OpenBGPD

bringing full views to OpenBSD since 2004

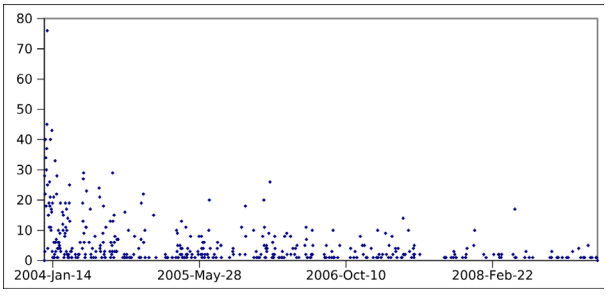
by Claudio Jeker

OpenBGPD was after OpenSSH the second "subproject" developed as part of OpenBSD but made available to a broader audience. Shortly after the first official release in OpenBSD 3.5 ports to FreeBSD and NetBSD appeared. Now 5 years later OpenBGPD grew from a niche BGP routing daemon to a real alternative if not even first choice for many usage cases. OpenBGPD is different in many regards when compared with quagga or Ciscos. It offers some unique features that simplify many setups and comes with sane defaults.

Intro - how it started

On October 22nd 2003 I received an email from Henning Brauer with the simple text "Hey, aren't you doing BPG as well? If so this may be of interest." At that time I was working for an ISP using zebra as routing daemon. Our core routers had massive issues to keep all sessions alive because zebra's bgpd could block for an extended period and neighbors considered the connection dead and closed the connection. Attached to his short mail was a description of the design of a new bgpd.
The idea of a better bgpd was a topic at work for some time so there was some excitement about this. I started looking at the code, his description, our own ideas and the RFC and after a few mail exchanges I was head over heels in it. At the beginning there were roughly 3000 lines of code implementing a basic session engine and a framework to do IPC. So Henning and I hammered additional code into this embryonic bgpd to make it do more then just sitting there and waiting. First in a private CVS repository and on December 17th 2003 bgpd got imported into OpenBSD. After roughly 4 month of development the initial release happened together with OpenBSD 3.5 which was limited but working. During the next two releases the code matured and bgpd became the first decision BGP routing daemon on OpenBSD -- and thanks to official OpenBGPD releases also the other BSD projects make more or less current ports available that many user enjoy. Slowly one feature after the other was added and there are still additional ideas and projects floating around waiting to be implemented.

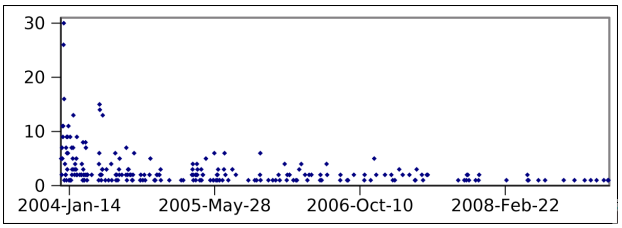
At the moment bgpd consists of 32 files with almost 2500 commits done over this 5 years. This does not include bgpctl or bgplg. Here some graphs illustrating the amount of commits done.



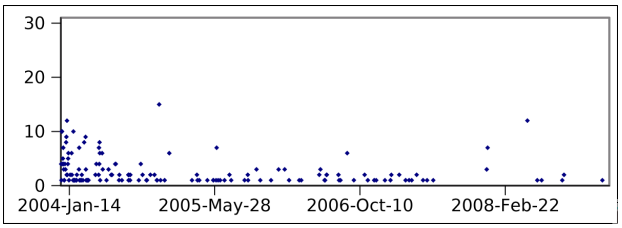
Number of commits per day done in src/usr.sbin/bgpd
from 2003-12-17 to 2009-02-12

After the initial rush of commits a continuous development happened until about 2007 when both Henning and I started to get too busy with way to many other projects and with the exception of a few spikes

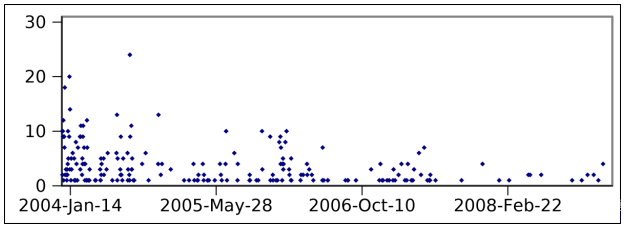
only minor commits happened. This is a sign of a mature product. The following graphs show the commits separated for the three main components of bgpd:



