ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/261601381

Introduction to the Quagga Routing Suite

Article *in* IEEE Network · March 2014

DOI: 10.1109/MNET.2014.6786612

CITATIONS
43

READS
2,940

2 authors, including:

Paul Jakma

3 PUBLICATIONS **51** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Quagga View project

All content following this page was uploaded by Paul Jakma on 07 July 2014.

The user has requested enhancement of the downloaded file.

Introduction to the Quagga Routing Suite

Paul Jakma, David Lamparter

***Abstract*—Off the shelf routing products often are constrained by design to meet specific needs, both in terms of software and hardware. Networking professionals may face problems that require customisation of software, additional processing facilities or data storage, that are not provided for by those products. The Quagga Routing Suite provides implementations of several common routing protocols, distributed over multiple processes communicating via IPC, and support for their development, with source code provided under a modification-friendly licence. Quagga can help networking professionals build such custom solutions, in combination with other Open Source software packages. Quagga also provides a path for network researchers to increase the visibility of their work, and make it available to a wider community for potential testing and use, increasing the impact of that research.**

I. Introduction

Off the shelf routing products often are constrained by design to meet specific needs, be it in terms of software or hardware. The software may offer no means to customise the routing functionality, there may be little means for data gathering or processing, and so on. The hardware may not have any storage capabilities, or sufficient spare capacity for additional data-processing.

Networking professionals and researchers sometimes face problems that go beyond the capabilities of such routing products. They may wish to combine routing functionality with existing open-source software packages, but be constrained on how many separate devices can be deployed. Researchers and operators might wish to log routing protocol announcements, process them, and make them available, and hence require storage and scripting capacities that are not available in existing routing products. Researchers may wish to extend routing protocols, but not want to have to write complete implementations from scratch.

The Quagga Routing Suite[1] is a package of Unix/Linux software implementing a number of common network routing protocols, including the "Routing Information Protocol" (RIP)[2, 3], Open Shortest Path First (OSPF)[4, 5], the Border Gateway Protocol (BGP)[6] and IS-IS[7]. The package also includes a routing-information management process, to act as intermediary between the various routing protocols and the active routes installed with the kernel. A library provides support for configuration and an interactive command-line interface. Quagga is available under the terms of the GPLv2 licence.

Quagga thus provides a useful addition to the tool-box of networking professionals. Its existing routing protocols can be extended to enable experimentation, logging or custom processing. The libraries and kernel daemon provide a framework for easier development of new routing protocol daemon. It can be combined with arbitrary other available software packages, to allow a wide-range of functionality to be integrated into a single device, or allow for imaginative solutions to networking problems.

The community of operators, vendors, non-profits and researchers around Quagga provide a forum for increasing the visibility of work, and a potential path to wider testing and deployment of proposed modifications to routing protocols, or new routing protocols.

Section II will examine the history of the Quagga project. Section III gives some overviews of different aspects of the architecture of Quagga, in terms of how its various processes communicate; the features provided by its support library; the internal architecture of its BGP process and extension mechanisms available. Section IV considers the tensions that exist in the scalability, performance and reliability of the code in Quagga, and gives some concrete examples of evaluations and issues. Section V gives an overview of possible uses and applications of Quagga. Section VI considers the impact it can have if researchers present and attempt to contribute their work to the wider Quagga community, and how to do so.

II. History

The code in Quagga started out with a project called "GNU Zebra", created in 1996 by Kunihiro Ishiguro. GNU Zebra was released under the Free Software Foundation's "GNU General Public Licence", (GPL), version 2 (GPLv2). Under this licence it and its source code were free to use, distribute, examine and modify by all, so long as they passed on the same rights to others if they distributed the code.

By 2000 GNU Zebra had relatively feature complete OSPFv2, RIP and BGP implementations, thanks to the efforts of Kunihiro Ishiguro and a few others. Kunihiro Ishiguro continued development of this code-base with a corporate body, IPInfusion, as a proprietary project under the moniker "ZebOS". Development of the GPL "GNU Zebra" slowed somewhat in 2001 and 2002.

As a result, in 2002, the Quagga project was created, as a fork of the GNU Zebra code-base. The GNU Zebra community largely and quickly moved over to Quagga. The Quagga project has acted as a focal point for a growing number of contributors, including from academia, for-profit corporations, and industry funded non-profits.

