Slow Watchdog Design

TABLE OF CONTENTS

[Design Ideas 1](#_Toc72871084)

[Watchdog.service 1](#_Toc72871085)

[Watchdog file 2](#_Toc72871086)

[Epoch1970’s Design 2](#_Toc72871087)

[Watchdog.conf 3](#_Toc72871088)

[Epoch1970 Design Concept 5](#_Toc72871089)

[Slow Watchdog Design for PowerPi 6](#_Toc72871090)

[Design Concept 6](#_Toc72871091)

[State Diagram 6](#_Toc72871092)

[Wdtest 6](#_Toc72871093)

[Wdrepair 6](#_Toc72871094)

[Watchdog.conf 6](#_Toc72871095)

[Watchdog file 6](#_Toc72871096)

# Design Ideas

Making a slow-reacting watchdog from the fast-reacting watchdog of the Raspberry Pi

Posting by Epoch1970 on [Raspberry Pi Forum](https://www.raspberrypi.org/forums/viewtopic.php?f=29&t=151179).

## Watchdog.service

The /lib/systemd/system/watchdog.service file on the powerpi looks like:

|  |
| --- |
| [Unit]  Description=watchdog daemon  Conflicts=wd\_keepalive.service  After=multi-user.target  OnFailure=wd\_keepalive.service  [Service]  Type=forking  EnvironmentFile=/etc/default/watchdog  ExecStartPre=/bin/sh -c '[ -z "${watchdog\_module}" ] || [ "${watchdog\_module}" = "none" ] || /sbin/modprobe $watchdog\_module'  ExecStart=/bin/sh -c '[ $run\_watchdog != 1 ] || exec /usr/sbin/watchdog $watchdog\_options'  ExecStopPost=/bin/sh -c '[ $run\_wd\_keepalive != 1 ] || false'  [Install]  WantedBy=default.target |

Epoch1970 recommends that we replace the [Install] section:

|  |
| --- |
| [Install]  WantedBy=multi-user.target |

## Watchdog file

The post next specifies a /etc/default/watchdog file that doesn’t work well (as noted in the post). Here is the /etc/default/watchdog that is running on the *powerpi* host.

|  |
| --- |
| # Start watchdog at boot time? 0 or 1  run\_watchdog=1  # Start wd\_keepalive after stopping watchdog? 0 or 1  run\_wd\_keepalive=1  # Load module before starting watchdog  watchdog\_module="none"  # watchdog\_module="bcm2835\_wdt" (epoch1970 version)  # Specify additional watchdog options here (see manpage).  # -q is no-act. for debug  # watchdog\_options=" -v" |

Options

* -v –verbose
* -s –sync. Try to sync the file system every time watchdog is called
* -b –softboot
* -F –foreground, run in foreground mode, useful with systemd
* -f –force, force the value of interval, ignoring sanity checks
* -c config-file –config-file config-file, instead of /etc/watchdog.conf
* -q –no-action

## Epoch1970’s Design

The bcm2835\_wdt watchdog module wants to read the watchdog device at least every 15 secs, so the watchdog daemon should write at something like twice that frequency to be safe. I've used 10 secs for the watchdog program and 15 secs for the hardware watchdog. Don’t use anything higher than 15seconds for the hardware watchdog, as it breaks things.

Epoch1970 was interested in rebooting if temperature goes critical (low probability), if a process goes haywire (low probability), if networking goes wrong (probable). In the first 2 cases, instant reboot (i.e., within 15 secs) is right for Epoch1970. For the last case, I'd rather let things settle for tens of minutes before rebooting. He doesn't want to ping any local or internet destination every 10 secs, a full minute is better.

To achieve this Epoch1970 recommends this basic design:

* Added a pinger script that gets daemonized at boot, pings the LAN gateway every minute, and writes the current uptime in a file in /run when the ping succeeds. When the ping fails, the file is untouched.
* Setup watchdog.conf to watch that file and react if it wasn't updated for 30 minutes
* Added a repair script that returns 1 (reboot) by default, but also manages a false alarm thrown by watchdog when system date has changed. Watchdog looks to see if the special file is more than 30 minutes old, and when the system time syncs after boot, suddenly this condition can happen. It's ironic because if time changes, in my case it means networking is fine and the system has acquired a good NTP clock and decides to sync system time... rebooting at that moment is really counter-productive.
* The solution to detect the false alarm is to use a monotonic clock instead of the date. The simplest source I've found was to "cat /proc/uptime" and write the uptime in the special file.
* The repair script, when called, gets uptime and compares it to the one written in the file. If the diff is about 1 minute, then this is a false alarm. If the diff looks like 30 minutes, then it is a valid alarm.