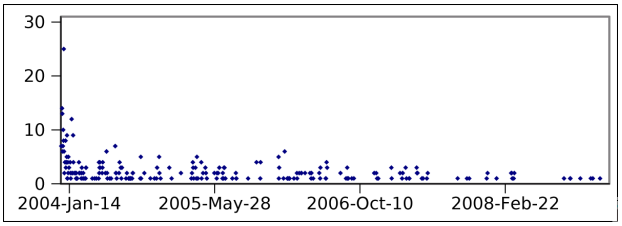
Commit graph for files belonging to the parent process



Commit graph for files belonging to the session engine



Commit graph for files belonging to the route decision engine



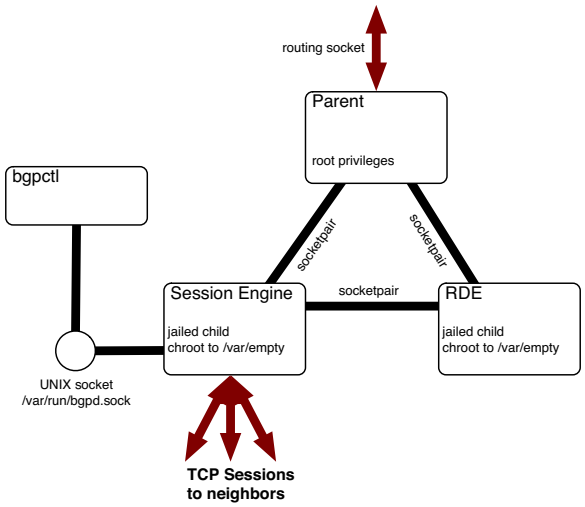
Commit graph for files which are used by all three processes

Every now and then a new milestone was reached. This is a selection of major events, the CVS log includes the full history.

- Dec. 17. 2003: bgpd was imported
- Jan. 2. 2004: bgpctl was added
- Jan. 13. 2004: max-prefix and announce self support
- Jan. 30. 2004: initial tcp md5sig support
- Feb. 19. 2004: initial filtering support
- Mar 11. 2004: initial communities support
- Apr. 24. 2004: first bits of IPv6 support added
- Apr. 25. 2004: neighbor cloning
- Apr. 27. 2004: IPsec support
- May 7 2004: support to add prefixes to a pf table
- May 21. 2004: RFC 2796 BGP route reflector support
- Oct. 19. 2004: allow neighbors to depend on the link state of an interface
- Nov. 1. 2004: first independent release of OpenBGPD
- Jul. 29. 2005: IPv6 multiprotocol support starts to work
- Jan. 24. 2006: soft-reconfigure in and out works
- Jan. 24. 2006: restricted bgpctl socket added
- Feb. 2. 2006: special neighbor-as community added
- Jun. 17 2006: carp demotion counter support
- Dec. 11. 2006: bgplg was added
- Mar 3. 2007: irrfilter generation added to bgpctl
- Apr. 23. 2007: 4-byte AS number support
- Feb. 11. 2008: another IPv6 specific bug fixed :)

Design

Why is OpenBGPD better then the alternatives?
The answer is simple because of its design.
Splitting the session handling from the routing table calculation ensured that as long as the userland was responsive no session would flap. Even if excessive time is spent in the routing table calculation, the preemptive scheduler of the system would ensure that the session engine is sending out the needed keepalives. At the same time being able to split the tasks into 3 processes allowed proper privilege separation. Only the parent process needs root privileges to alter the kernel routing table. The session engine and the route decision engine don't need special privileges and run out of a chroot() jail. In 2003 almost nothing was dropping privileges. Especially the routing daemons failed at it.



Three processes with specific tasks which communicate via socketpairs plus a control utility to gather information from bgpd

The interprocess communication between the three processes is based on a simple messaging system that is now used in many other daemons in OpenBSD. This imsg framework is using a simple buffer API which is also used to handle the network sessions. The buffer code is using poll() to do non-blocking read and writes to the various sockets open in the processes. First simple pipes were used to communicate between the processes but later on this was changed to socketpairs to allow file descriptor passing between processes. File descriptor passing became necessary to support configuration reloads and is now among others used for mrt dumps.

