HTTP over SCTP

<Guide to use one-to-one style interface with withsctp tool>

2005년 7월

김 상 태 (saintpaul1978@mail.knu.ac.kr)

경북대학교 컴퓨터과학과 통신프로토콜 연구실

요약

본 문서는 차세대 수송계층 프로토콜인 SCTP (Stream Control Transmission Protocol) 위에서 동작하는 HTTP 서버와 클라이언트의 사용 방법을 소개한다. SCTP는 소켓 API에서는 하나의 소켓으로 여러 Association을 처리하는 one-to-many interface를 제공하고 있지만 기존 TCP 코드를 수정해야 하는 단점이 있다. 따라서 이 문서에서는 기존의 TCP 코드를 거의 – 경우에 따라서는 전혀 – 손대지 않고 SCTP 응용으로 전환할 수 있는 one-to-one interface를 사용한 HTTP 서버/클라이언트의 사용법을 다룬다. 특히 리눅스 커널 2.6.10상에서 lksctp-tools에 포함되어있는 withsctp를 사용한 방법을 다룬다.

목차

1. SCTP 소켓 인터페이스 소개

이 절에서는 SCTP 소켓 인터페이스에 대해 간략히 알아본다. SCTP 소켓 API의 설계할 때 고려된 세 가지 목표가 기존 소켓 API와의 호환성 유지와 one-to-many 및 one-to-one 스타일 인터페이스 제공이다.

1.1. One-to-many 스타일 인터페이스

One-to-many 인터페이스는 UDP 소켓과 비슷하게 소켓 하나로 여러 SCTP Association를 처리할 수 있다. 이것을 사용하면 서버 프로세스가 프로세스나 쓰레드를 여러 개 생성하지 않아도 많은 클라이언트의 연결 지향형 서비스의 요청을 처리할 수 있게 된다.

One-to-many 인터페이스를 사용하기 위해서는 다음과 같은 방법으로 소켓을 생성한다.

- ✓ sd = **socket**(PF_INET, SOCK_SEQPACKET, IPPROTO_SCTP);
- ✓ sd = **socket**(PF_INET6, SOCK_SEQPACKET, IPPROTO_SCTP);

이렇게 생성된 소켓은 기존에 사용되는 모든 소켓 API와 새로 제공되는 SCTP 전용 API들을 one-to-many 스타일로 사용할 수 있다. 즉, listen과 bind를 거친 후 accept를 수행할 필요 없이 이 소켓만 가지고 여러 클라이언트와 통신을 할 수 있다. 따라서 서버 프로그램의 코드를 더 단순화 하면서, 프로세스의 오버헤드를 줄일 수 있게 된다.

하지만 one-to-many 스타일은 기존의 서버 응용의 코드를 수정해야 하는 단점을 가지고 있기 때문에 기존의 TCP응용을 one-to-many 스타일의 SCTP 응용으로 전환하는 데에는 부가적인 노력이 필요하다.

1.2. One-to-one 스타일 인터페이스

One-to-one 인터페이스는 TCP 소켓과 비슷한 모양으로 연결 지향형 서비스를 수행한다. 즉, 하나의 소켓이 하나의 SCTP Association 만을 제어할 수 있다. SCTP 소켓 API가 기존의 소켓 API들을 완벽히 지원하기 때문에 one-to-one 인터페이스를 사용하면 기존의 TCP 코드들을 거의 수정하지 않고도 SCTP로 전환이 가능하다.

One-to-one 인터페이스를 사용하기 위해서는 다음과 같은 방법으로 소켓을 생성한다.

- ✓ sd = **socket**(PF_INET, SOCK_STREAM, IPPROTO_SCTP);
- ✓ sd = **socket**(PF_INET6, SOCK_STREAM, IPPROTO_SCTP);

소켓 생성을 위와 같이 SCTP로 하는 것과 netinet/sctp.h를 include하는 것을 제외하면 기존의 TCP 응용은 컴파일만 다시 수행하여 SCTP로 전환할 수 있게 된다.

2. One-to-one 스타일 인터페이스를 사용한 HTTP over SCTP

이 절에서는 실제로 HTTP 서버와 클라이언트간의 통신을 SCTP를 사용하여 테스트한다. 먼저 테스트 환경 구축에 대해 간략히 알아보고, 실제로 SCTP를 사용한 HTTP서버와 클라이언트간의 통신을 수행해 본다. 또한 SCTP 통신이 원활이 되는지 이더리얼로 패킷을 캡쳐해 본다.