## Watchdog.conf

Relevant parts of /etc/watchdog.conf:

|  |
| --- |
| watchdog-device = /dev/watchdog  watchdog-timeout = 15  interval = 10  logtick = 180  realtime = yes  priority = 1  file = /run/wdpinger-gw  change = 1800  temperature-device = /sys/class/thermal/thermal\_zone0/temp  max-temperature = 75  max-load-5 = 18  max-load-15 = 12  min-memory = 5  allocatable-memory = 1  repair-binary = /usr/local/bin/wdrepair.sh  repair-timeout = 60 |

This lazily reboots if the network is not reachable for 30 minutes straight, doesn’t stress the machine or the network too much, and keeps the benefits of the fast-reacting hardware watchdog for critical over-load or over-temperature conditions.

We need to integrate these entries into the current powerpi /etc/watchdog.conf:

|  |
| --- |
| #ping = 172.31.14.1  #ping = 172.26.1.255  interface = wlan0  #file = /var/log/messages  #change = 1407  # Uncomment to enable test. Setting one of these values to '0' disables it.  # These values will hopefully never reboot your machine during normal use  # (if your machine is really hung, the loadavg will go much higher than 25)  max-load-1 = 24  #max-load-5 = 18  #max-load-15 = 12  # Note that this is the number of pages!  # To get the real size, check how large the pagesize is on your machine.  #min-memory = 1  #allocatable-memory = 1  #repair-binary = /usr/sbin/repair  #repair-timeout = 60  #test-binary =  #test-timeout = 60  # The retry-timeout and repair limit are used to handle errors in a more robust  # manner. Errors must persist for longer than retry-timeout to action a repair  # or reboot, and if repair-maximum attempts are made without the test passing a  # reboot is initiated anyway.  #retry-timeout = 60  #repair-maximum = 1  watchdog-device = /dev/watchdog  watchdog-timeout = 15  # Defaults compiled into the binary  #temperature-sensor =  #max-temperature = 90  # Defaults compiled into the binary  #admin = root  #interval = 1  #logtick = 1  log-dir = /var/log/watchdog  # This greatly decreases the chance that watchdog won't be scheduled before  # your machine is really loaded  realtime = yes  priority = 1  # Check if rsyslogd is still running by enabling the following line  #pidfile = /var/run/rsyslogd.pid |

## Epoch1970 Design Concept

The premise is that the watchdog program never fails to run and is never killed by the OS. Running it real-time is important (esp. with a 10 sec. frequency). The machine starts watchdog and a separate daemon called "wdtest" at boot. Wdtest watches processes and refreshes regularly a few keepalive files, Epoch1970 has it running its test loops every minute. It has a keepalive file for itself (/run/wdtest), so that watchdog can call the repair program in case the test daemon was killed.  
  
Let's imagine the ntpd process goes poof (not unlikely): wdtest stops doing "cat $UPTIME > /run/wd-ntp" regularly. The file ages. Once the file is 3600 seconds old, watchdog calls "/usr/local/bin/dom2\_watchdog/wdrepair.sh 250 /run/wd-ntp". (250 is the watchdog error code for file out-of-date. On older systems you might receive code -6 instead.)

The wdrepair script gets the current uptime and checks if the uptime stored in /run/wd-ntp is about 1hr. in the past:

* If less than that, it means watchdog was triggered by a system date change (by NTP probably). It touches the flag file to avoid being called again in 10 seconds for the same, false, reason by watchdog. Then it exits 0. With exit 0, watchdog is happy with the repair and will not reboot.
* If the uptime delta is indeed 1hr, it means the test loop decided many times against updating /run/wd-ntp because it couldn't find the process. It's dead and cold. So, the repair script stops the ntp service (to please systemd), runs ntpdate (to step the clock if needed), restarts the service, and then exits 0.
* If the repair program chooses to exit 1 (anything other than 0 indeed), watchdog reboots immediately. If the repair program fails to return within 120 seconds, watchdog reboots the machine.