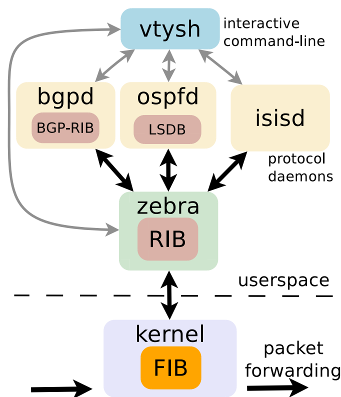

Figure III.1. Quagga Architecture

IPv4	*IPv6*	
zebra		FIB/kernel arbitration
babeld		BABEL wireless mesh protocol
bgpd		BGPv4+
isisd		IS-IS
ospfd	ospf6d	OSPFv2 and OSPFv3
ripd	ripngd	RIPv1/v2, and RIPng

Table III.1
Quagga daemons

Quagga has seen support added for various routing protocol extensions to, e.g., BGP and OSPF, as well as having support added for additional protocols, such as IS-IS. A number of companies have integrated Quagga into more cohesive commercial routing products.

III. Architecture

A. Overview

Quagga consists of a set of processes communicating via IPC, as shown in Figure III.1. Network routing protocols such as BGP, OSPF and IS-IS are run in processes such as *bgpd*, *ospfd*, *ospf6d*, *isisd* and so on. One daemon process, called "*zebra*" (a name inherited from GNU Zebra), acts as an intermediary between the kernel's forwarding plane and these routing-protocol processes. An additional interactive, command-line tool called "*vtysh*" allows these processes to be monitored and their configuration modified. The bundled daemons and the protocols they implement are shown in Figure III.1.

The *zebra* process maintains a shadow copy of packet forwarding related state, such as the network interfaces and the table of currently active routes. The latter is often referred to as the "Forwarding Information Base" (FIB). Typically the kernel is managing packet forwarding and so the kernel maintains these. However it would be possible to customise *zebra* to interact with other forwarding engines, such as dedicated hardware, if needed. A "Forwarding Plane Manager" protocol is provided to help facilitate this.

The *zebra* process also gathers routing information from the routing-protocol processes and stores these, together with its shadow copy of the FIB, in its own "Routing Information Base". The *zebra* process may also have static routes configured into its RIB. The *zebra* process then is responsible for selecting the best route from all those available for a destination and updating the FIB to use the best route. Additionally, information about the current best routes may be distributed to the protocol daemons. The *zebra* process also keeps the routing daemons informed of changes in network interface state, if applicable.

The BGP routing protocol is somewhat unusual, in that it is possible for active BGP speakers to not be involved in forwarding packets. For example, BGP route-servers may act as hubs for other routers, offloading some route-selection work to them; or BGP speakers may be involved in network management, reacting to events by injecting special routes, e.g. to counter-act Denial of Service attacks. For this reason the *bgpd* daemon may be run on its own, without any *zebra* daemon.

Routing processes communicate with the *zebra* process through a protocol called "ZServ". "Zserv" is versioned and intended to be stable. The *vtysh* command line tool communicates with other processes via a simple string passing protocol, where the strings are essentially identical to the commands entered.

B. Support library

A C support library provides several facilities, both general and networking related. Due to its heritage, the library is named "*libzebra*". Table III.2 provides an overview of the modules in the library. The API for this library is generally kept stable, and incompatible changes to it are strongly discouraged.

The library provides a bounded stream data structure, in *lib/stream.{c,h}*. This allows network messages to be marshalled and demarshalled in a safe way, with logical mistakes caught, so ensuring the process is safely terminated rather than corrupted. Network protocol clients are strongly encouraged to make use of this abstraction for all external IO, as it can significantly reduce the impact of security critical bugs.

The style of multi-processing in the existing Quagga code is to use distinct processes with IPC between them, and non-preemptive, co-operative threads within each process. The configuration, UI and IPC modules in the library rely on the co-operative thread sub-system to function. The "*if*" network interface state module can be kept automatically updated with the underlying system, if used with the *"zclient"* module.