2.1. 테스트 환경 구축

테스트 환경을 구축은 SCTP 모듈 활성화, lksctp-tools 설치, HTTP 서버/클라이언트 설치의 세 단계로 이루어 지며 테스트 환경의 최종 모습은 다음과 같다.

- ✓ 서버 측
  OS: 레드햇 리눅스 9.0
  커널: 2.6.8.1 (SCTP Built-in)
  응용: Apache 1.3.33, lksctp-2.6.10-1.0.2
- ✓ 클라이언트 측
  OS: 데비안 리눅스 3.1
  커널: 2.6.10 (SCTP Built-in)
  응용: Mozilla 1.7.8, lksctp-2.6.10-1.0.2

2.1.1. SCTP 모듈 활성화

SCTP를 사용하기 위해서는 기본적으로 리눅스 커널 2.6 이상의 버전이 필요하다. 리눅스 커널을 새로 컴파일을 하여 SCTP 관련 모듈을 Built-in하거나 (이 경우 IPv6 모듈로 Built-in 해야 함) modprobe 명령으로 SCTP 모듈을 메모리에 적재하면 리눅스에서 SCTP를 사용할 수 있다. 다음 표 1 은 SCTP 모듈 Built-in을 위한 커널 옵션을 보여준다.

표 1. SCTP 모듈 Built-in 을 위한 커널 옵션

```
Device Drivers --->
[ ]   Networking support -->
              Networking options --->
              ... 중략 ...

<*> The IPv6 protocol (EXPER ... )
[*]   IPv6: Privacy Extensions ...
<*>           IPv6: AH transformation
<*>           IPv6: ESP transformation
<*>           IPv6: IPComp transformation

<*>           IPv6: IPv6-in-IPv6 tunnel
```

```
Networking options --->
... 중략 ...

SCTP Configuration (EXPER ...) -->
<*>             The SCTP Protocol (EXPER ...)
[*]             SCTP: Debug messages
[*]             SCTP: Debug object counts

SCTP: Cookie HMAC Algorithm
```

2.1.2. lksctp-tools 설치

또한 SCTP 응용 개발을 위해서는 lksctp-tools가 필요하다. 현재 lksctp-tools의 최신버전은 lksctp-2.6.10-1.0.2 이고 LKSCTP 공식 홈페이지(http://lksctp.sourceforge.net)에서 필요한 패키지를 받을 수

있다. lksctp-tools의 설치방법은 너무 간단하므로 생략한다. 다만 소스를 컴파일하여 설치하면 /usr/local을 기본경로로 설치된다. SCTP 모듈 활성화와 lksctp-tools를 설치가 끝나면, 다음의 명령으로 SCTP의 지원 여부를 알 수 있다.

- ✓ **$ checksctp**
  SCTP supported

lksctp-tools에는 몇 가지 유용한 프로그램이 제공되는데 이 문서에서 가장 활용 빈도가 높은 것은 withsctp이다. withsctp는 DL 라이브러리를 사용하여 응용 프로그램의 socket() 호출을 overriding 하여 SOCK_STREAM으로 생성시킨 모든 소켓을 one-to-one 스타일의 SCTP 소켓으로 생성해 주는 툴이다. 즉, withsctp를 사용하면 TCP 응용 코드를 수정하지 않고도 SCTP로 전환할 수 있게 된다. 여기에서는 withsctp의 내부 구조는 생략한다. withsctp의 사용법을 다음과 같다.

- ✓ **$ withsctp sshd**
  $ withsctp ssh localhost

2.1.3. HTTP 서버 및 클라이언트 설치

이번 테스트에 사용된 HTTP 서버 및 클라이언트는 Apache 1.3.33 과 Mozilla 1.7.8 이다. 각각 공식 홈페이지에서 받아서 설치한다. (http://www.apache.org, http://www.mozilla.org)

