오류 모델의 영향을 배제하기 위해서 오류 모델을 삽입하지 않았다.

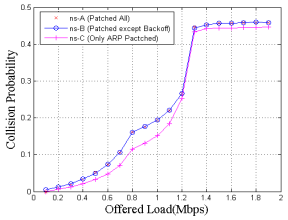
(그림 3) 은 세 개의 프로그램에 대해서 충돌률을 도식한 것이다. 분석은 트레이스 파일에 "COL"과 "TGN" 문자열이 보이면 충돌로 세었으며, 계산은 충돌 횟수를 패킷 전송 횟수로 나누었다.

(그림 3) 결과에서 보듯이 백오프 알고리즘의 수정은 충돌률 변화에 영향을 주지 않았다.

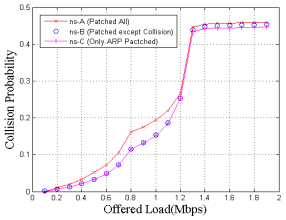
### 3.3 오류 이벤트 출력 알고리즘 검증

오류 이벤트의 검증 알고리즘의 영향만을 고려하기 위해서 세 개의 ns 프로그램을 사용하였다. ns-A와 ns-C는 3.2절에서 사용한 프로그램과 동일하며, ns-B에는 충돌률 검증 패치만 안한 프로그램을 사용하였다.

(그림 4)은 세 개의 프로그램에 대해서 충돌률을 도식한 것이다. (그림 4)의 결과에서 보듯이 수정된 알고리즘의 충돌률은 기존의 ns-2.31의 충돌률보다 높게 나왔으며 그 오차는 평균 4.6%이다. 트레이스 파일의 분석과 충돌률의 계산은 3.2절과 동일하다.




(그림 3) 백오프 수정 결과 분석 그래프




(그림 4) 오류 이벤트 수정 결과 분석 그래프

## 4. 결론

본 논문은 네트워크 시뮬레이터 ns-2.31의 버그를 보고하고 수정방안을 제안한다. 결론적으로 백오프 알고리즘의 경우 제안하는 수정방안은 기존의 백오프 알고리즘과 결과가 같아서 패치의 의미가 없었다. 그러나 ns-2.31의 오류 이벤트의 부정확한 출력 알고리즘은 실제로 오차를 발생시키고 있음을 확인하였으며, ARP를 끄고 키는 기법은 트레이스 파일의 분석 도구의 작성을 쉽게 하였다.

추후 연구로는 여러 가지 성능 척도를 계산하고 잘 알려진 수학모델과 비교하여 ns-2.31에서 잘못 구현된 알고리즘을 계속 찾아낼 것이다.

## 참고문헌

[1] "Network Simulator ns-2", http://www.isi.edu/nsnam/ns/

[2] ANSI/IEEE Std 802.11, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999

[3] "Nabble Free Forum for ns-2", http://www.nabble.com/Re%3A-AOMDV-for-ns-2.29-Available-p10379709.html

[4] A. Kochut, A. Vasan, A. U. Shankar, and A. Agrawala, "Sniffing out the correct Physical Layer Capture model in 802.11b", pp. 252-261, Proc. of ICNP'04, Oct. 2004