Instead of a CLI OpenBGPD uses a configuration file. The Cisco like CLI fails in many ways. Probably any network administrator has a story where a major incident happened because he could not enter the needed commands fast enough and some intermediate config became active and resulted in havoc. It is also hard to reconfigure stuff because the negated version of the active command need to be entered first which is almost impossible to do so in an automated way.

OpenBGPD uses a configuration file that can be reloaded atomically. The reload succeeds or fails but no matter what no intermediate config is run at any time. The possibility to include files into the master config file makes it easier to generate parts of a configuration via external tools.

There is no virtual tty to log into the daemon offering some sort of CLI to view the internal state. Instead OpenBGPD comes with a control tool to get the status of various parts of bgpd. It is even possible to clear a neighbor or issue a config reload through that tool. The control tool uses a local UNIX socket to communicate with the session engine and via the session engine messages are forwarded to the other processes. Again all information is packed into imsgs and sent back and forth. Because bgpctl is a normal UNIX command which is run from the user shell it is possible to pipe and alter the output with other well known commands. This is very powerful compared to the limited options of the other CLIs. Most Cisco CLI users are jealous about the comfort of a real shell OpenBGPD users enjoy. In OpenBSD 4.1 a less privileged control socket was added so that it was possible to implement a looking glass application without fearing somebody would take over the router.

Internals - How does it work

The BGP protocol is covered in RFC 1771 which got updated by RFC 4271. BGP itself is a path distance vector routing protocol where each destination has a path attribute that is part of the prefix. The prefix with the shortest path is considered best. On every hop an element -- the AS number -- is added to the path. BGP extents this simple algorithm with additional attributes and filters that may modify the attributes.

Session Engine

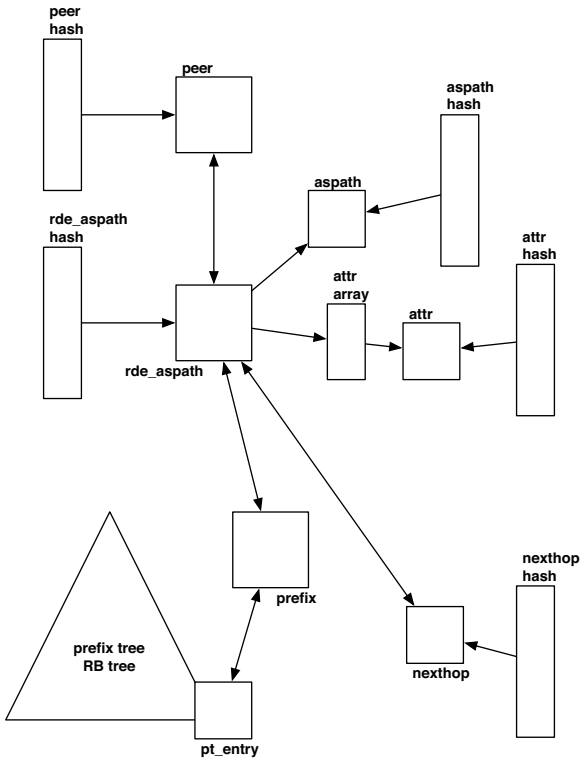
In bgpd the session engine initiates a session to a configured neighbor either by opening a TCP connection to the peer or by accepting an incoming TCP connection. After an initial welcome message exchange that ensures that everything is setup consistently the session is considered established. To keep the session up either updates or keepalive messages need to be sent out after a certain time. Keepalives are directly generated by the session engine to ensure that even high load on the RDE does not result in a session drop. The session engine does basic integrity checking of BGP messages but beforehand it needs to chop up the byte stream received via TCP. Messages that don't modify the routing table -- `OPEN`, `NOTIFICATION` and `KEEPALIVE` -- are directly handled by the session engine. Update messages are sent to the RDE and the parent process never gets any message with data that came from the wire.

Route Decision Engine

Apart from the BGP updates only neighbor state updates are sent from the session engine to the RDE. The other direction is about as easy because only updates and update error messages are sent back. The control messages issued by bgpctl use a second pipe so that large backlogs are not holding these messages up for too long.