In the Epoch1970 repair script the only condition that triggers reboot (it runs "sync" and exits 1) is if wdtest was unable to ping the LAN gateway for half an hour consecutively. Wdtest runs every minute, and it runs the LAN ping test every time it loops. So, it only reboots if pinging the gateway has failed 30 times in a row.

# Slow Watchdog Design for PowerPi

## Design Concept

Watchdog will monitor the powerpi host for failures of any of these services:

* Wi-Fi access by pinging the gateway at 192.168.74.1
* Network access by pinging a google end point (8.8.8.8)
* Load averages that exceed a threshold for a given time

On failure of a ping test, a repair script is configured that attempts to reset the wi-fi service (wlan0) to restore the ability to ping the gateway and the google end point. Note that the interface testing capability of watchdog is not used because traffic is so light on this host that it continuously reports insufficient traffic and triggers a reboot.

Consider using the epoch1970 method of pinging the LAN gateway with a separate test program if testing with the simpler watchdog ping test is not successful.

## State Diagram

## Testing

Installation tests

1. Check content of /lib/systemd/system/watchdog.service file
   1. Enable watchdog service: sudo systemctl enable watchdog
2. Check to see if the /etc/default/watchdog file exists and compare its contents to the desired watchdog file contents.
3. Start with all tests disabled in /etc/watchdog.conf. Set only these parameters:

|  |
| --- |
| watchdog-device=/dev/watchdog  watchdog-timeout=15  interval=1  logtick=180  realtime=yes  priority=1  admin=pi |

1. Check watchdog service on startup behavior
   1. Modify the /etc/default/watchdog file with run\_watchdog=0.
   2. Reboot and validate that the watchdog service does not start.
   3. Modify the /etc/default/watchdog file with run\_watchdog=1.
   4. Reboot and validate that the watchdog service does start.
   5. Set /etc/default/watchdog watchdog\_options=“ -v -q”
      1. Validate that watchdog log is verbose in /var/sys/syslog
      2. Validate that watchdog failures do not cause a reboot, but do log an event that would reboot if -q option is not set
      3. Validate that the pi user gets email when a reboot event occurs.

Configuration tests

1. Max load settings

|  |
| --- |
| max-load-1 = 24  max-load-5 = 18  max-load-15 = 12 |

Run high load script for 1, 5 and 15 minutes and verify that the max load settings operate properly. After reboot occurs running high load for 1 minute, comment max-load-1 and run high load script again. After reboot occurs running high load for 5 minutes, comment max-load-5 and repeat to test 15-minute load test. Once all load tests succeed in rebooting, un-comment the three settings.

1. Repair script

|  |
| --- |
| repair-binary = /usr/local/bin/wdrepair.sh  repair-timeout = 60  retry-timeout = 60  repair-maximum = 1 |

Run the high load script and verify that the wdrepair.sh is triggered to run.

With a wdrepair.sh that runs forever without returning, verify that the repair-timeout of 60 seconds does trigger a reboot after a minute.

With a wdrepair.sh that does nothing but return 1, verify that the reboot occurs immediately after wdrepair.sh runs after a repair-maximum of 1 try.

Validate that setting repair-maximum to 10 causes the wdrepair.sh that returns 1 is run 10 times before a reboot.

With a wdrepair.sh script that kills the high load script and checks that load average is reduced and then returns 0, validate that a reboot is avoided.

1. Test and repair scripts  
   These scripts are all located in /etc/watchdog.d directory and are detected by watchdog when it restarts. Watchdogd runs these scripts at the same frequency as those that are run as configured in watchdog.conf. The scripts are called with the parameter ‘test’ and if other than ‘0’ is returned, the scripts are called again with the parameter ‘repair’ and the return value from calling the script with the ‘test’ parameter. If other than ‘0’ is returned, watchdogd reboots.  
     
   The standard output and error output of these scripts is logged in the /var/log/watchdog folder with a log file for each script.  
   1. LAN test and repair – wdlan.sh  
      Operation  
      Checks to see if the LAN has been reachable at 192.168.74.1 within the last 5 minutes and resets up timer if reachable. If LAN is down, attempt to restart LAN service. If LAN is back up, return 0. If LAN fails to come up for 60 minutes, requests reboot.  
        
      Tests for wdlan.sh  
      Validate in logs that wdlan.sh checks the LAN service and resets the timer while it is up to show the last time up in the /run/wdlan file, returning 0 to watchdog.   
        
      Remove network connection to simulate loss of LAN connection.  
        