The co-operative threads facility provides support for a number of types of tasks. Timer tasks may be specified with either second or millisecond precision, and are responsible for re-scheduling themselves as and when required. Read and write tasks may be setup to run when a file-descriptor becomes ready. Events are tasks which are

Basic Data Structures	linklist	
	hash	Hash table
	pqueue	Priority queue
	vector	Vector / growable array
Checksums / Hashes	checksum	IP header and Fletcher (TCP, IS-IS) checksums.
	jhash	Jenkins' hash
	md5	MD5 cryptographic checksum (obsolete)
Networking Data Structures	if	Network interface state
	prefix	Network address prefix
	stream	Bounded & checked, network byte-order aware, packet buffer
	table	Longest-matching prefix lookup table support
Configuration & UI	command	Configuration & monitoring command definition support
	vty	
I/O	buffer	Low-level buffer for output.
	network	Basic read/write convenience functions
	sockopt	Convenience functions for socket operations
Filtering	filter	IP address/prefix access-list filters
	plist	IP prefix list filters
	routemap	Match/action filtering
Co-operative threads	thread	Event, I/O and timer co-operative threads
	sigvent	Signal handlers
	workqueue	Work queues
Process support	daemon	Process daemonisation
	log	Logging, to UI and/or file.
	memory	Memory allocation, debug & tracking
	pid_output	Locked PID file writing
	privs	Least-privilege support, on certain systems
IPC	agentx	SNMP AgentX interface
	smux	SNMP SMUX interface (obsolete)
	zclient	ZServ client interface

Table III.2
Quagga library functionality

run as soon as possible. Background tasks are similar, but scheduled with a lower priority. The co-operative threads module keeps detailed usage statistics on the tasks run, which can be queried using the interactive interfaces. Abstractions for time-keeping are also provided by this module.

Listing 1 shows the gist of a simple signal handler, timer and read IO thread. The timer thread runs after 5 seconds and is passed a handle to external state, and must reschedule itself as required. The provided signal handler will actually run in normal process context, as only a small library stub runs in actual signal context. This "signal handler" therefore not restricted in what it can do, as normal Unix signal context code is. The read thread will be called when the given socket becomes ready to read.

Listing 1 Example co-operative threads

```
static int timer_count;
int timer (struct thread *thread) {
  int *count = THREAD_ARG(thread);
  printf ("run %d of timer\n", (*count)++);
  thread_add_timer (master, timer, count, 5);
}

void sighup (void) {
  printf ("processed hup\n");
}
struct quagga_signal_t sigs[] = {
  { .signal = SIGHUP, .handler = &sighup, },
};

int read_thread (struct thread *t) {
  int sock = THREAD_FD(t);
  int msg = read_msg (sock);
  thread_add_read (master, read_thread,
                   NULL, sock);
  return msg;
}

void setup (void) {
  signal_init (master, array_size(sigs),
               sigs);
  thread_add_timer (master, timer,
                    &timer_count, 5);
  thread_add_read (master, read_thread,
                   NULL, sock);
}
```

C. Other IPC Mechanisms

The *zebra* daemon contains functionality to allow *zebra* to manage FIBs in forwarding planes other than the kernel. The protocol is described in "*fpm/fpm.h*" in the sources. This allows *zebra* to connect to agents managing other forwarding planes via a socket.

The *ospfd* daemon, implementing OSPFv2, offers an OSPF-API IPC interface to allow its database of OSPF "Link State Announcements", the LSDB, to be queried. This interface also allows opaque LSAs to be injected into the OSPF network, via *ospfd*.

D. Internals overview

The *bgpd* BGP daemon internally hangs all its data-structure off a global, *bgp_master* singleton object. This contains some global configuration data and holds references to the event handling sub-system and work-queues; the main listening socket; and then references a list of all the BGP instances. Each BGP instance in turn holds configuration data, and references a special "self peer"; a list of configured peers; a list of peer groups, each of which references a special peer to hold configuration data for the group and a list of peers in the group; as well as a set of Routing Information Base (RIB) tables, for each Address Family (AFI), e.g. IP, IPv6, etc., and Subsequent-Address Family (SAFI), e.g. unicast, multicast, etc.; as shown in Figure III.2.