The RDE needs to store all the prefixes plus the corresponding path attributes in the router information base (RIB). The RFC specifies three RIBs, Adj-RIB-

In, Adj-RIB-Out and Loc-RIB. The first two are the incoming and outgoing RIB of a peer (adjacency) and in most cases these tables are not needed. Only when `softreconfig in` is configured it is necessary to store the update before and after filtering. The Adj-RIB-Out can be calculated at any time from the Loc-RIB. The Loc-RIB is the local RIB where the decision process is run on and the actual routing table is calculated from. By normalizing the dataset plus some additional deduplication it is possible to store all data in a memory efficient way.



Datastructures used in the RDE

The RDE splits updates into multiple objects trying to reduce the memory foot print and making routing table calculations easy. All routes are added to a address family dependent red black tree -- the current IPv4 table consists of roughly 280'000 `pt_entries`. The main BGP path attribute object is struct `rde_aspath`. Complex data like the aspath or optional attributes that do not influence the decision process are stored externally in own hash tables with a reference count per object to track the number of users. Having some data in additional tables allows the RDE to allocate a particular attribute only once instead of per path and results in a modest additional memory requirement for `softreconfig in`. Struct `prefix` is the glue between a prefix and the corresponding path attributes. A struct `prefix` is marked as belonging to the Loc-RIB, Adj-RIB-In or both. So the various RIBs are merged into one big database.

RDE Memory Statistic

```
RDE memory statistics
   283114 IPv4 network entries using 8.6M of memory
  1074015 prefix entries using 32.8M of memory
   188752 BGP path attribute entries using 14.4M of memory
   169095 BGP AS-PATH attribute entries using 6.5M of memory,
          and holding 188752 references
     9774 BGP attributes entries using 229K of memory
          and holding 178697 references
     9773 BGP attributes using 180K of memory
RIB using 62.7M of memory
```

This memory statistic is from a router that has 4 full views and runs on a Via C3 with 700MHz and 512MB of RAM. As one can see for each IPv4 network entry and peer a struct `prefix` is allocated which consumes most of the allocated memory.

Lets go back to the processing of messages. When a neighbor state changes it may be necessary to send the full table over to the other side. This is currently a bit of a week point of bgpd. The table dumps are done in one shot because the RDE is allowed to block for a long time. While may be true for the BGP sessions it blocks almost all bgpctl commands as well and so affects the responsiveness of bgpctl. This is one of the hot topics that I try to solve in the near feature. Luckily neighbors don't flap that often...
When an update is received it is bisected into withdraws and changes. Withdraws don't carry any attributes with them. The attributes are not necessary to identify the prefix to be remove so there is also no need to filter on withdraws. In the worst case we try to remove something that is not in the tree. Prefix changes and additions -- which are just changes with no previous information stored in the tree -- have path attributes attached to them and after parsing the message the entry is filtered and maybe modified. Actually it is the other way around, path attributes arrive with multiple prefixes which share the same attribute set but that's nitpicking. If the update is not denied by a policy it is added to the tree and the route decision process is run.

Route Decision Process

The route decision process is defined by the RFC and selects the best path from a selection of available paths. In bgpd the route decision process orders the prefix list of a `pt_entry`. The first entry of this list is the best path to a destination. The route decision process evaluates the following steps to compare two prefixes. The next step is only evaluated if the current one returned a draw:

1. Only prefixes which belong to the Loc-RIB are considered. Other prefixes have the lowest preference.

2. Paths that are marked as not loop free are ineligible and have therefor a lower preference then eligible prefixes.
3. A prefix is only eligible if the nexthop of that prefix is reachable.

Now the actual decision process starts

4. The local-preference is compared first. The prefix with the biggest local-pref is the best one.
5. For prefixes with equal local-pref the AS-PATH length is compared. The shorter the path to the destination the better.
6. Prefixes with lower ORIGIN are preferred. The ORIGIN defines from which source a prefix was crated. The RFC defines 3 types IGP, EGP and incomplete with the first one having the lowest origin number.
7. The MED is compared and the prefix with the lowest number is used. The multi-exit-discriminator is special because only prefixes with the same leftmost AS should be considered unless `rde med compare always` is set.
8. Prefixes received by an external BGP session (EBGP) are preferred over those coming from an interior peer (IBGP).
9. Now the weight is compared. The weight is a bgpd extension that can be set locally and is considered as last tuneable allowing to prefer otherwise equal prefixes. The lowest weight wins.

From here on the prefixes could be considered equal but the decision process must return one best prefix. So additional information is used to make this decision. The last two steps were arbitrarily chosen and specified by the RFC.

