**I. Layer 2 (노순현)**

1. 진행상황

프로젝트 제안 슬라이드에 나온 서버 – AP – 노트북이 통신하는 시스템을 대상 시스템으로 가정할 때, AP는 사전에 생성한 L4 ACK을 서버로 보낼 시점을 결정해야 한다. 이를 위해서는 노트북에서 AP로 L2 ACK이 전송되는 시점을 파악하고, 그때마다 해당 L4 ACK을 보낼 조건이 충족되었는지(L4 ACK에 해당하는 L2 ACK이 도착하였는지)를 확인하여야 한다. 이를 위해서 금주까지 다음과 같은 작업을 수행하였다.

1. Ath9k driver의 interrupt service routine 분석

Ath9k driver의 핵심 코드는 drivers/net/wireless/ath/ath9k/main.c에 포함되어 있다(바닐라 리눅스 커널 기준). 기본적으로 Atheros 무선랜 카드는 하드웨어적으로 L2 ACK 관련 처리를 진행하지만, 처리를 할 때 interrupt를 발생시켜 Linux kernel에게 해당 TX frame이 잘 전송되었음을 확인시킨다.

Atheros 무선랜 카드가 interrupt를 발생시키면 ath\_isr() 함수가 수행된다. 이 함수는 ath9k\_hw\_getisr() 함수를 통해 interrupt가 발생한 원인을 status 변수에 저장하고, 그 원인에 따라 적절한 처리를 해준다.

우리가 실험할 머신에서는 ath9k\_hw\_getisr() 함수가 ar9003\_hw\_get\_isr() 함수로 매핑되어 있었다(사용하는 머신에 따라 ar9002\_hw\_get\_isr() 함수로 매핑될 수도 있음. 우리의 대상 AP에서 검증 실험이 필요). 해당 함수 내에서는 REG\_READ() 함수를 통해 무선 랜 카드의 AR\_INTR\_ASYNC\_CAUSE 레지스터의 값을 읽는다. 이 레지스터에는 interrupt의 발생 원인이 저장되어 있다.

참고 문헌이 없어서 실험을 통해 해당 레지스터 값을 확인해본 결과 L2 ACK을 정상적으로 받을 경우 해당 레지스터에서 AR\_ISR\_TXOK flag가 set 된다는 것을 추정할 수 있었다. 우리가 구현할 솔루션은 AR\_INTR\_ASYNC\_CAUSE 레지스터의 AR\_ISR\_TXOK flag가 켜졌는지를 확인하고, 만약 켜졌다면 TX frame이 잘 전송된 것이므로 L4 ACK을 전송할지 유무를 판단하면 된다.

1. Ath9k driver의 L2 frame의 sequence number 저장 데이터 구조 분석

앞에서 언급했듯이, L2 ACK이 전송되었다고 무턱대고 L4 ACK을 차례대로 전송하는 것은 잘못된 구현이다. 해당 L2 ACK이 어떤 frame에 대한 ACK인지를 판단하기 위해서는(duplicated ACK 유무를 판단하기 위해서는) TX frame의 sequence number를 확인하여야 한다.

Ath9k driver의 경우 TX의 sequence number가 ath\_tx\_status 구조체에 저장되어 있다. 해당 구조체에는 ts\_seqnum이라는 변수가 존재한다. L2 TX frame과 L2 ACK을 매칭시키는 과정에서는 해당 변수 값을 확인하면 된다.

1. 분석 결과 검증

기본적인 검증 실험은 Atheros 무선랜 카드가 설치된 데스크탑에서 진행하였다. Ath9k driver를 사용하는 Linux-3.10.49 기반 Ubuntu를 설치하였다. (Linux-3.10.49는 최종적으로 사용할 openwrt의 base Linux version임)

다음과 같은 검증을 진행하였다.

* ath\_isr() 함수의 호출 여부 확인 🡪 확인 됨
* ar9003\_hw\_get\_isr() 함수의 호출 여부 확인 🡪 확인 됨
* AR\_INTR\_ASYNC\_CAUSE 레지스터의 AR\_ISR\_TXOK flag가 ACK을 받을 때마다 켜지는지 확인 🡪 조교님께서 설치하신 아마존 서버에서 TCP packet을 받으면서 실험한 결과, TX/RX packet마다 AR\_ISR\_TXOK flag가 한 번씩 켜지는 것을 확인함. 커널 레벨의 체크 코드 아래 첨부. 추가적인 검증 필요.

|  |
| --- |
| static bool ar9003\_hw\_get\_isr(struct ath\_hw \*ah, enum ath9k\_int \*masked)  {  …  if (isr) {  …  if (isr & (AR\_ISR\_TXOK))  printk("COMNET::AR\_ISR\_TXOK flag is set\n");  …  }  } |

1. 구현계획

차주에는 다음과 같은 작업을 진행할 예정이다.