      Validate in logs that wdlan.sh does not reset the timer and returns 1 to watchdog after being called to ‘test’. When wdlan.sh is called to repair with error 1, reports the last LAN uptime from /run/wdlan no more frequently than once per minute and if less than 5 minutes, returns 0 to watchdog.  
        
      Validate in the logs that wdlan.sh is called to repair with error 1 to restart LAN service once every 5 minutes and tracks restart attempts in /run/wdlancount, returning 0 to watchdog.  
        
      Validate in the logs that wdlan.sh is called to repair with error 1 and returns non-zero to watchdog if LAN service has failed to start for 60 minutes (12 restart attempts).  
        
      Validate that watchdog reboots after an hour of LAN downtime.  
        
      Restore the network connection to simulate recovery of LAN connection.  
        
      Validate in logs that wdlan.sh either finds the service up and resets the timer in /run/wdlan, returning 0 to watchdog, or wdlan.sh is called with repair and error 1 and restarts the LAN service successfully to validates successful connection to the LAN gateway, returning 0 to watchdog.
   2. WAN test and repair – wdwan.sh  
      Operation  
      Checks to see if the WAN at 8.8.8.8 has been reachable within the last 10 minutes and resets up timer if reachable. If WAN is down, check that LAN service is up. If it is down, return 0 and allow wdlan.sh to repair the LAN problem. If WAN comes up, return 0. If WAN cannot be reached for 60 minutes while LAN is up, requests reboot.  
        
      Tests for wdwan.sh  
      During LAN tests, validate that wdwan.sh detects that LAN is not reachable and returns 0 to watchdog to allow wdlan.sh to get LAN working first.  
        
      Validate in logs that wdwan.sh called with ‘test’ parameter returns 0 if WAN is reachable and records the current time in /run/wdwan.  
        
      Block access to WAN address 8.8.8.8.  
        
      Validate in logs that wdwan.sh called with ‘test’ parameter returns 1 if WAN is not reachable and does not update /run/wdwan.  
        
      Validate in logs that wdwan.sh called with ‘repair 1’ parameter reports the time since last success in /run/wdwan no more than once per minute and returns 0 while the time since last success is less than 10 minutes.  
        
      Validate in logs that wdwan.sh called with ‘repair 1’ parameter attempts to restart the WAN service once every 10 minutes. If successful and WAN endpoint can be reached, update success time in /run/wdwan and return 0 to watchdog. If unsuccessful, return 0 to watchdog and update restart attempt count in /run/wdwancount.  
        
      Validate in logs that wdwan.sh called with ‘repair 1’ parameter notes attempt count has reached 6 (60 minutes of downtime) that is not LAN downtime and returns 1 to watchdog.  
        
      Validate that watchdog reboots after 60 minutes of LAN up but WAN down time.  
        
      Open access to WAN address 8.8.8.8  
        
      Validate that wdwan.sh called with ‘test’ parameter returns 0 and updates last success time in /run/wdwan and resets restart count in/run/wdwancount.
   3. Adafruit.io test and repair – wdadafruitio.sh

Operation

Checks to see if adafruit.io has been reachable within the last 60 minutes and resets up timer if reachable. If it has not been reachable and if LAN is down, returns 0 to allow wdlan.sh to repair. If WAN is down, returns 0 to allow wdwan.sh to repair. If they are both up, log the failure to reach adafruit.io and return 0. Never requests a reboot for this failure.  
  
Tests for wdadafruitio.sh

* 1. Rfopowerd test and repair – wdrfopower.sh  
     Operation: Checks to see if the rfopowerd daemon has been running within the last 5 minutes, and if it has not been running for more than 5 minutes, restarts the rfopowerd. If rfopowerd comes up, return 0, else if rfopowerd fails to come up after 5 attempts over 25minutes, requests reboot.

