Reliable udp

네트웍

2007.04.18. 12:04

---

1 개요

UDP는 명시적인 연결이 없고, 데이터그램 단위로 데이터를 전송하는 프로토콜이다. 흐름 제어 역시 없기 때문에 전송된 데이터가 순서대로, 반드시 도착하리라는 보장도 없다.

대신 명시적인 연결이 없기 때문에, 대등한 관계, 즉 두 피어가 동시에 시작이 되는 경우에는 유용할 수 있다. TCP 같은 경우에는 어느 한쪽이 반드시 패시브 모드로 리스닝을 하고 있어야 하기 때문에, 앞에 말한 것과 같은 경우에는 쓰기가 애매하다.

하지만 그대로 쓰기에는 흐름 제어가 없기 때문에 곤란하다. 그렇다고 이것저것 다 집어넣다보면 UDP를 가장한 TCP가 되어버린다. 그래서 생각한 것이 순서의 보장은 제외하고 반드시 도착한다는 것만 보장하자는 것이었다.

반드시 도착하는 것을 보장하기 위해서는 두 가지 방법이 있다.

1. 수신측에서 중간에 빠진 패킷을 송신측에게 요청한다.
2. 수신측에서 받은 패킷에 대한 정보를 송신측에게 알려준다.

말은 둘이 비슷해보이지만, 구현 방법은 상당히 달라진다. 첫번째는 에러가 있을 경우 뭔가를 요구하는 것이고, 두번째 방법은 에러가 없을 경우 이것을 알려주는 방법이다.

첫번째 방법 같은 경우, 수신측에서 요구를 해올지 안 해올지 알 수가 없기 때문에 패킷을 계속 가지고 있어야한다. 즉 송신측에서 1,2,3번 패킷을 차례대로 보냈다고 하자. 무사히 받았다고 날아오는 것이 없기 때문에 보낸 패킷을 가지고 있어야한다. 언제 이 패킷을 삭제해야하는가를 결정할 방법이 애매하다.

두번째 방법의 경우에는 아직 인증을 받지못한 패킷만 가지고 있으면 되기 때문에 메모리적으로는 부담이 덜 된다. 방식 자체도 이게 옳다고 생각했기 때문에 현재 라이브러리에서 선택한 방법은 두번째였다.

데이터그램의 길이 제한 역시 생각해 볼 문제다. 예전에 들은 것인데, 라우터를 거치는 데이터그램 중 1500 바이트가 넘어가는 것은 전송이 금지된다고 들은 적이 있다. 자세히 조사해보고는 싶지만 귀찮다. -_- 데이터그램의 길이가 제한이 되어버리면 제일 문제가 되는 것이, 제한 길이를 넘어가는 데이터그램의 존재다. 이런 데이터그램은 순서 보장을 통해서 차례대로 실행해야만 하기 때문이다. 근데 현재로서는 순서 보장 계획이 없다. 결국 하나의 데이터그램에 하나의 패킷만 넣고, 최대 길이를 넘으면 어서트시키는 방식으로 해결해놨는데, 최대 길이를 넘는 것이 나올 때에는...대책없다.

2 구현

일단 앞쪽에서도 언급했듯이 수신측에서 패킷을 받았다고 송신측에게 알려주는 방식을 채택하기로 한다. 이 방식을 채택했을 때 구현해야 할 것은...

1. 다른 피어에게 보낸 데이터그램의 리스트 관리 기능 – 인증을 받기 전에는 송신한 데이터그램을 삭제하지 말고, 가지고 있어야한다.
2. 수신측이 송신측에게 데이터그램을 무사히 받았다고 알려주는 기능
3. 무사히 받았다고 수신 측이 알려왔을 때, 송신측에서 해당하는 데이터그램을 삭제할 수 있는 기능
4. 데이터그램이 도착했을 때, 데이터그램이 이미 실행한 것인지의 여부를 판단할 수 있는 기능
5. 순서를 건너뛴 데이터그램이 도착했을 때, 빠진 데이터그램을 기억해두는 기능

이 모든 것이 각각의 피어마다 존재해야한다. TCP의 1:1 통신과는 달리 UDP는 하나의 소켓을 통해서 여러 피어와 통신을 하기 때문이다. 결국 아래와 같은 모양의 클래스가 하나 나온다.

```
class UDPPeerState
{
    // IP + PORT의 문자열 값으로서 피어를 구분하는 역할을 한다.
    string m_PeerKey;

    // 받아서 실행한 데이터그램 중 제일 값이 큰 것
    DatagramID_t LastID;

    // 데이터그램이 순서대로 오지 않고, 건너뛴 경우...
    // 즉 11번 데이터그램 다음에 13번 데이터그램이 왔다 같은 경우,
    // "12번이 빠졌다"라는 정보를 기억하기 위한 리스트
    list<Datagram> OmittedList;

    // 무사히 실행한 데이터그램의 리스트
    // 나중에 송신측에게 무사히 실행했다는 사실을 기록한 데이터그램을 날려줄 때 쓰인[
    list<Datagram> AckList;
}
```

2.1 데이터그램을 보낼 때 해야할 일

1. 데이터그램의 ID를 부여한다. 이는 보낼 때마다 +1씩 증가시키는 방법으로 처리하면 된다.
2. 데이터그램을 보낸다.
3. 데이터그램에 대한 인증을 받지 못했을 경우, 다음에 전송해야하는 시간을 기록해둔다.
4. 데이터그램에 대한 인증을 받지 못했을 경우, 전송을 시도할 최대 횟수를 기록해둔다.
5. 데이터그램을 보냈지만, 아직 인증받지 못한 데이터그램의 리스트에다가 집어넣어둔다.