* AR\_ISR\_TXOK flag가 켜지는 시점이 L2 ACK을 받는 시점인지를 확실하게 검증
* 검증이 되었으면, 사전에 만들어둔 L4 ACK의 전송 조건을 판단하는 코드 작성
* 코드 작성이 완료되었으면 L4 쪽에 구현한 솔루션과 merge 시도
* 실험 및 검증

**II. Layer 4 (이선호, 이현민)**

1. 진행상황

앞서 진행하던 작업은 서버와 클라이언트의 직접 연결에 대해서 테스트 한 것이었기 때문에 이번에는 중간에 라우터를 넣고 실험을 진행하였다.

* 1. 라우터에서는 ip\_forward 함수를 virtual function인 dst\_input로 지정하여 사용할 것이라고 생각하고 이를 확인하기 위하여 ip\_forward.c의 ip\_forward(sk\_buff skb) 함수에서 메시지를 프린트 하도록 하였다.
  2. 들어온 패킷을 Layer 4 (TCP layer)로 올려주기 위하여 ip\_forward() 함수에서 바로 ip\_local\_deliver(sk\_buff skb) 함수를 호출하였고, 인자로는 ip\_forward의 인자로 들어온 skb를 전달하여 주었다.
  3. ip\_local\_deliver() 함수가 제대로 호출되는지 확인하기 위하여 함수에서 메시지를 프린트 하도록 하였다.

이와 같이 코드를 수정한 후 과제 슬라이드에 나온 대로 Kernel compile, Module compile, Insert/Remove module 과정을 실행하였다. 현재 프린트 하는데 약간의 문제가 있어 이를 수정하고 있는 중이다.

참고로 openwrt 소스에는 linux-3.10.49 소스가 두 군데 존재한다.

1. openwrt/build\_dir/target-mips\_34kc\_uClibc-0.9.33.2/linux-ar71xx\_generic/linux-3.10.49
2. openwrt/build\_dir/toolchain-mips\_34kc\_gcc-4.8-linaro\_uClibc-0.9.33.2/linux-3.10.49

모듈 컴파일 할 때target-mips\_34kc\_uClibc-0.9.33.2 하위 폴더의 파일을 쓰는 것으로 봐서 i)의 소스를 쓰는 것으로 보이지만 정확히 알지 못하여 i)과 ii)의 리눅스 커널 코드 모두를 수정한 후 사용하였다.

1. 구현 계획

Layer4에서 새로운 함수를 정의하는데 그 함수를 I\_ack이라 하자. 먼저 데이터 패킷이 들어오면 sk을 queue에 저장하고 그 패킷의 ack\_num을 list에 저장한다. 그리고 ACK이 온다면 ack\_num를 저장하고 있는 list와 ACK패킷의 seq\_num을 비교해서 같으면 client에서 보낸 TCP ACK이라 생각할 수 있으므로 이 ACK 패킷은 버리고 또한 ack\_num list에서 해당 값을 지운다. 그리고 ack\_num list에서 같은 값이 없으면 아무 행동도 하지 않고 그 패킷을 forwarding 시켜준다. Tcp\_ack은 tcp\_send\_ack(sock\* sk)을 시작으로 만들어 지므로 client에서 L2 ACK이 올 때 호출해준다.

다음 3가지 상황이 있을 때 Ap는 다음과 같은 과정을 거친다.

1. server에서 client에게 데이터를 보낼 때

원래대로라면 Ap의 route.c에서 virtual fuction인 dst\_input을 ip\_forward로 지정해서 client로 forwarding 해준다. 앞으로 구현할 사항은 Ip\_forward 함수가 호출 될 때 I\_ack을 호출해서 위에 언급한 것처럼 sk를 queue에 저장하고 ack\_num를 list에 저장한 후 이 패킷을 client로 forwarding 해준다.

1. 클라이언트에서 L2 ACK을 보낼 때

Ap에서 L2 ACK을 받으면 Interrupt가 걸리고 그 과정에서 layer4의 새로운 함수를 호출한다. 그 함수는 queue에서 sk를 하나 뺀 뒤에 tcp\_send\_ack(sk)을 호출한다. 그러면 Ap에서 server로 TCP ACK을 보내게 된다.

1. 클라이언트에서 TCP ACK을 보낼 때

원래대로라면 TCP ACK 패킷은 1번 상황의 데이터 패킷처럼 Ap가 server로 forwading해준다. 1번 상황에서 구현한대로 하면 이 패킷이 Ap로 들어오면 I\_ack을 호출하게 된다. TCP ACK의 seq\_num는 전에 보낸 데이터 패킷의 ack\_num와 같으므로 TCP ACK의 seq\_num와 ack\_num list중 값이 일치하는 것이 있을 것이다. 그러면 해당하는 ack\_num는 list에서 제거하고 TCP ACK 패킷도 보내면 안되므로 없애버린다.