Technische Universität München
Lehrstuhl Informatik VIII
Prof. Dr.-Ing. Georg Carle
Christian Grothoff, Ph.D.
Stephan M. Günther, M.Sc.

Master Course Computer Networks

Project: Proxies for HTTP over SCTP

Submission deadline is February 1st via SVN
Some project parts have additional separate deadlines

General Information

For this project, your overall task is to gain some experience with SOCKS, HTTP, SCTP, protocol design and network measurement and to demonstrate your skills by implementing a pair of proxies and doing some experimental evaluation. This project cannot be done quickly, you must plan to spend several weeks for each of the implementation, testing, measurement and documentation tasks. You are encouraged to discuss problems and strategies with your fellow students, but not allowed to look at code together. Please be aware that you will immediately fail the project if you do copy & paste.
This is a Master-level project and good **programming skills** are mandatory for parts of the project. Not all of the code must be implemented in C; in fact, as long as you can show us how to run the code on GNU/Linux, any language is acceptable for any part of the project. Especially measurements might be better done with the help of scripting languages.
This project consists of five parts: planning, protocol design, two interdependent implementations and a measurement study. Each part has a different deadline and weight for the project grade (as indicated). The project is open-ended in terms of optimizations you might choose to implement on purpose. This means that you are free to choose how to proceed once you have the basic functionality implemented. We want you to be creative!
For your investigation, we suggest you look at tools such as `wireshark`, `wget`, `siege` and possibly GNU libmicrohttpd. Finding out how to install, use and evaluate the output of these tools is part of the project.
You must submit all of your source code and reports via Subversion by **February 1st**. Some project parts have additional separate deadlines. Late submissions will receive a score of zero unless you notify the instructor about a valid reason (such as sickness) **prior** to the deadline.
You need to register for the project at the course page. You will need submit "LRZ Kennung". It is used for authentication and to grant access to your Subversion repository which will be used for the submission of your PDF reports and your project code. Furthermore, your repository contains an SSH key for access to your virtual machines. The virtual machines have been configured with a specific network setup and an HTTP server that the benchmark should be run against. You need to perform your measurements using the provided system; however, you're of course free to do your actual development elsewhere.
Details how to access your repository will be sent by mail after the registration period for the project has ended. In the meantime, please make sure you have an "LRZ Kennung", and are familiar with Subversion and `ssh` (including `ssh-keygen` and related tools).

1 Project Plan (5%) – November 8th

Prepare a project plan that gives details about your approach, and lists intermediate milestones which allow yourself to assess whether you progress as planned.

2 Protocol Design: HTTP-SCTP (10%) – December 15th

Design a protocol for tunneling HTTP over SCTP. Your implementation must be able to tunnel multiple HTTP streams over the same SCTP connection, and it must be possible to add additional HTTP streams to an existing SCTP association to be executed in parallel (no HOL blocking).
You can choose to specify optimizations such as avoiding re-transmission of redundant HTTP headers. You need to submit a PDF with a protocol design by the deadline. Your final implementation can deviate from this design, but you must then describe and justify the changes in your final report.

3 SCTP Server: HTTP-SCTP to HTTP-TCP (30%)

Implement an SCTP server which implements your HTTP-SCTP protocol, accepts incoming connections over SCTP, forwards them to a (local) HTTP server via TCP and sends the responses back via HTTP-SCTP.
You can assume that the target HTTP-SCTP server runs on a well-known (as in, defined by you) port at the same IP address as the original target of the HTTP-TCP request.

4 SOCKS Proxy: HTTP-TCP to HTTP-SCTP (30%)

Implement a SOCKS proxy which accepts HTTP connections via TCP and then creates an SCTP connection. You might want to use GNU libmicrohttpd to implement the HTTP server (there is an example for a working SOCKS proxy with GNU libmicrohttpd in the GNUnet `src/gns/` codebase); however, you are welcome to write your own SOCKS proxy from scratch or use other libraries. Naturally, the HTTP server will **not** see the proper client IP address for the HTTP-SCTP requests. This is not a concern for this project.