Apache 1.3.33 버전은 DSO(Dynamic Shared Object)의 사용 등 다양한 설치 방법이 있다. 기본 설정은 모든 모듈이 정적으로 httpd라는 바이너리로 빌드되는 것이며, 기본 설치 경로는 /usr/local/apache이다. 이번 실험에서는 필요한 모듈을 동적으로 로딩하는 DSO를 사용했으며 설정은 다음의 스크립트를 수정하였다.

```
#! /bin/sh

./configure --enable-access=shared      --enable-auth=shared ₩
            --enable-include=shared     --enable-log_config=shared ₩
            --enable-env=shared         --enable-setenvif=shared ₩
            --enable-mime=shared        --enable-status=shared ₩
            --enable-autoindex=shared--enable-asis=shared ₩
            --enable-cgi=shared         --enable-negotiation=shared ₩
            --enable-dir=shared         --enable-imap=shared ₩
            --enable-actions=shared     --enable-userdir=shared ₩
            --enable-alias=shared
```

./configure 후, make, make install을 수행하면 지정된 설치 디렉토리(이 경우는 /usr/local/apache)에 Apache 서버가 설치 된다. /usr/local/apache/bin/httpd -l 명령으로 빌드한 모듈을 볼 수 있다.

- ✓ **$ /usr/local/apache/bin/httpd -l**
  Compiled-in modules:
    http_core.c
    mod_so.c

Mozilla의 설치는 Mozilla 공식 홈페이지에서 제공하는 바이너리 설치를 권장한다. 소스 설치는 너무 많은 옵션으로 ./configure 자체가 복잡하고, 빌드 시간도 많이 소요된다. 만약 소스 설치를 하고 싶다면 Mozilla Build Configurator (http://webtools.mozilla.org/build/config.cgi)를 이용한다. 리눅스에서 바이너리의 설치는 필요한 파일을 받아 압축만 풀고 원하는 경로로 옮기면 끝이다. 이번 테스트에 사용한 Mozilla는 GTK2+ 모듈을 사용하는 mozilla-i686-pc-linux-gnu-1.7.8-gtk2+xft.tar.gz이다. 이 것을 /usr/local/mozilla-gtk2 로 압축을 풀어서 사용하였다.

2.2. withsctp를 사용한 HTTP over SCTP

이번 절에는 withsctp를 사용하여 Apache 서버와 Mozilla 클라이언트의 통신을 SCTP 상에서 수행해 본다. 앞 절에서 소개한 테스트 환경이 구축이 되면 다음과 같은 간단한 방법으로 HTTP / SCTP 응용을 테스트할 수 있다.

2.2.1. HTTP 서버 및 클라이언트의 실행

✓ 서버 측

```
# cd /usr/local/apache              ← Apache가 설치된 디렉토리로 이동
# withsctp bin/apachectl start      ← withsctp를 사용한 Apache 서버의 실행
/usr/local/apache/bin/apachectl start: httpd started
# cat logs/httpd.pid                ← 부모 프로세스의 프로세스 식별자 확인
5767
```

✓ 클라이언트 측

```
$ cd /usr/local/mozilla-gtk2        ← Mozilla가 설치된 디렉토리로 이동
$ withsctp ./mozilla                ← withsctp를 사용한 Mozilla 클라이언트의 실행
```

2.2.2. HTTP/SCTP 응용의 통신결과 캡쳐

이제 전 절에서 설명한 방법대로 실행한 Apache 서버와 Mozilla 클라이언트간의 통신 결과를 살펴본다. 한번은 withsctp를 사용한 SCTP 통신을 수행하였고, 한번은 withsctp를 사용하지 않고 TCP 통신을 수행하였다. 서버의 도메인은 http://protocol.knu.ac.kr이지만 SCTP를 사용한 클라이언트의 경우는 DNS로부터 정보를 받을 수 없다. 따라서 때문에 직접 웹브라우저창에 서버의 IP 주소를 직접 입력하여 수행해야 되지만 편의상 /etc/hosts 파일에 protocol.knu.ac.kr의 IP 주소를 cpl 맵핑하여 http:///cpl/로 접속하였다. 다음은 그 결과 들이다.

► 패킷 수 비교 화면

다음 그림 1 은 HTTP/TCP와 HTTP/SCTP 통신에 사용된 패킷의 수를 비교한 화면이다. withsctp를 사용한 SCTP 통신이 TCP 통신보다 조금 더 많은 패킷 수를 발생시켰다.

