PEPA-NG

What is Pepa-Ng

Pepa-Ng is a program designed for passing messages between three sets of sockets:

- The Shva server: It allows sending and receiving messages.
- The IN channel receives messages and forwards them to the Shva server.
- The OUT channel sends the messages received from the Shva server.

Why Is It “Pepa” and “Ng”

The Pepa-Ng is the latest version of the Pepa software. Its predecessor, Pepa, was created to link the external server "Shva" with the readers and writers using pipes on the local hosts. The original name referred to the fact that it was a "pipe" tunnel.

The "Ng" in Pepa-Ng stands for "New Generation." This latest version of Pepa has been fully redesigned and now works with TCP/IP sockets. Pipes are no longer supported.

The Shva Server

The Shva server sends messages to Pepa-Ng, which is then forwarded to the OUT socket. The address of the Shva server is defined in the command line and represented by a single socket connection..

The IN channel

Two socket sets represent the IN channel internally: one listening socket and multiple reading sockets. The listening socket is dedicated to accepting incoming connections. The reading sockets are dedicated to receiving messages from the external actor.

An external actor should always communicate with the reading socket; the information about the listening socket is only given here to explain the Pepa-Ng internal implementation.

Pepa-Ng reads messages from all IN reading sockets in first-in-first-out (FIFO) mode and forwards them to the Shiva server. Since TCP/IP communication is out of Pepa-Ng's control, there is no guarantee that multiple buffers sent through the several IN sockets will be received in the same order. It is only a guarantee that packets sent through the same socket will be received in the same order.

To illustrate the ordering, let's pretend there are two IN reading sockets: A and B. The external actor sends a group of buffers simultaneously:

Socket A: {a1, a2, a3}
Socket B: {b1, b2, b3}

All A buffers are guaranteed to be received in order: {a1, a2, a3}.
It is also guaranteed that all B buffers will be received in order: {b1, b2, b3}.
However, if these buffers were sent in this order:
A+B: {a1, b1, a2, b2, a3, b3}
they can be received, for example, in different ways:
{a1, a2, b1, a3, b2, b3}, {a1, b1, a2, b2, b3, a2}, and other permutations where the ordering of the A group is always {a1, a2, a3} and the ordering of the B group is always {b1, b2, b3}.

The IN channel operates in "server" mode, which means that Pepa-Ng creates a listening socket and accepts connections. When an external actor connects, Pepa-Ng adds the new "reading" socket to the monitored socket list.

If the external actor disconnects, Pepa-Ng removes this reading socket from the list of monitored reading sockets. However, it is essential to note that Pepa-Ng will never close the listening IN socket.

Summarising, the Pepa-Ng IN channel always has one listening socket and {0, N} reading socket, where N is configurable. See "Command Line Options" for the configuration details.

The OUT channel

The OUT channel is internally represented by two sockets: one listening and one writing socket. The listening socket is dedicated to accepting incoming connections. The writing socket is dedicated to forwarding messages from the Shva server to the external actor. An external actor always communicates with the writing socket; the information about the listening socket is only given here to explain the Pepa-Ng internal implementation.

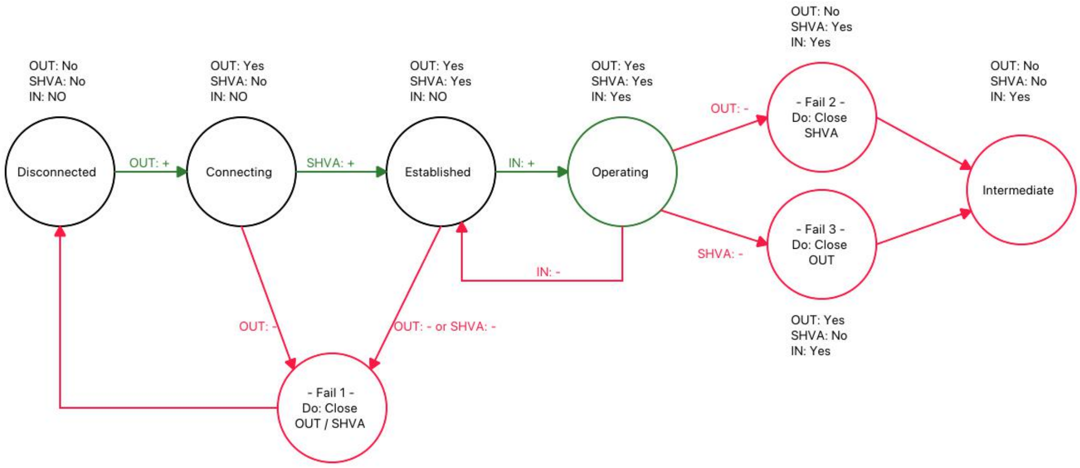
It is important to note that the OUT channel does not support multiple writing sockets. If an additional actor tries to connect, this connection attempt is ignored. However, Pepa-Ng will not terminate this additional connection in this case, so the external actor might wait an undefined period. Nevertheless, this additional connection may be accepted after the existing external actor is disconnected and the existing single writing socket is closed. This situation is not designed and defined, so you can not rely on the Pepa-Ng behavior since it can be changed in future implementation. Just do not try to open multiple connections to the OUT channel.

