Project for CS399: Development and Testing of Simple Internet Phones

David L. Dill Patrice Godefroid

October 27, 1999

1 Overview

This project will consist of two parts.

- Part 1: Development and testing of a switch for simple internet phones.
- Part 2: Interoperability testing with other switches.

Development will be done in C and following the guidelines below. Testing will be done using VeriSoft. Precise instructions are presented in the following sections.

2 Simple Internet Phones: Specification

In the first part of this project, you will develop a switch implementing the basic signalling and call processing logic for simple internet phones. This switch will support three features: basic call, call forwarding busy and conference call.

This section presents the requirements that your switch has to conform to in order to be a valid solution to the problem. This section is referred to as the "specification" in what follows.

Note that this specification is not complete, i.e., it does not restrict the solution space to a unique solution. When several interpretations are possible, use the one that you find the most appropriate. General robust implementations are better than restrictive ad-hoc ones. Since you cannot control the behavior of the other components in the network, adopting a defensive style of programming is a good idea (always expect the unexpected).

2.1 Overall Architecture

A network is composed of a finite set of pairs of processes. Each pair consists of a phone process P_i and a switch process S_i, and is identified by an address ("i" in this text). Network elements (phones and switches) can communicate by sending messages to each other. Since the communication is asynchronous, message queues (of bounded size) are used to provide buffering.

For this project, a phone process simulates sequences of actions that a telephone user can perform, such as picking up the phone, dialing a number, hanging up, turning on or off features, etc. For testing purposes, this process will be modeled by a simple nondeterministic C program that can execute any sequence of such actions.

The switch process S_i associated to a phone process P_i represents that phone in the network. All messages from P_i are sent to S_i, and all messages to P_i are sent by S_i. A switch S_i can also send and receive messages from any other switch in the network. In the first part of this project, you will develop the code for such a switch S_i.

Each phone process P_i has a single input queue, while each switch process S_i has two input queues, one for receiving messages from its phone P_i and one for receiving messages from other switches. Phones and Switches communicate with each other according to some specific rules that are now described. (This protocol is inspired by an existing protocol named Q931).

2.2 Messages

Here is the complete list of messages that can be exchanged between network elements. For each message sent by a switch (to its associated phone or to another switch), the rule defining when the message should be sent is also presented.

1. From phone to switch:
   (a) off-hook: the phone is off-hook.
   (b) dial(addr): the address "addr" is dialed.
   (c) on-hook: the phone is on-hook.
   (d) call-forwarding-busy(addr): call-forwarding-busy is turned on with the address "addr" as the forwarding address if "addr" is nonzero; if "addr" is zero, the call-forwarding-busy feature is turned off.
   (e) conference-call: used for calls with more than two participants.
2. From switch to phone:

(a) dial-tone: sent after receiving "off-hook" and if the phone is not already ringing or in a call; also sent after "conference-call" as described in Section 2.5.
(b) start-ringing: sent after receiving "setup(addr)" and if the phone is not already ringing or in a call.
(c) stop-ringing: sent after receiving "disconnect(addr)" and if the phone is currently ringing.
(d) ringing-indication: sent after receiving "alerting(addr)" and if the phone is still off-hook.
(e) busy-indication: sent after receiving "suspend(addr)" and if the phone is still off-hook.
(f) connect-indication: sent after receiving "connect(addr)" or "connect-ack(addr)" and if the phone is still off-hook.
(g) disconnect-indication: sent after receiving "disconnect" and if the phone is still off-hook.

3. From switch to switch: (note: the argument "addr" in lower-case denotes the sender of the message for all the following message types)

   (a) setup(addr): sent to "ADDR" after receiving "dial(ADDR)" and if no "on-hook" has been received since sending last "dial-tone".
   (b) alerting(addr): sent to "ADDR" after receiving "setup(ADDR)" and if the phone is not already ringing or in a call.
   (c) suspend(addr1,addr2): sent to "ADDR" after receiving "setup(ADDR)" and if the phone is already ringing or in a call; "addr1" is the address of the sender; "addr2" is the forwarding address specified by the last message received of the form "call-forwarding-busy(addr2)".
   (d) connect(addr): sent to "ADDR" after receiving "off-hook" and if last message sent to "ADDR" was "alerting(addr)".
   (e) connect-ack(addr): sent to "ADDR" after receiving "connect(ADDR)" and if the phone is still off-hook.
   (f) disconnect(addr): sent to "ADDR" after receiving "on-hook" and if last message sent to "ADDR" was "connect-ack(addr)" or "setup(addr)".
   (g) release(addr): sent to "ADDR" after receiving "disconnect(ADDR)".