As shown in Figure III.3, each BGP RIB table is headed by a *bgp_table* object, which contains some basic descriptive

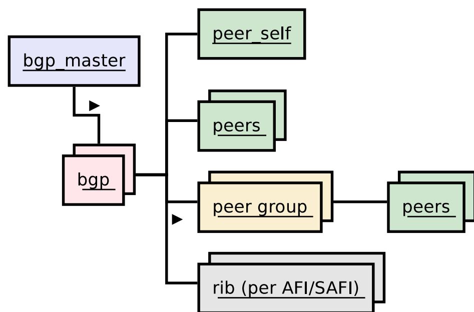

Figure III.2. High-level overview of *bgpd*

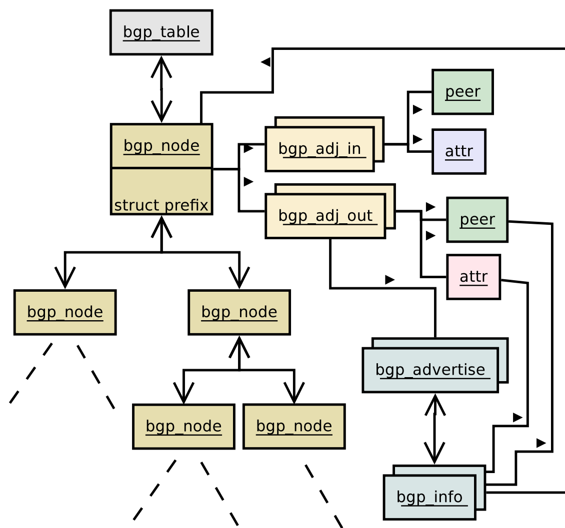

Figure III.3. BGP routing table overview

information. The routing table itself is composed of a binary tree of *bgp_node* objects, each of which contains the prefix it is describing, and maintains references to lists of advertisements received and advertised for this prefix, through the *bgp_adj_in* and *bgp_adj_out* objects. These conceptually correspond to the Adj-RIB-In and Adj-RIB-Out of [6], and together they allow the 3 notionally distinct RIBs of BGP to be indexed through a single tree, saving memory. The *bgp_adj_out* references the attribute set represented, and a list of *bgp_info* objects which are advertising that attribute set, and contain further, more specific information. The *bgp_adj_in* references only *peer* and attribute information, which is all that is necessary to be able to clean up routes when they are removed or when a BGP peer is disconnected.

Relatively heavy use is made in *bgpd* of normalising data-structures, through interlinking and sharing of objects, particularly in the RIB. The lifetimes of these shared objects are managed with manual reference-counting and garbage-collection. This allows memory use to be minimised. However, this obviously comes with the cost of increased complexity and increased scope for bugs.

Further overview of the *zebra*, *ospfd*, and *ripd*, daemons may be found in [8].

E. Extensions

Quagga's design, with functionality distributed across separate processes, facilitates both the addition of new protocols, as well as the extension of individual protocols. E.g., the zclient version of *olsrd*[9] demonstrates the former quite succinctly, integrating an existing implementation of OLSR[10] in only 390 lines of C code. As does the recent addition of a Babel[11, 12] implementation into Quagga.

More interesting in an academic context is probably modification of existing routing protocol implementations with relative ease. An extreme example of this is the US Naval Research Laboratory's implementation of OSPF MANET[13, 14] extensions, almost turning Quagga's OSPF implementation inside out. Ongoing protocol developments also find their way into Quagga as the result of thesis projects, as in the case of OSPFv3 auto-configuration[15, 16] or as efforts by the larger community, as with BGP SIDR security extensions[17] which is seeing involvement from the US's NIST[18].

There have also been efforts to implement MPLS for Quagga, dating back as far as 1999[19]. Unfortunately, these have only recently come to fruition, and will need further effort to mature. A major factor in this has been the need for system support below Quagga for performing label switching. It is reasonable to assume here that, unlike with IP routing, a lack of demand for label switching with general purpose systems accounts for some of the troubles here. Recent developments, in particular MPLS support in Linux's OpenvSwitch and on BSD systems hopefully herald progress here.

The RouteFlow project[20] makes OpenFlow-capable hardware available as forwarding plane to Quagga as a standards-based interface. This combines the customisability of open source software with the abilities of specialised hardware, allowing the combination of the best of both worlds.

IV. Scaling, Performance & Reliability

There is a balance to be struck between runtime performance, memory usage, CPU usage and reliability through isolation. There are trade-offs to be made between the use of complex indices, separate processes and IPC, data-sharing, and internal threading, which all impact on runtime performance, reliability and scalability and may result in a tension between them.