5 Measurement Report (25%) – February 1st

Measure the performance of accessing the preconfigured website on your virtual machine with your HTTP-SCTP protocol for the network link vs. using multiple HTTP-TCP connections. Specifically, the idea is that your HTTP client should either use the SOCKS proxy via loopback, then use a (lossy, high-latency) link to the target system via SCTP and then at the server's system access the SCTP server which then obtains the website via TCP again via loopback. This should be compared to using multiple HTTP-TCP links via the same network infrastructure.
We're interested in bandwidth consumption and latency. Please follow the instructions below to simulate a lossy link. This is necessary to obtain comparable and reproducible results. The instructions assume that you have setup the VMs as explained in the lecture, i.e. VM B routes between A and C. On VM B (the one with a total of 3 network interfaces) you have two virtual interfaces eth1 and eth2. Use the following command to simulate delay / packet loss:

```
# tc qdisc add dev <interface> root netem \
  delay <mean> <deviation> <correlation> \
  loss <probability> <correlation>
```

For the delay, you can specify the mean value in ms, a deviation from this mean (+/- N ms), and a correlation. The last parameter indicates how much the randomized delay for packet n depends on the value for packet n-1. For the loss, you only specify the probability in % and the correlation value.
A valid command thus looks like this:

```
# tc qdisc add dev eth1 root netem delay 100ms 20ms 25% loss 1% 25%
```

To remove the qdisc from the interface issue:

```
# tc qdisc del dev <interface> root
```

After installing a qdisc you may want to check whether it is active by pinging between VMs A and C. Note that a qdisc has only effect on ingress packets. **As a result you must install the same qdisc on both `eth1` and `eth2` to obtain a symmetric link.**
For your (final) benchmarks please

1. use the reference webpage we provided,
2. use our VMs, and
3. test your code with the following nine settings:

no.	delay mean	delay dev.	delay corr.	loss prob.	loss corr.
0	0ms	0ms	25%	0%	25%
1	0ms	0ms	25%	1%	25%
2	0ms	0ms	25%	5%	25%
3	100ms	20ms	25%	0%	25%
4	100ms	20ms	25%	1%	25%
5	100ms	20ms	25%	5%	25%
6	500ms	100ms	25%	0%	25%
7	500ms	100ms	25%	1%	25%
8	500ms	100ms	25%	5%	25%

4. compare the results to running the benchmark directly using HTTP over TCP only (without any of your software)

Make sure that you have installed the same qdisc on **both interfaces** of your middle VM and remove the old qdisc before adding another one.
For benchmarking, you need to install GNU parallel and curl (you may need to run “apt-get install parallel curl”). First, use the `sproxy` tool from the Siege HTTP benchmark suite to create a “urls.txt” file from the website (make sure the result is free of the comment lines). Once you have generated that file, you can then run

```
$ time cat urls.txt | parallel -j 30 curl -o /dev/null --socks4 localhost:5880 {}
```

to download all URLs from `urls.txt` in parallel (with 30 streams active at the same time) using your proxy running on port 5880. Leave out the proxy argument to access the site directly.
Your report must include a description of your final protocol (in particular any improvements you made over the trivial implementation discussed in class), your measurement setup, your experimental results and an interpretation of the results with focus on suggestions for further improvements.

6 Scope

Your implementations must support HTTP with the “HEAD” and “GET” methods. However, we will give bonus points for working support of HTTPS or “PUT” and “POST”.
The SOCKS proxy must work if the SCTP server is running on the target system. Implementation SCTP-detection and falling back to HTTP-TCP if SCTP will again give bonus points.

7 Futher reading

- `http://www.ibm.com/developerworks/linux/library/l-sctp/`
- `http://tools.ietf.org/html/draft-natarajan-http-over-sctp-01`
- `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`