10. If `rde route-age evaluate` is set older prefixes which tend to be more stable are considered. By default this step is skipped because it makes the decision process undetermined.
11. If there is still no best route the `router-id` of the neighbor is used. The lowest router-id wins.
12. If there are multiple sessions to the same neighbor the IP addresses are compared and again the lowest one is chosen.

When the best prefix changed during the process a route update is sent to the parent process and an update is generated for all peers but only after the output filters were run.

Parent Process

The parent process receives route updates and nexthop verification requests from the RDE. Additionally the process listens on the routing socket for anything interesting like link state changes. The parent process maintains the forward information base (FIB) which is what the kernel uses for routing and informs the RDE about changes affecting nexthops or announced networks. The other task done by the parent process is reloading the configuration and passing the new config down to the child processes.
Managing the routing table is harder then it sounds. The code to talk to the routing socket is complex because other daemons or the user himself may alter the routing table at any time and bgpd needs to pick up these changes and correctly merge them with the own view and IPv6 tries to make it even harder by needing special hacks and using badly aligned structures.

Configuration

The configuration is based on the yacc parser used by pf. While the syntax is different the behaviour and features are the same. The config file consists of four sections: macros, global configuration, neighbor settings and filters. The macros are not a real section but they allow to define variables at the beginning that can be reused later in the config.

There are quite a few settings that affect the operation of the daemon globally. First of all the `AS` number needs to be specified. It also makes sense to specify a `router-id` instead of letting bgpd pick (the wrong) one and to limit the addresses to `listen on`. It is also normal to announce one or more `networks`.

There are additional knobs which influence the RDE.

- nexthop verification can be set less strict
- the decision process can be tuned
- it is possible to run as a transparent router that does not prepend the own AS to the AS path.

simple global config

```
AS 65001
router-id 192.0.2.2
listen on 192.0.2.2

network 192.168.42.0/24
```

Multiple neighbors can be grouped together by a `group` section. Each `neighbor` in the group inherits the properties from its group. The most important element is the `remote-as`. Normally it is also good to set a description. This is often enough to get a session running but there are many other options to change the default behaviour. bgpd tries to come with sane defaults so that only a minimal set of options are needed to be set per peer but not everything can be

done automatically. It is also possible to define a neighbor template. If a connection is received from the specified network range a new neighbor is cloned from the template. A template does not even require a `remote-as` instead the one passed in the `OPEN` message is used.

a possible neighbor config

```
group "AS65002" {
     remote-as 65002

     max-prefix 5000 restart 15
     neighbor 192.0.2.1 {
          descr "AS65002 primary"
     }
     neighbor 192.0.2.3 {
          descr "AS65002 backup"
     }
}
```

The last section is the filter specification. There are three types of rules: allow, deny and match. A match rule will only modify the path attributes without influencing the filter decision. A rule can match on neighbors and groups, AS numbers, communities, prefixes and prefixlen ranges. Every rule can change the path attributes in various ways.

a minimal filter section

```
# filter out prefixes longer than 24 or shorter than 8 bits
deny from any inet prefixlen 8 >< 24
# do not accept a default route, multicast and experimental
networks
deny from any prefix 0.0.0.0/0
deny from any prefix { 224.0.0.0/4, 240.0.0.0/4 } prefixlen
>= 4
# mark the prefix so that we know where we learned it from
match from any set community 65001:neighbor-as
```

This is just the tip of the configuration iceberg. The bgpd.conf manual page has the complete reference. With `bgpd -nv` it is possible to see what bgpd is doing with the configuration and may be a big help in debugging a configuration.

Conclusions

After 5 years of OpenBGPD it is clear this is no toy implementation -- I would say it newer was but I'm biased. OpenBGPD is used in many mission critical systems in the core of the Internet. Not only in small network setups but also in places like Internet exchange points where many ISP peer with each other across an OpenBGPD route-server.

Many concepts implemented in bgpd are used in various other daemons like the imsg framework or the configuration file parser. The parser is actually based on an other successful product -- pf -- but only with bgpd the parser got so much attention that it got reused in other tools. Not only the other routing daemons like ospfd, ripd or ospf6d are based on bgpd also relayd, hostapd, snmpd, ypldap and cwm use code initially developed for bgpd.

A major mistake in the 4-byte AS number RFC specification was identified because of OpenBGPD and this way before the Internet as a whole was affected. It shows how important alternative implementations of core protocols are.