Here are some additional rules.

1. A phone should never receive "start-ringing" if it is off-hook.
2. A phone should never receive "dial-tone" or "ringing-indication" or "busy-indication" or "connect-indication" or "disconnect-indication" when it is on-hook.

3. A phone should never receive "stop-ringing" if it is not already ringing (triggered by "start-ringing") or if it off-hook.
4. Once a call is established, the callee can never terminate the call: if the callee hangs up (by sending "on-hook") then picks up the phone again (by sending "off-hook'), the call is still on.
5. After a phone sends "on-hook" followed by "off-hook", the next message sent to the phone should either be "dial-tone" or "connect-indication", or the phone should be in a call.

Note that any message that can be sent by a phone to its switch can be sent at any time. This implies that "off-hook" and "on-hook" messages do not necessarily alternate: for instance, a "on-hook" message may follow another "on-hook" message without any "off-hook" message occurring in between.

We now present a brief description of the three features that need to be implemented, as well as illustrative scenarios.

2.3 Basic Call

A basic call involves two phones and their associated switches. Let us denote the originating phone (caller) by P_1 , its associated switch by S_1, the terminating phone (callee) by P_2 and its associated switch S_2. The standard scenario for successfully making a call follows the following steps.

1. Initially, P_1 and P_2 are on-hook.
2. P_1 sends "off-hook" to S_1.
3. S_1 receives "off-hook" and sends "dial-tone" to P_1.
4. P_1 receives "dial-tone" and sends "dial(2)" to S_1, where "2" is the address of S_2.
5. S_1 receives "dial(2)" and sends "setup(1)" to S_2.
6. S_2 receives "setup(1)" and sends "start-ringing" to P_2 and "alerting(2)" to S_1.
7. S_1 receives "alerting(2)" and sends "ringing-indication" to P_1.
8. Meanwhile, P_2 sends "off-hook" to S_2.
9. S_2 receives "off-hook" and sends "connect(2)" to S_1.
10. S_1 receives "connect(2)" and sends "connect-indication" to P_1 and "connect-ack(1)" to S_2.
11. S_2 receives "connect-ack(1)" and send "connect-indication" to P_2.

12. The connection (also referred to as "call") is now established and both parties can talk using P_1 and P_2.
13. To end the call, P_1 sends "on-hook" to S_1.
14. S_1 receives "on-hook" and sends "disconnect(1)" to S_2.
15. S_2 receives "disconnect(1)" and sends "disconnect-indication" to P_2 and "release(2)" to S_1.
16. After P_2 sends "on-hook" to S_2, P_2 is free to make or receive another call.

Note that only the caller P_1 can end the call: for instance, if after Step 12 in the above scenario, the callee P_2 sends "on-hook" followed by "off-hook" to S_2, the call is still on between P_1 and P_2.

2.4 Call Forwarding Busy

Calls to a busy phone can be redirected to another phone as follows. A forwarding address "addr" for a phone can be set by executing "call-forwarding-busy(addr)" on that phone.

When a switch S_1 receives "suspend(2,addr)" from another switch S_2 and when "addr" is nonzero, S_1 knows that the phone P_2 associated with S_2 is busy and that call-forwarding-busy is turned on on P_2 with "addr" as the forwarding address. Therefore, S_1 does not send "busy-indication" to P_1, but instead sends "setup(1)" to the switch at the address "addr". The call setup between P_1, the phone at address "addr" and their associated switches then proceeds as usual (see previous section).

2.5 Conference Call

This feature allows more than two phones to participate in a call. Once the caller has successfully established a connection with a first callee (for instance, after following the first 12 steps of the scenario described in Section 2.3), the caller P_1 can initiate another connection by sending "conference-call" to S_1. S_1 receives "conference-call" and responds by sending "dial-tone" to P_1. The call setup with the second callee then proceeds as usual (see Section 2.3). Once this second connection is established successfully, the 3 parties (the caller and the two callees) can communicate with each other. This process can be repeated again by the caller to add one-by-one more participants (callees) to the call.

If the caller P_1 sends a second "conference-call" message to S_1 before the connection to the second callee is successfully established, the second connection attempt is terminated: when S_1 receives the second "conference-call" from P_1, it sends a "disconnect(1)" to the second callee if it has previously sent a "setup(1)" to this callee. However, this does not impact the conference call between the caller and the callee(s) previously connected.

