

12th Assignment: Network Protocols and Architectures, WS 13/14

Question 1: (150 points) Protocol Implementation: Gateway Control Protocol

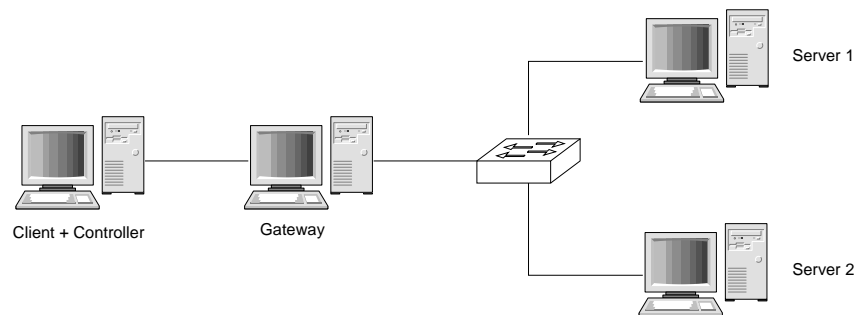


Figure 1: Experimental Setup

This assignment is dedicated to the implementation of the UDP-based Gateway Control Protocol (GCP) in Assignment 11.

The functionality of the gateway is twofold: (i) the forwarding process can be controlled by control messages and (ii) it will forward *MSG* messages to the destination host. If a channel to a destination host indicated by its IP and UDP port number is open the gateway forwards the messages to its destination. Otherwise, the gateway drops the message. For the sake of simplicity, assume that the reverse direction, i. e., server to client, is never blocked.

Assume a loss-free channel, i. e., the protocol used for this assignment does *not* need to consider retransmissions. As indicated in the figure above, the gateway should support multiple clients and servers. As the data messages contain both the sender and the destination addresses, no state needs to be maintained for the forwarding functionality of the gateway.

The implementation of the client + controller (`client.pl/.py`) as well as the server (`server.pl/.py`) for the GCP is provided via ISIS¹. Therefore, you do not need to implement your own client and server! Help on the usage of these tools can be obtained using the `--help` parameter. While `client.pl` sends data and control messages `server.pl` implements a simple echo server that echos incoming messages.

The GCP is using UDP and defines seven different message types in plain ASCII text. The messages are defined as follows:

- **Data:** `MSG|DST_IP|DST_UDP_PORT|SRC_IP|SRC_UDP_PORT|Message text`
 This will send a *data* message containing *Message text*. The address of the sending host is indicated by *SRC_IP* and *SRC_UDP_PORT*. Similarly, the address of the receiving host is indicated by the two *DST* fields. Note: on the way back (server to client), the *DST* and *SRC* fields will be swapped by the server. The message is *never* acknowledged!
 Example message: `MSG|127.0.0.1|12880|127.0.0.1|12881|Foo bar`
- **Soft state control:** `SOPEN|DST_IP|DST_UDP_PORT|LENGTH`
 This control message opens a bidirectional communication channel to the host indicated by the *DST* fields for a time span of *LENGTH* seconds. If the state is not refreshed within the

¹see <https://www.isis.tu-berlin.de/2.0/mod/resource/view.php?id=36743>

indicated time span—either by a data message (MSG) or by a SOPEN—the channel is closed automatically. A data message will reset the timer to restart the period indicated in the most recent SOPEN message.

Example message: `SOPEN|127.0.0.1|12880|2`

- **Soft state control:** `SCLOSE|DST_IP|DST_PORT`

This control message closes a channel previously opened by SOPEN.

Example message: `SCLOSE|127.0.0.1|12880`

- **Hard state control:** `HOPEN|DST_IP|DST_UDP_PORT`

This control message opens a bidirectional communication channel to the host indicated by the *DST* fields. The channel remains open unless explicitly closed by HCLOSE.

Example message: `HOPEN|127.0.0.1|12880`

- **Hard state control:** `HCLOSE|DST_IP|DST_UDP_PORT`

Close a communication channel that was opened by HOPEN.

Example message: `HCLOSE|127.0.0.1|12880`

- **Control:** `SUCCESS|DST_IP|DST_UDP_PORT|LENGTH`

Control message sent by the gateway to indicate a successful execution of a gateway command, i. e., SOPEN (SCLOSE) or HOPEN (HCLOSE). All values of the command, e. g., DST_IP, DST_UDP_PORT or LENGTH, are copied unmodified into the reply. If no length is present, i. e., in the case of HOPEN, the length is set to 0.

Example message: `SUCCESS|127.0.0.1|12880|2`

as a response to `SOPEN|127.0.0.1|12880|2`

- **Control:** `FAIL|DST_IP|DST_UDP_PORT|LENGTH`

Control message sent by the gateway to indicate a failed execution of a gateway command, i. e., SOPEN (SCLOSE) or HOPEN (HCLOSE). All values of the command, e. g., DST_IP, DST_UDP_PORT or LENGTH, are copied unmodified into the reply. If no length is present, i. e., in the case of HOPEN, the length is set to 0.

Example message `FAIL|127.0.0.1|12880|2`

as a response to `SOPEN|127.0.0.1|12880|2`

Implement the gateway in the hard-state and soft-state variant. Please note that *both* variants must be implemented in the same program. Do *not* provide two different programs for the hard state and for the soft state version. Which variant is chosen should only depend on the protocol messages the gateway receives.

Extend the run script `run.sh` so that it takes all actions required to run your program (compile code, start the gateway listening on port 12890). We will not accept solutions that are shipped with a non-working or incomplete run script.

Choose the language of your preference to implement the protocol and function of the gateway. The final grading will depend on a set of automatic test cases—similar to the ones provided in `client.pl`—that check whether the protocol is implemented correctly.

Your program *must* run on the Ubuntu machines provided by IRB. We will *not* accept any Windows programs! Provide the source code in a readable form.

Due Date: Wednesday, January, 29th 2014 only until 09:55 h s. t.

- **As ZIP file:** Uploaded via ISIS (<https://www.isis.tu-berlin.de/2.0/course/view.php?id=349>)
- Put your name, StudentID number (Matrikelnummer) **and** the name of your tutor on your solution.