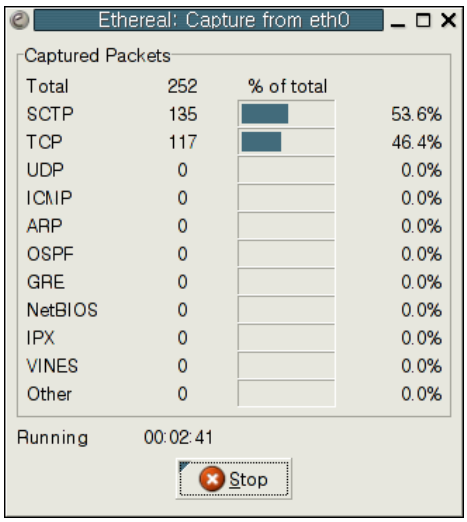

그림 1. HTTP/SCTP vs. HTTP/TCP: 패킷 수 비교

► 패킷 캡쳐 결과

그림 2 는 withsctp를 사용한 SCTP 통신의 결과를 나타낸다. 4-Way Handshaking 이후 클라이언트에서 전송한 최초의 DATA 패킷이 바로 HTTP의 GET 메시지이다. 아직 까지 이더리얼 프로그램이 SCTP의 DATA 패킷의 상위 단계의 프로토콜을 분석하지 못하고 단순히 DATA로만 표시하고 있지만 HTTP 통신은 원활히 수행 됨을 볼 수 있다.

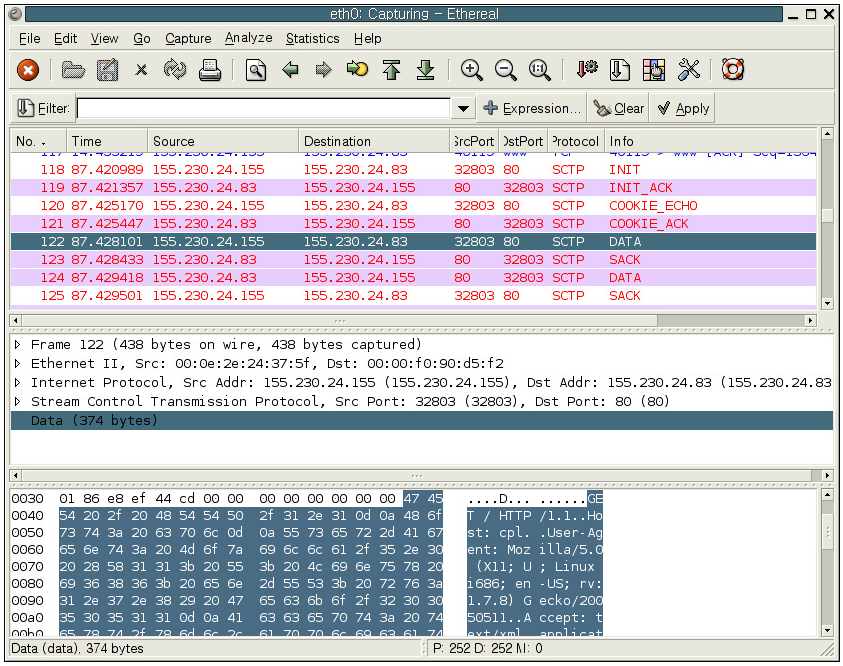

그림 2. HTTP/SCTP 패킷 캡쳐 화면

그림 3 은 withsctp를 사용하지 않은 TCP 통신의 결과이다. HTTP 통신이 정상적으로 이루어짐을 알 수 있다. 위와는 달리 이더리얼에서는 TCP 통신을 사용하는 응용 프로토콜을 분석해서 표시해 주고 있다.