To end the conference call, the caller P_1 simply sends "on-hook" to S_1, which then sends one "disconnect(1)" message to each of the callees in the call. The disconnection phase then proceeds as usual, following Steps 15 and 16 of the scenario described in Section 2.3.

3 Part 1: Development and Testing

3.1 Description

Develop the code for a switch that conforms to the specification presented in the previous section. Then, test that switch processes executing your code can communicate with each other while conforming to the specification.

Testing your code may involve playing scenarios manually using the veriSoft simulator, writing assertions for testing that rules described in the specification are satisfied, writing other phone processes, etc. No specific instructions are given concerning the type of testing that should/need be performed. The only recommendation is to test the correctness of your code as thoroughly as possible.

To get started, copy the implementation environmemt and test harnesses available on saga3 in

`~patrice1/verisoft/examples/project`.

Follow the instructions and explanations given in the README file in that directory.

Note that you may modify this code as you wish. However, be careful not to modify the interface (overall architecture and message types) between phone and switch processes, since this interface needs be preserved in order for your code to communicate with switches developed by other project participants (see Part 2 below).

3.2 Deliverables (Deadline: November 19)

Code implementing your switch. This code has to be written in C, compilable with gcc on the machine saga3, and fit into a single self-contained file. Please leave the file anonymous (do not include your name or email address).

Send this file by email to `god@bell-labs.com` by **Midnight, Friday November 19, 1999**. Reception of the file will be acknowledged via email.

Warnings: this deadline if *firm*! Also, *only the first file received from each participant will be accepted!* (Subsequent files will be automatically discarded.)

4 Part 2: Interoperability Testing

4.1 Description

The purpose of this part is to test that your switch can communicate with switches developed by other participants of the project.

By Monday November 22 (midnight), you will receive via email 3 source files, each file corresponding to a switch developed by another participant. Each switch will be identified by an (anonymous) <identification number> placed in a comment at the beginning of the file.

Test the interoperability of these switches combined with your switch in order to detect any problems, in other switches, in your switch or in the combination of switches. To get started, use the implementation environment and test harnesses you have already used for testing your switch in Part 1.

Again, no specific instructions are given concerning the type of testing that should/need be performed. For instance, you are free to test other switches one-by-one, or together, or both, using the configurations (number of network elements) of your choice. The only recommendation is to try to find as many problems (bugs and incompatibilities) as possible.

4.2 Deliverables (Deadline: December 5)

After performing interoperability testing, prepare a short report (3 to 5 pages are sufficient) summarizing your findings. Your report should have five sections (in the order of your choice).

One section for each of the 3 switches coming from other participants that you have tested against your switch. The title for each of these sections should be “Switch <identification number>”. In each section, start with a grade for the switch: 1 (good), 2 (average), or 3 (poor). Justify your evaluation. (Note: you may give the same grade to more than one switch, any combination is allowed.)

In the fourth section entitled “My Switch”, report any problems detected in your code, if any. (Don’t be shy – problems in your switch detected by others but missed by you are not good.)

In the fifth section entitled “Miscellaneous”, present any comment you may have on any aspect of this project (for instance, comments on your code, on the specification, on testing your code, on interoperability testing, on the project, on the course, etc.). Also, tell us what you think of VeriSoft. (Did you like/hate it? Why? What should be improved? Etc.) Finally, tell us how much time you have spent on this project and what percentage of this time you have spent on which part (such as design/development, testing, interoperability testing, report, etc.). Thank you in advance for your feedback.

Note that the grades given by other participants to your code will not determine your grade. We know several interpretations of the specification are possible, and the most popular interpretations are not necessarily the "right ones". So, relax and play the game.

The formats accepted for the report are plain (ascii) text, postscript, and pdf. Your report should fit into a single self-contained file.

Send this file by email to `god@bell-labs.com` by **Midnight, Sunday December 5, 1999**. Reception of the file will be acknowledged via email.

Warnings: this deadline if *firm*! Also, *only the first file received from each participant will be accepted!* (Subsequent files will be automatically discarded.)

5 Final Remarks

Any questions should be sent to `god@bell-labs.com`. Questions and answers will be sent to all the project participants in order to avoid redundant questions.

Please remember to be careful with the IPC resources on the machines on which you run your code!