2.2 데이터그램을 받았을 때 해야할 일

1. 데이터그램이 이미 한번 실행했던 데이터그램이 아닌지 검사한다. 이는 중간에 빠진 데이터그램의 리스트와 현재까지 도착한 데이터그램ID의 최대값을 이용해서 이루어진다.
2. 이미 한번 실행했던 것이라면 그냥 무시하고, 아니라면 정상적으로 실행한다.
3. 현재까지 도착한 데이터그램ID의 최대값과 중간에 빠진 데이터그램의 리스트를 갱신한다.

4. 데이터그램을 "정상적으로 실행했지만, 송신측에 알려주지 않은 리스트"에다가 등록한다.
5. 만일 데이터그램이 수신측이 보내온 인증 데이터그램이라면 보냈지만, 아직 인증받지 못한 데이터그램의 리스트 에 있는 해당하는 데이터그램들을 삭제해준다.

2.3 주기적으로 처리해야할 일

1. 정상적으로 실행했지만 , 송신측에 알려주지 않은 리스트에 데이터그램ID가 들어있다면 이를 송신측에게 송신한다. (이 데이터그램 역시 데이터그램을 보내는 것이므로, 데이터그램을 보낼 때 해야할 일들을 차례로 수행한다.)
2. 보냈지만, 아직 인증받지 못한 데이터그램의 리스트를 검색하면서 재전송해야하는 데이터그램이 있다면 재전송한다. (재전송할 때마다 다음에 전송해야하는 시간과 전송 시도 횟수를 갱신해야한다.)
3. 최대 재전송 횟수를 초과한 데이터그램은 삭제해버린다. (끝까지 전송되지 않는 데이터그램이 결국 생길 수 있다.)

2.4 빠진 데이터그램인지 확인하는 방법

일단 전제 조건으로 송신측에서 송신하는 데이터그램들은 순차적인 ID를 가져야한다. 즉 제일 처음에 보낸 데이터그램이 1번이었다면 다음으로 보내는 데이터그램은 2번, 3번, 4번 이런 식으로 처리되어야한다는 것이다. 전제 조건이 만족되었을 때 데이터그램의 도착 순서에 대한 시나리오는 다음과 같다. 현재 마지막으로 도착한 데이터그램이 ID가 9라고 하자. 이는 10번 데이터그램이 도착해야 한다는 말이다.

1. 10 11 12 : 모든 것이 순서대로 왔다. LastID는 매번 증가한다.
2. 11 10 12 : 먼저 LastID가 11로 증가한다. 9->11로변했으므로, 10의 값이 중간에 빠진 데이터그램 의 리스트에 추가된다. 다음으로 10이 오면 리스트에 존재하므로, 실행한다. 그리고 리스트에서 데이터를 삭제한다. 12는... 정상적으므로 실행한다.
3. 10 10 11 : 제일 처음 오는 10은 당연히 정상적으로 실행된다. 두번째로 오는 10은 LastID값에 의해서 무시된다. 11은 다시 정상적으로 실행된다.

이걸 다시 코드로 만들어보면 다음과 같은 모양이 된다.

```
bool UDPPeerState::validateDatagramID(DatagramID_t ID)
{
    // 만일 데이터그램의 ID가 마지막으로 실행한 데이터그램의 ID보다 작다면,
    // 혹시 이전에 빠진 데이터그램인지 검사해본다.
    if (ID < m_pImpl->LastID)
    {
        list<OMIT_INFO>::iterator itr = m_pImpl->OmitList.begin();
        for (; itr != m_pImpl->OmitList.end(); itr++)
        {
            if (ID == (*itr).ID)
            {
                // 이전에 빠진 데이터그램이라면 실행해야 한다.
                return true;
            }
        }
    }
    // 새로운 데이터그램이다.
    else if (ID > m_pImpl->LastID)
    {
        // 어쨌든 true다.
        return true;
    }

    // 이까지 왔다는 말은 데이터그램의 ID가 마지막으로 받은 데이터그램의
    // ID와 같다는 말이므로, 무시해야 한다.
```

```
        return false;
    }
```

3 예외 처리

3.1 PEER 중 어느 한쪽이 비정상적으로 종료된 경우

통신을 하고 있던 양쪽 중에서 어느 한쪽이 죽었다고 하자. 이런 경우 생기는 문제는 두 가지로 나눌 수 있다.

3.1.1 죽은 PEER가 상당시간 계속 죽어있는 경우

이 경우의 문제는 죽은 쪽에서 인증 패킷을 보내주지 않음으로 인해, 송신 측에서 패킷을 계속 재전송하는 데 있다. 이 문제는 재전송의 횟수에 제한을 줌으로서 해결할 수 있다. 물론 둘 다 살아있는데도 불구하고, 재전송 최대 횟수만큼의 송신이 실패할 우려(예를 들어 15번을 최대 재전송횟수로 잡았는데, 15번 모두 실패!)도 있겠지만, 확률은 상당히 낮다고 본다.

3.1.2 죽은 PEER가 다시 살아난 경우

이 경우 죽은 PEER는 보내는 데이터그램의 ID를 초기값부터 보내온다. 즉 수신 측에서 유지하고 있던 인증 정보가 완전히 쓸모없어져 버리는 것이다. 이런 경우 수신 측은 송신 측에 맞춰서 인증 관련 정보를 초기화해줘야한다. 이것을 위해서 송신측에서는 임의의 PEER에게 패킷을 보낼 경우, 해당하는 PEER에게 패킷을 보낸 적이 있는지 조사하고 보낸 적이 없다면, 수신 측에게 인증 관련 정보를 초기화하라는 패킷을 보내 준 후에 실제 패킷을 날려야한다.