Scaling of memory may be achieved though increased data-sharing. This is easiest within one process. E.g., BGP attributes that are identical between different routes can be shared, and routes that are queued to be sent to different BGP peers can be shared. However, this also introduces risks to reliability, as it is easy for code to make mistakes in managing the lifetime of shared data objects. It may also reduce runtime performance by adding overheads.

Reliability may be increased by using separate processes, communicating through IPC. Failures in one process need not cause other processes to fail, e.g., the OSPF daemon can crash without affecting BGP. However, this may come at the cost of needing to copy more data around, which has runtime costs and may affect memory scaling.

Scaling of run-time may be achieved through increased multi-processing. One option is the use of fully pre-emptive threads, however these can reduce reliability through increased opportunities for race conditions. It may also be achieved through the use of separate processes, at the cost of IPC.

Clearly, there is a tension between these different goals. The sweet-spot may change in time, as relative costs of CPU, memory and programmer time change. Quagga's current design, of using one daemon per routing protocol along with one daemon acting as RIB/FIB manager, originates from the 1990s. Memory was more expensive then, and computers had few CPU execution contexts. The ever-increasing transistor counts of Moore's law generally were spent on increasing the sequential speed of those few execution contexts. A small, fixed number of processes which were able to maximise data-sharing made sense in this context.

Today, things have changed. Increasing transistor budgets are being spent on adding ever more execution contexts, rather than increased sequential speed. Memory has also gotten cheaper - though access time has gotten more expensive. The single-process per routing-protocol model arguably is no longer appropriate, as it results in a fixed number of execution contexts, and so does not make use of large numbers of cores.

In particular, *bgpd* has significant scaling problems with large numbers of peers, which became apparent with, e.g, route-servers at IXPs. Efforts were made before to increase granularity of its internal co-operative threading, but this offered only a little respite. Other efforts have been made, sponsored by Euro-IX, to fully-thread b*gpd*, with some success. Though, those changes have been wide-ranging, and to date have not been merged back into mainstream Quagga. Threading *bgpd* so that peer communication and initial route-processing is handled by threads has been reported to have significant performance benefits[21]. A more dynamic allocation of work to threads in *bgpd* could give additional performance gains[22].

There is perhaps also room to experiment with dividing *bgpd* into separate processes that communicate via IPC. This might also require revamping the configuration and interactive control/monitoring modules to support the case where a single routing protocol has its functionality distributed over multiple processes. Such functionality would also benefit other protocols. E.g., *ospfd* is also starting to have its scalability tested heavily by a few users with very large numbers of LSAs in their OSPF, e.g., where they inject very large numbers (> 20k) of external routes into OSPF.

An open question is how badly a many-process approach would affect memory scaling, and how the trade-offs would compare with the fully-threaded approach.

However, sometimes there are relatively clear gains to be made. The initial implementation of Dijkstra's shortest-path algorithm in *ospfd* in GNU Zebra, and inherited by Quagga, was based on a simple, ordered linked-list to maintain the discovered vertices. Its performance was quite adequate with small networks, the low-overhead making it more than competitive with other implementations. However, beyond small networks, *ospfd* would perform poorly[8], scaling with $O\left(n^2\right)$ in runtime on graphs with n nodes and $O\left(n\right)$ edges[23]. Replacing the ordered, linked-list with a priority queue in Quagga vastly improved the scalability of the Dijkstra SPF algorithm, allowing *ospfd* to run in $O\left(n \log n\right)$ time, and with lower constant factors than other implementations[23].

V. Applications

GNU Zebra and, later, Quagga have found use in a number of areas, both commercial and academic. As Quagga is supported on a number of standard Unix/Linux systems, this allows it to be run on a breadth of hardware, from small embedded routers, to large hardware. Its GPL licence guarantees the freedom to modify the source code. Quagga may be useful in situations where standard router offerings do not suffice, e.g. because more CPU power or memory is required than is available with standard router hardware, or where a more flexible, powerful or customisable software environment is needed than offered by traditional router software environments.