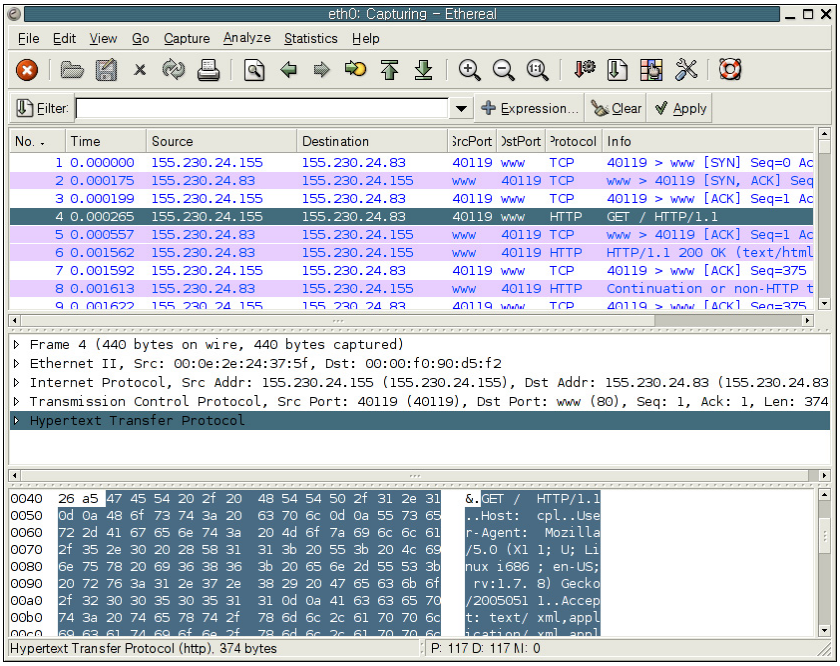

그림 3. HTTP/TCP 패킷 캡쳐 화면

► **웹 브라우저 화면 캡쳐 결과**

그림 4 는 SCTP로 통신하고 있는 웹 브라우저의 화면이다. 데이터 전송을 다 끝내지 못하고 서버의 응답을 기다리는 것을 볼 수 있는데, 이 것은 외부 TCP 서버에서 지원하는 카운터를 링크했기 때문이다.

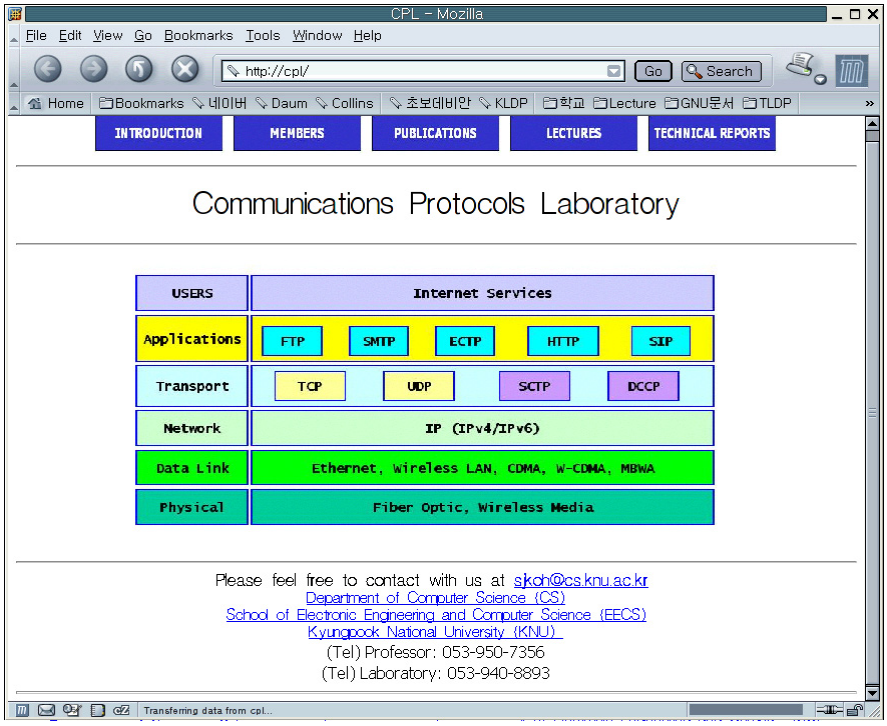

그림 4. HTTP/SCTP 웹 브라우저 화면

그림 5 는 TCP로 통신하고 있는 웹 브라우저의 화면이다. TCP를 사용하는 경우 (당연하지만) 외부 카운터를 정상적으로 받아 오는 것은 알 수 있다.