The OUT channel operates as a server where Pepa-Ng starts a listening socket and waits for an external actor to connect. Once connected, the Pepa-Ng state machine is ignited.

If the OUT channel actor gets disconnected, Pepa-Ng will terminate the writing socket, disconnect Shva, and return to a waiting state until a new actor connects to OUT.

It is important to note that the OUT listening socket will remain open and never be closed. Pepa-Ng only creates and terminates the write socket. While the writing socket can be created and destroyed, the listening socket always awaits incoming connections.

State Machine of Pepa-Ng


OUT: No
SHVA: No
IN: NO
Disconnected
OUT: +
OUT: Yes
SHVA: No
IN: NO
Connecting
SHVA: +
OUT: Yes
SHVA: Yes
IN: NO
Established
IN: +
OUT: Yes
SHVA: Yes
IN: Yes
Operating
OUT: -
OUT: No
SHVA: Yes
IN: Yes
- Fail 2 -
Do: Close
SHVA
OUT: No
SHVA: No
IN: Yes
Intermediate
SHVA: -
- Fail 3 -
Do: Close
OUT
OUT: Yes
SHVA: No
IN: Yes
IN: -
OUT: -
OUT: - or SHVA: -
- Fail 1 -
Do: Close
OUT / SHVA

Command Line Options of Pepa-Ng

Mandatory Configuration

Long	*Short*	*Arg*	
--shva	-s	IP:PORT	Address of SHVA server to connect to in form: '1.2.3.4:7887'
--out	-o	IP:PORT	Address of OUT listening socket, waiting for OUT stream
--in	-i	IP:PORT	Address of IN listening socket, waiting for OUT stream connection, in form '1.2.3.4:3748'

Optional Configuration

Long	*Short*	*Arg*	
--inum	-n	N	Max number of IN clients, by default 1024
--bsize	-b	N	Size of SEND / RECV buffer, in Kb; if not given, 64 Kb used
--daemon	-w		Run as a daemon process; WARNING: When --daemon is ON, it disables terminal printings and disbales colors
--pid	-P	File	Create PID file 'file'; full path should be provided. By defailt, /tmp/pepa.pid file created
--log	-l	N	Logger level, accumulative: 0 = no log, 7 includes also {1-6}; Default level is 7 (everything) 0: none, 1: fatal, 2: trace, 3: error, 4: debug, 5: warn, 6: info, 7: note The same log level works for printing onto the terminal and to the log file
--file	-f	file	Name of log file to save the log into
--dir	-d	dir	Name of directory to save the log into
--noprint	-p		DO NOT print log onto the terminal. By default, it will be printed
--dump	-u		Dump every message into the log file
--color	-c		Enable color in terminal printings (always disabled in 'daemon' mode)
--monitor	-c	N	Run monitor. It will print status every N seconds; default output in bytes
--divider	-r	N|C	Monitor units: it can be a character (C) or a number (N): ~ The character argument to print units: 'b' for Bytes, 'k' for Kbytes, 'm'
--version	-v		Show version + git revision + compilation time
--help	-h		Show this help

Emulation Configuration (testing only)

Long	*Short*	*Arg*	
--emusleep	-S	N	Sleep N microseconds between buffer sending; 0 by default
--emubuf	-B	N	Max size of a buffer, N must be, >= 1; It is 1024 by default
--emubufmin	-M	N	Min size of a buffer, N must be; >= 1; It is 1 by default

Embedded Options of Pepa-Ng

The Pepa-Ng software has a range of preset values, which are typically stored in the pepa_config.h file. These values are utilized during the initialization process of various Pepa-Ng options when the software is launched. In case a value isn't defined as a prompt option, the built-in values are used as a fallback.