One application is using BGP as a route-server, acting as a hub for the processing of BGP routes and so avoiding the need for the clients of the route-server to all connect with each other. Quagga has facilities to allow it to maintain a separate BGP-RIB for each client, and thus apply policy separately from the perspective of each client. With these per-client specific BGP RIBs, a route-server is able to consider every route from every client in the context of every other client. This does however have scaling limitations, which are further exacerbated with a single-process/thread *bgpd*. Further details on this feature are in the Quagga documentation.

Another common use is the data-logging of routing protocols. The "MRTv2" format is supported for dumping BGP state and activity, for later analysis. Projects such as the "University of Oregon Route Views Archive Project"[24] and the "RIPE Routing Information Service"[25] use GNU Zebra and Quagga to monitor and archive Internet BGP updates, from a number of observation points around the Internet. These are then available for anyone to download and analyse. The results of many research papers came from data logged this way.

Quagga can also be used with virtualisation (KVM/QEMU, OpenVZ, VMWare, etc), to cheaply

simulate network scenarios, for educational or research purposes. A number of research papers have used Quagga as part of simulations of existing routing systems, or to help evaluate proposed modifications to those systems in this way. As an example, [26] used Quagga to evaluate a system to integrate IGP and BGP information from across a network in a central controller to provide scalable routing. Further, the code may also be adapted and used with other software, according to the terms of the GPLv2 licence, e.g., integrating portions of *bgpd* code into *ns-2*[27].

Further, it is quite feasible to run many instances of the Quagga BGP daemon simply as regular processes and have them connect locally to each other, and so build virtual BGP networks but without the overhead of system-level virtualisation. The Quagga sources provide a script, in "*tools/multiple-bgpd.sh*", giving an example of how to do this. This can greatly simplify the testing or evaluation of BGP scenarios. One or more of these *bgpd* processes could also connect with external BGP speakers, and so this can be used also to help evaluate other BGP implementations.

VI. Community

There is a community around Quagga, where users seek advice from each other and contributors propose and discuss modifications. The community is centred around the website at http://www.quagga.net and its mailing lists. Discussions about usage, and other general matters, take place on the quagga-users[28] email list. The quagga-dev[29] list is used for discussions on the development of Quagga.

Modifications are generally made by consensus after a review process, curated by the maintainers of Quagga. The maintainers may be reached privately via maintainers@quagga.net, however only matters that genuinely require privacy will be answered there - support and development queries should be directed to the appropriate email lists mentioned above.

A. Contributing Back

Many people who modify Quagga intend for their modifications to stay private, which is perfectly acceptable under the GPL licence as long as the result is not redistributed. However, for anyone who intends to use and work on Quagga for extended periods of time and make ongoing modifications, it will generally be easier to stay synchronised with changes in the public Quagga if their own modifications are incorporated into it.

Others may be open to having their modifications integrated with the public Quagga, but may not have the time to do so. This appears often to happen with research projects, where time is only budgeted for research goals, and little or none for contributing the code to the community. However, contributing back to Quagga can have a research impact, particularly in the longer term. Merely presenting a modification back to the Quagga community will gain the research some additional visibility with network operators and other researchers. Doing what is required to have the changes integrated into Quagga will give it yet longer-term visibility, and get that code to a yet wider audience of network operators. That wider audience of operators potentially may test or even deploy that work. Such use may potentially lead to additional research opportunities in the future.

In short, contributing back to Quagga and engaging with the community may help increase the impact of those research projects that involved extending Quagga.

B. Contribution Process

Getting a modification integrated into Quagga requires presenting the modifications for consideration, and building a consensus on the fitness for inclusion. This may require engaging with others in the community, particularly for more complex modifications. The process for this is detailed on the "Development" section of the website, but is relatively informal. To start the process:

- Prepare the modifications as 1 or more change-sets, in "diff -uwb" format, preferably as git commits
- Email the change-sets to the quagga-dev list for review

Anyone in the Quagga community may offer reviews, and are encouraged to. The review and discussions may sometimes ask that the changes-sets be further modified. If the changes are complex, or subtle, further explanation may be asked for. The rationale for the change-sets may be challenged. The review process may sometimes feel adversarial, but try not to take this as a personal slight. Some changes asked for may even seem trivial (e.g. style issues), but the reviewer may ask for them to avoid an accumulation of trivial issues, out of a long-term concern for the code-base. Submitting a change for inclusion is the start of a process which may sometimes take time, engagement, and possibly further work to refine the changes before they can be accepted.

