# Pdnsd

From ArchWiki

pdnsd (http://members.home.nl/p.a.rombouts/pdnsd/index.html) is a DNS server designed for local caching of DNS information. Correctly configured, it can significantly increase browsing speed on a broadband connection. Compared to bind or dnsmasq it can remember its cache after a reboot; "p" stands for persistent.

## Contents



## Installation

Install `pdnsd (https://www.archlinux.org/packages/?name=pdnsd)` from the official repositories.

## Configuration

## Initial preparation

The sample configuration file that comes with pdnsd needs a few changes before the daemon can start.

## Format

The `pdnsd.conf` file uses a fairly simple format, but it has some differences from most other configuration files you have likely encountered. It has a collection of sections of various types. A section is started with the name of the type of section and an opening curly bracket (**{**) and is ended by a closing curly bracket (**}**). Sections cannot be nested.

Inside each block is a series of options of the following format:

```
option_name=option_value;
```

Notice the semicolon at the end; unlike some formats, it is not optional.

Comments are started with either **#** or **/***. The former goes to the end of the line, the latter continues until it reaches ***/**.

## DNS servers

pdnsd needs to know the address of at least one DNS server to collect DNS information from. This part of the setup differs depending on whether you have a broadband connection or dial-up. Broadband users should use the first server section as a starting point, dial-up users the second, leaving the other server sections commented out.

label
: The `label` option is used to uniquely identify a server section. It is completely arbitrary, but one good choice is the name of your ISP.

ip
: This option, used in the default broadband configuration, tells pdnsd the addresses of DNS servers to use. Multiple addresses should be separated by a single comma, with optional whitespace before or after the comma. You can just copy the addresses from `/etc/resolv.conf` .

file
: The `file` option can be used instead of `ip` to specify a set of DNS server IPs. Its value is the path to a file with servers listed in `resolv.conf` format. The default dial-up configuration uses it because the PPP client writes `/etc/ppp/resolv.conf` with the addresses it gets from the PPP server. You should not need to change it unless you want to use a different DNS server than your ISP gives you by default.

The rest of the server section will work without any more changes. For details on all the available options, see the pdnsd manual (http://members.home.nl/p.a.rombouts/pdnsd/doc.html).

### DNS servers with DHCP connections

When netctl is installed, pdnsd can be notified of the ip addresses of the name servers by resolvconf (see resolvconf(8) man page) and the notifications become dynamic when you use Automatic switching of profiles.

To configure this feature, remove the broadband server section and update the dial-up server section with the following changes:

```
  label = resolvconf;
  file = /etc/pdnsd-resolv.conf;
```

Edit `/etc/resolvconf.conf` to configure resolvconf with pdnsd as one of its subscribers:

```
  name_servers=127.0.0.1
  pdnsd_resolv=/etc/pdnsd-resolv.conf
```

And run `resolvconf -u` to update `/etc/pdnsd-resolv.conf` with the addresses of the name servers (ignore the error message saying that the pdnsd socket cannot be accessed). This updating is only needed once before starting manually pdnsd.

## OpenDNS

The `pdnsd.conf` file comes with OpenDNS settings built in; you can simply remove (or comment out) the dialup and broadband sections above it (being careful not to remove the necessary global setup at the very top of the file), and then uncomment it to use OpenDNS resolution.

However, OpenDNS does some weird things to Google. You need to deny results from OpenDNS that return one of OpenDNS's Google-proxy machines if you want to avoid this behaviour (for many people, it can increase Google requests from, say, 15ms, to 75ms+). The exact servers' IPs change, but you can run an `drill www.google.com @208.67.222.222` (provided by `dnsutils` (https://www.archlinux.org/packages/?name=dnsutils)) to find the current IPs. You'll know if the query is being proxied, because the server's name will resolve to something like `google.navigation.opendns.com`. For me, these addresses were `208.67.216.230` and `208.67.216.231`.

Once you know the IPs, you can replace the `pdnsd.conf`'s already-existant `rejected` IPs inside the OpenDNS `server { … }` declaration. Make sure you retain the prefixes.

OpenNIC (http://wiki.opennicproject.org/Tier2) is a reliable alternative to OpenDNS.

### Testing

You should now have a working `pdnsd` daemon. Start it.

You can test it with the `drill` utility (from the `ldns` (https://www.archlinux.org/packages/?name=ldns) package):

```
$ drill www.google.com @127.0.0.1
```

If everything works, you should see a list of IP addresses associated with Google.

For the second time you look up the address, query time should be under 1 ms.

### System setup

Make sure to enable the `pdnsd` service.