Implementation Notes

Internally, Pepa-Ng is designed as a one-threaded loop, based on the epoll system call. It also can run an additional thread called “monitor,” but it is an optional behavior.

When the epoll signalizes that a socket requires attention, the Pepa-Ng processes all socket descriptors, tests them for errors, and reads all buffers from sockets. Every buffer is immediately forwarded to its destination.

Internal Buffering

Pepa-Ng does not implement internal buffering of received packets. Every packet is immediately forwarded to its destination.
If a failure happens, and reading/writing sockets are closed, the Pepa-Ng will try to drain all reading IN sockets before closing them, i.e., it will read all buffers and drop them. This behavior helps to avoid the situation of dangled sockets.

Logging Options

The Pepa-Ng uses an external “slog” logger to print out statistics. This logger is released as an Open Source code. The Pepa-Ng code includes the slog code as a part of it and does not rely on an external GIT repository.

Testing: Emulator And Its Usage

A dedicated Emulator was developed to test Pepa-Ng. The Emulator mimics the Shva server and connects to the IN and OUT channels. The Emulator does not produce actual packets; it creates buffers of different sizes (see Emulation Configuration for details), fills them with text data, and creates a buffer stream.

The Emulator also emulated sockets disconnection to test how the Pepa-Ng state machine handles these situations.

Testing: Build-In Monitor

A built-in monitor is a part of Pepa-Ng, initially created for testing. However, it is kept as a part of production as a convenient tool.
The Monitor can be enabled when Pepa-Ng is started; see "Command Lines." There are several configuration options the user can use.

Here is an example of the Monitor output:

```
### STATUS: Freq: 3 seconds, Units: Kbytes ### SHVA [+10346 | 3448/sec] ---> OUT [+10346 | 3448/sec] ###
IN [+37661 | 12553/sec] ---> SHVA [+37661 | 12553/sec]   ###
### STATUS: Sockets: OUT LISTEN FD[1]: 5 | OUT WRITE FD[1]: 6 | SHVA FD[1]: 7 | IN LISTEN FD[1]: 8 | IN
ACCEPTORS[4] ###
```

There are two lines per Monitor iteration.

The first line shows how much data per second was transferred between Pepa-Ng channels. This line starts with two variables: "Freq," which indicates how often the monitor prints statistics. The second variable, "Units," describes the units in which the Monitor reports the statistics. The units can be configured on the Pepa-Ng start. The predefined units are "Bytes, Kbytes, Mbytes." For example, the user can also use their units if they wish to report how many chanks of 100 bytes are passed.

The statistics are printed in the form "+N | N/sec." The first, "+N," is the absolute value of how many data units are transferred between the Monitor iteration. In the example above, you can see that during 3 seconds, the Pepa-Ng transferred 10346 Kbytes from SHVA to OUT channel and 37661 Kbytes from into SHVA.

The second value, "N/sec," shows how many units were transferred through the channel per second. In the example above, 10346 (Kb) / 3 (seconds) = 3448/sec from SHVA to OUT.

The second line shows the status of the Pepa-Ng sockets. The Pepa-Ng prints out the numbers of file descriptors of opened sockets; if a socket is closed, it will be a "-1" value. The last in this string, "ACCEPTORS[N]," is the number of IN reading sockets. This example means there are 4 external actors connected to the IN stream, so Pepa-Ng opened 4 reading sockets.

Pepa-Ng Compilation

The Pepa-Ng can be compiled as a dynamically linked or static object.

```
# make clean - clean all compiled objects and targets
# make pepa - compile pepa-ng
# make emu - compile emulator
# make static - compile static Pepa-Ng
```

The default “make” command equals “make clean static.”