For modifications which are likely to be substantive, it is a good idea to try involve the community in discussion earlier in the development process rather than later. Doing all design and coding in private risks that the Quagga community will disagree with aspects of the direction taken. When the modification is finally presented to the community, this may lead to reviews requesting major changes, requiring time the contributor had not planned for. These risks can be reduced by instead conducting design discussions in public on the quagga-dev list, prior to the bulk of the coding work.

Some general tips for successfully contributing to Quagga:

- Try to separate changes into logically consistent, distinct change-sets, with each change-set as small as

possible. Avoid mixing together unrelated changes in one change-set.
- Give a rationale with the change-set, so the reviewer can understand:
  - The abstract motivation for the change: What problem is fixed? What feature is added?
  - The general method by which the change-set addresses the problem, or adds the feature.
  - The confidence that can be had in the change: What testing was done?
- Providing test-cases and/or unit tests can *greatly* help make a case for a change.
- The review and integration process sometimes takes time, and potentially require multiple rounds of refinement of the work and review. This is especially true for larger contributions. Account for this in project plans as far as possible.

Those wishing to modify Quagga are strongly encouraged to read the "HACKING"[30] document first. This is included in the Quagga sources as "*HACKING.tex*", and also available in PDF form on the main website. It contains guidance on the established development practices, styles and processes for Quagga. Contributors to Quagga are strongly encouraged to follow it where-ever possible. If a practice seems outdated, non-optimal or unnecessary, the best course of action is to edit and update the document, and then submit those proposed changes to the quagga-dev email list for discussion, rather than ignoring the advice.

VII. Conclusion

Quagga can be a useful addition to the toolkit of networking professionals, both in commercial and research settings. It provides a hackable platform for solving networking problems. There is a diverse community around it, ranging from network operators, to routing platform vendors, to academic researchers, amongst others. This community is open and consensus based. Everyone is welcome and encouraged to step forward and contribute, be it through development of the code, updates to the documentation, code reviews, helping with infrastructure, or suggesting improvements to processes.

Quagga is what you make of it. See you there!

References

[1] "Quagga software routing suite." [Online]. Available: http://www.quagga.net/

[2] G. Malkin, "RIP Version 2," RFC 2453 (INTERNET STANDARD), Internet Engineering Task Force, Nov. 1998, updated by RFC 4822. [Online]. Available: http://www.ietf.org/rfc/rfc2453.txt

[3] G. Malkin and R. Minnear, "RIPng for IPv6," RFC 2080 (Proposed Standard), Internet Engineering Task Force, Jan. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2080.txt

[4] J. Moy, "OSPF Version 2," RFC 2328 (INTERNET STANDARD), Internet Engineering Task Force, Apr. 1998, updated by RFCs 5709, 6549, 6845, 6860. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt

[5] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6," RFC 5340 (Proposed Standard), Internet Engineering Task Force, Jul. 2008, updated by RFCs 6845, 6860. [Online]. Available: http://www.ietf.org/rfc/rfc5340.txt

[6] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), Internet Engineering Task Force, Jan. 2006, updated by RFCs 6286, 6608, 6793. [Online]. Available: http://www.ietf.org/rfc/rfc4271.txt

[7] D. Oran, "OSI IS-IS Intra-domain Routing Protocol," RFC 1142 (Informational), Internet Engineering Task Force, Feb. 1990. [Online]. Available: http://www.ietf.org/rfc/rfc1142.txt

[8] Y.-D. Lin, R.-H. Hwang, and F. Baker, *Computer Networks: An Open Source Approach*. McGraw-Hill, 2012. [Online]. Available: http://highered.mcgraw-hill.com/sites/0073376248/

[9] C. Franke, "OLSRd, quagga zclient enabled version," 2011. [Online]. Available: http://git.nowhere.ws/?p=olsrd-zclient.git;a=summary

[10] T. Clausen and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)," RFC 3626 (Experimental), Internet Engineering Task Force, Oct. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3626.txt

[11] J. Chroboczek, "The Babel Routing Protocol," RFC 6126 (Experimental), Internet Engineering Task Force, Apr. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6126.txt

[12] "Babel âĂŤ a loop-avoiding distance-vector routing protocol." [Online]. Available: http://www.pps.univ-paris-diderot.fr/~jch/software/babel/