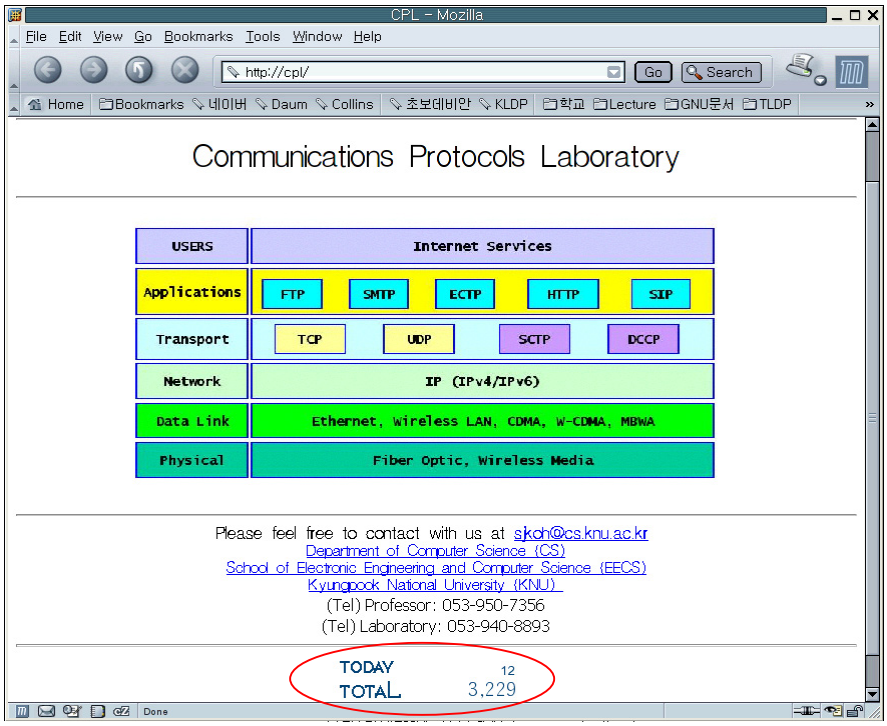

그림 5. HTTP/TCP 웹 브라우저 화면

3. 결론 및 향후 연구

지금까지 리눅스 커널 2.6.x와 lksctp-tools에서 제공하는 withsctp 툴을 사용한 HTTP/SCTP 통신을 수행하는 방법을 살펴 보았다. 이 번 테스트를 통해서 웹과 같은 TCP 응용을 최소한의 노력 만으로 SCTP로 전환하는 것이 가능함을 알 수 있었다. 또한 리눅스 환경에서는 withsctp의 제공으로 컴파일도 다시 수행할 필요 없이 one-to-one 인터페이스를 사용하는 SCTP통신이 가능함을 확인 하였다.

하지만 이 문서에서 언급하지 않은 몇 가지 해결해야 할 사항이 있다. 필자의 실험 결과 withsctp를 사용한 통신에는 아직까지 밝혀지지 않은 몇 가지 문제점이 있었다. 본 문서에서 테스트한 Apache의 버전은 1.3.33이었는데 Apache 버전 2.0.x 에서는 서버 측에서 html 파일을 읽어 들여서 해석하는 데에 문제가 있었다. withsctp를 사용하지 않고 직접 코드를 수정하여 one-to-one 스타일의 SCTP 소켓을 사용한 경우에도 withsctp를 사용했을 때와 동일한 결과를 보였다. 이것으로 미루어 보아 withsctp 자체의 문제가 아니라 현재 리눅스 상에 구현된 SCTP의 one-to-one 인터페이스의 호환성의 문제로 보여진다.

버전이나 프로그램에 따른 들쭉날쭉한 withsctp의 동작은 HTTP 서버에서만 아니라 다른 응용에서도 나타났다. withsctp를 사용한 vsftpd는 훌륭히 동작하는 반면, withsctp를 사용한 proftpd에서는 클라이언트 측에서 파일 리스트를 받아온 후에 예기치 않은 3분 가량의 지연 현상이 생겼다. 또한 withsctp를 사용한 ssh 통신은 제대로 수행이 되었다.

이러한 실험들을 통해서 내린 결론은 “one-to-one 인터페이스를 사용한 TCP응용의 SCTP 전환은 생각외로 아주 쉽게 가능하지만 응용에 따라서 기복을 나타낸다.”는 것이다. 이 것을 해결하기 위해서는 다음에 초점을 둔 연구가 더 많이 요구된다.

- ✓ 현재까지 구현된 리눅스 상의 SCTP 프로토콜 one-to-one 스타일 인터페이스가 기존 소켓 API를 완전히 지원하지 못한다면 어느 부분에서 문제가 있나?
- ✓ TCP와 SCTP one-to-one 인터페이스 모두에 완벽한 호환성을 유지하기 이해서는 기존의 TCP 응용을 어떻게 작성하면 되는가?

앞으로 문제점이 들어난 응용들의 소스들을 분석하여 위의 문제를 해결한다면 SCTP가 차세대 수송계층 프로토콜로 확고히 자리를 매김 하는데 밑거름이 될 것이다.