It starts right after `network.target`, as services that use the network rely on a working DNS, i.e. `network-online.target` (see: Upstream Wiki (http://www.freedesktop.org/wiki/Software/systemd/NetworkTarget/)).

## Tips and Tricks

### Performance settings for home broadband users

Many users have broadband connections where the DNS server is slow or unreliable, and would like to use `pdnsd` as a caching server to minimize the number of DNS queries that need to be made. After doing the setup detailed above, the following settings in the `/etc/pdnsd.conf` will help improve the performance in this role:

Under global settings:

```
neg_rrs_pol=on;
par_queries=1;
```

Under server settings:

```
proxy_only=on;
purge_cache=off;
```

The `neg_rrs_pol=on;` policy means that when a negative response comes back for a query, pdnsd server will still cache the result even if the response is not "authoritative". This is important since watching DNS queries will reveal that there are many requests for AAAA records (DNS queries for IPv6) which will never return results since many domains are not using IPv6, as well as MX records since not every domain has an MX record. Without the negative caching, these requests will be sent

even after a domain name has been cached, and in this role you do not want the extra DNS requests being made. It is important to use this option in conjunction with the `proxy_only=on;` option to minimize the number of queries coming out of the system.

The `par_queries=1;` option is useful if you specify more than one DNS server in your "server" section below. It specifies an increment of how many parallel queries will be made at once. For example, if four DNS servers are listed in the "server" section, and `par_queries=2;` (the default), then the first 2 servers will be queried simultaneously, and if both of the first two servers fail, `pdnsd` will move on to the next two and query them simultaneously. The setting used above means that one DNS server at a time gets queried, so you can list two or more DNS servers in the "server" section, and the second one will only be queried if the first one fails. This helps minimize traffic, but if the first server fails you will have to wait through the timeout before the second server will be queried. Tweak this setting for your own preferences, and if you only specify one server in the "server" section then you do not need to worry about it.

The `proxy_only=on;` setting is mentioned below in the FAQ and is important for home broadband users since you generally are using only one or two DNS servers instead of trying to do the full-blown hierarchical name resolution that a full DNS server would do. This setting will prevent `pdnsd` from resolving all the way back to the "authoritative" name server, and instead accept the results of the DNS servers that were already specified in the "server" section. Once again, this reduces the number of DNS queries you need to make, improving performance.

The `purge_cache=off;` setting tells `pdnsd` not to remove cache entries even if they have outlived the DNS record's time-to-live metric. This can be very useful when your ISP's DNS server goes down and you want to be able to access name lookups for domains you frequently use despite the outage. Records will still be bumped out of the cache based on age once the cache becomes full (see `man pdnsd.conf` on how to set the size of the cache).

## Additional performance settings

### TTLs (Time-To-Live)

Each DNS resource record returned from a server includes a maximum time-to-live, or TTL. This tells the recipient how long to store the record and when to do a new lookup on it. Many DNS records have relatively short TTLs, such as 3600 (one hour). This means that after one hour, pdnsd will attempt a new lookup on this entry, regardless of whether it has a cached record for it available. It will improve performance to override this default TTL by setting a global minimum TTL, causing fewer lookups to be performed. The disadvantage to using a minimum TTL that is too long is that a cached record may be out of date (the IP address of the host may be changed, but your client will not know this because it will receive the cached address). However, most IP addresses do not change hourly or even daily.

Times are specified in seconds by default, or you may append an "m", "h", "d", or "w" to the time to specify minutes, hours, days, or weeks.

`min_ttl` in the global settings sets a minimum TTL for cached records, causing pdnsd to ignore the default TTL in the record received from the server. On a slow connection or with a slow DNS server, you may want to set this to several hours to reduce the number of lookups ( eg `min_ttl=6h;` ).

`neg_ttl` in the global settings sets a minimum TTL for non-existent domains. If a server tells pdnsd that a domain does not exist, it will not try to lookup that domain again until this amount of time has elapsed.

### Timeouts

Setting shorter timeouts means that pdnsd will give up on an entire query or a given server query more quickly, resulting in faster performance. The disadvantage to setting timeouts too short is that pdnsd might return an error on a lookup simply because the server was not given enough time to respond.

`timeout` in the global settings determines when pdnsd gives up on an entire query and returns an error to your browser or other client. Setting the global timeout option makes it possible to specify quite short timeout intervals in the server sections (see below). This will have the effect that pdnsd will start querying additional servers fairly quickly if the first servers are slow to respond (but will still continue to listen for responses from the first ones). (If you use query_method=tcp_udp it is recommended that you make the global timeout at least twice as large as the largest server timeout, otherwise pdnsd may not have time to try a UDP query if a TCP connection times out.)