[13] "OSPF MANET designated routers (OSPF-MDR) implementation." [Online]. Available: http://www.nrl.navy.mil/itd/ncs/products/ospf-manet

[14] R. Ogier and P. Spagnolo, "Mobile Ad Hoc Network (MANET) Extension of OSPF Using Connected Dominating Set (CDS) Flooding," RFC 5614 (Experimental), Internet Engineering Task Force, Aug. 2009, updated by RFC 7038. [Online]. Available: http://www.ietf.org/rfc/rfc5614.txt

[15] "OSPFv3 auto-configuration for quagga." [Online]. Available: https://github.com/edderick/quagga_zOSPF

[16] A. Lindem and J. Arkko, "OSPFv3 auto-configuration." [Online]. Available: http://tools.ietf.org/html/draft-ietf-ospf-ospfv3-autoconfig

[17] M. Lepinski and S. Kent, "An Infrastructure to Support Secure Internet Routing," RFC 6480 (Informational), Internet Engineering Task Force, Feb. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6480.txt

[18] “BGP secure routing extension (BGP-SRx).” [Online]. Available: http://www-x.antd.nist.gov/bgpsrx/

[19] J. Leu, “MPLS for linux.” [Online]. Available: http://sourceforge.net/apps/mediawiki/mpls-linux/index.php?title=Main_Page#History

[20] M. R. Nascimento, C. E. Rothenberg, M. R. Salvador, C. N. A. CorrÃła, S. C. de Lucena, and M. F. MagalhÃčes, “Virtual routers as a service: The RouteFlow approach leveraging software-defined networks,” in *Proceedings of the 6th International Conference on Future Internet Technologies*, ser. CFI ’11. New York, NY, USA: ACM, 2011, p. 34âĂŞ37. [Online]. Available: http://doi.acm.org/10.1145/2002396.2002405

[21] K. Wang, L. Gao, and M. Lai, “A scalable multithreaded BGP architecture for next generation router,” in *Proceedings of the International Conference on Human-centric Computing 2011 and Embedded and Multimedia Computing 2011*, ser. Lecture Notes in Electrical Engineering, J. J. Park, H. Jin, X. Liao, and R. Zheng, Eds. Springer Netherlands, 2011, vol. 102, pp. 395–404. [Online]. Available: http://dx.doi.org/10.1007/978-94-007-2105-0_36

[22] S. Grover, A. Dhanotia, and G. Byrd, “A canonical multicore architecture for network routers,” in *Architectures for Networking and Communications Systems (ANCS), 2011 Seventh ACM/IEEE Symposium on*. IEEE Computer Society, 2011, pp. 134–144. [Online]. Available: http://dx.doi.org/10.1109/ANCS.2011.30

[23] V. Eramo, M. Listanti, N. Caione, I. Russo, and G. Gasparro, “Optimization in the shortest path first computation for the routing software GNU zebra,” *IEICE TRANSACTIONS on Communications*, vol. E88-B, no. 6, pp. 2644–2649, Jun. 2005. [Online]. Available: http://search.ieice.org/bin/summary.php?id=e88-b_6_2644

[24] D. Mayer, “University of oregon route views archive project.” [Online]. Available: http://routeviews.org/

[25] “RIPE routing information service (RIS).” [Online]. Available: https://www.ripe.net/data-tools/stats/ris

[26] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, “Design and implementation of a routing control platform,” in *Proceedings of the 2Nd Conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI’05. Berkeley, CA, USA: USENIX Association, 2005, p. 15âĂŞ28. [Online]. Available: http://dl.acm.org/citation.cfm?id=1251203.1251205

[27] X. A. Dimitropoulos and G. F. Riley, “Efficient large-scale BGP simulations,” *Computer Networks*, vol. 50, no. 12, pp. 2013 − 2027, 2006, <ce:title>Network Modelling and Simulation</ce:title>. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128605003269

[28] “Quagga users email list.” [Online]. Available: http://lists.quagga.net/mailman/listinfo/quagga-users

[29] “Quagga development email list.” [Online]. Available: http://lists.quagga.net/mailman/listinfo/quagga-dev

[30] “Quagga HACKING document.” [Online]. Available: http://www.nongnu.org/quagga/docs/HACKING.pdf

View publication stats