## 

## Watchdog file

/etc/default/watchdog

|  |
| --- |
| # Start watchdog at boot time? 0 or 1  run\_watchdog=1  # Start wd\_keepalive after stopping watchdog? 0 or 1  run\_wd\_keepalive=1  # Load module before starting watchdog  watchdog\_module="bcm2835\_wdt"  # debug options  watchdog\_options=" -v -q" |

Options

* -v –verbose
* -s –sync. Try to sync the file system every time watchdog is called
* -b –softboot
* -F –foreground, run in foreground mode, useful with systemd
* -f –force, force the value of interval, ignoring sanity checks
* -c config-file –config-file config-file, instead of /etc/watchdog.conf
* -q –no-action

## Watchdog.service

/lib/systemd/system/watchdog.service

|  |
| --- |
| [Unit]  Description=watchdog daemon  Conflicts=wd\_keepalive.service  After=multi-user.target  OnFailure=wd\_keepalive.service  [Service]  Type=forking  EnvironmentFile=/etc/default/watchdog  ExecStartPre=/bin/sh -c '[ -z "${watchdog\_module}" ] || [ "${watchdog\_module}" = "none" ] || /sbin/modprobe $watchdog\_module'  ExecStart=/bin/sh -c '[ $run\_watchdog != 1 ] || exec /usr/sbin/watchdog $watchdog\_options'  ExecStopPost=/bin/sh -c '[ $run\_wd\_keepalive != 1 ] || false'  [Install]  WantedBy=multi-user.target |

## Watchdog.conf

/etc/watchdog.conf

|  |
| --- |
| # Uncomment to enable test. Setting one of these values to '0' disables it.  # These values will hopefully never reboot your machine during normal use  # (if your machine is really hung, the load average will go much higher than 25)  max-load-1 = 24  max-load-5 = 18  max-load-15 = 12  # Note that this is the number of pages!  # To get the real size, check how large the page size is on your machine.  #min-memory = 1  #allocatable-memory = 1  repair-binary = /usr/local/bin/wdrepair.sh  repair-timeout = 60  #test-binary =  #test-timeout = 60  # The retry-timeout and repair limit are used to handle errors in a more robust  # manner. Errors must persist for longer than retry-timeout to action a repair  # or reboot, and if repair-maximum attempts are made without the test passing a  # reboot is initiated anyway.  retry-timeout = 60  repair-maximum = 1  watchdog-device = /dev/watchdog  watchdog-timeout = 15  interval = 1  logtick = 180  realtime = yes  priority = 1  # Defaults compiled into the binary  #temperature-sensor =  #max-temperature = 90  # Defaults compiled into the binary  admin = pi  log-dir = /var/log/watchdog  # Check if rsyslogd is still running by enabling the following line  #pidfile = /var/run/rsyslogd.pid  #file = /run/wdpinger-gw  #change = 1800 |

## wdrepair.sh

Installed at /usr/local/bin/wdrepair.sh

This program is named by the repair-binary parameter and is the program that is called by the watchdog daemon whenever it decides a reboot is needed. If wdrepair.sh returns '0' indicating that it successfully repaired the problem, watchdog does not reboot and continues. If it goes through this process more times than ***repair-maximum*** times, then it reboots anyway. If the ***repair-binary*** program returns other than '0', watchdog reboots. If the ***repair-binary*** takes longer than ***repair-timeout*** seconds, watchdog reboots.

## High load test - highload.sh

Test script that creates a very high load average almost immediately on the processor.

LAN test and repair – wdlan.sh  
Operation: Checks to see if the LAN has been reachable at 192.168.74.1 within the last 5 minutes and resets up timer if reachable. If LAN is down, attempt to restart LAN service. If LAN is back up, return 0. If LAN fails to come up for 60 minutes, requests reboot.

WAN test and repair – wdwan.sh  
Operation: Checks to see if the WAN at 8.8.8.8 has been reachable within the last 10 minutes and resets up timer if reachable. If WAN is down, check that LAN service is up. If it is down, return 0 and allow wdlan.sh to repair the LAN problem. If WAN comes up, return 0. If WAN cannot be reached for 30 minutes, requests reboot.

## Adafruit.io test and repair – wdadafruitio.sh

Operation: Checks to see if adafruit.io has been reachable within the last 60 minutes and resets up timer if reachable. If it has not been reachable and if LAN is down, returns 0 to allow wdlan.sh to repair. If WAN is down, returns 0 to allow wdwan.sh to repair. If they are both up, log the failure to reach adafruit.io and return 0. Never requests a reboot for this failure.

Rfopowerd test and repair – wdrfopower.sh  
Operation: Checks to see if the rfopowerd daemon has been running within the last 5 minutes, and if it has not been running for more than 5 minutes, restarts the rfopowerd. If rfopowerd comes up, return 0, else if rfopowerd fails to come up after 5 attempts over 25minutes, requests reboot.

## Troubleshooting