`tcp_qtimeout` in the global settings determines how long a TCP query connection may be left open.

`timeout` in the server settings determines how long pdnsd will wait for a response from each server. Setting this to a shorter time means that pdnsd will give up on a non-responsive server more quickly and will move on to the next available server, sometimes resulting in a faster overall response time. On a fast connection, setting this to 4 or 5 seconds is not unreasonable.

### Debugging

To see what servers pdnsd is using for a particular lookup, how timeouts are working, and what default TTLs are being used by domains, turn debug on in the global settings:

```
debug=on;
```

Restart pdnsd and monitor the pdnsd.service for changes with the systemd journal:

```
journalctl _SYSTEMD_UNIT=pdnsd.service
```

Be sure to turn debug off for general use as leaving it on may degrade performance.

### Cache size

By default, pdnsd will automatically create authoritative records for all entries in `/etc/hosts`. If you have a lot of entries, for example if you are using it for ad blocking, the default maximum cache size provided by `/etc/pdnsd.conf` may not be large enough, resulting in DNS requests not being cached for their expected amount of time.

To increase the cache size, edit the `perm_cache` line in the 'global settings' section of configuration file (size in kB).

Alternatively, you can prevent pdnsd from preemptively sourcing your hosts file by adding the option `authrec=off` to the 'source' section. If, for whatever reason, setting authrec to off does not work, an easy workaround is to create a separate hosts file (eg `/etc/hosts-pdnsd`) with only your system information and point your 'source' section to that instead, while leaving your original hosts file intact. This way, pdnsd will reference `/etc/hosts` only when performing lookups. So for example:

/etc/hosts-pdnsd

```
#<ip-address>   <hostname.domain.org>   <hostname>
127.0.0.1       localhost.localdomain   my_hostname
::1             localhost.localdomain   localhost
```

## Shared server for your LAN

If you have several computers on your network, you may want to make pdnsd the DNS server for them all. This allows your entire network to share a single DNS cache, making repeated lookups much faster. To allow this, simply set `server_ip` in the `global` section to the name of your network interface (usually `eth0`). If you have set up a firewall, tell it to allow connections to port 53 from any address on your network.

Now you can configure the other computers on your network to use the computer running pdns as their primary dns server.

## Name blocking

pdnsd allows you to specify hosts or domains that it should never return results for. This allows you to use it as a primitive ad or content blocker, among other things. Create a new `neg` section in `pdnsd.conf`. `neg` sections have two main options. `name` is the name of the host or domain you want to block. `types` can be set to `domain` to block all hosts in the given domain. The default `pdnsd.conf` gives an example that blocks all ads from doubleclick.net.

## pdnsd-ctl

From the pdnsd-ctl(8) manpage:

> ***pdnsd-ctl*** *controls* ***pdnsd****, a proxy dns server with permanent caching. Note that the status control socket must be enabled (by specifying an option on the pdnsd command line or in the configuration file) before you can use* ***pdnsd-ctl****.*

A couple of useful commands to get you started...

View cache:

```
# pdnsd-ctl dump
```

Flush cache:

```
# pdnsd-ctl empty-cache
```

# Troubleshooting

## result of uptest for 192.168.x.x: failed

You can successfully ping your ISP's dynamic DNS server, even though the log shows the following:

```
$ journalctl -f _SYSTEMD_UNIT=pdnsd.service
```

```
result of uptest for 192.168.x.x: failed
```

Check the interface configured in `/etc/pdnsd.conf` global section exists:

```
interface = any;
```

or the one in the server section:

```
interface=enp2s0;
```

The correct name can be found by running: `ifconfig .`

# FAQs

Q) It does not seem much faster to me. Why?

**A)** The extra speed gained from running a local DNS cache is all in how long it takes to connect to a server. Throughput, what people normally think of as speed, will not be affected. The difference is most noticeable when browsing

the web, as that typically involves small downloads from several servers. With slower connections, especially dial-up, throughput is the primary bottleneck, so there will not be as large a difference percentage-wise.

Q) Why is it so much slower now than before?

**A)** You almost certainly have the `proxy_only` option turned off in one of the server sections of `pdnsd.conf`. By default, pdnsd frequently asks several DNS servers about a domain to get the most accurate response possible. The `proxy_only` option disables this feature. It should be turned on if you use the DNS server provided by your ISP.

Retrieved from "https://wiki.archlinux.org/index.php?title=Pdnsd&oldid=355918"
Category: Domain Name System

---