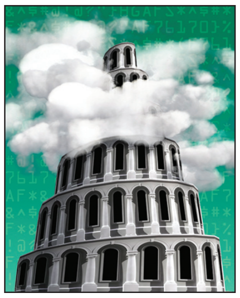

Routers for the Cloud

Can the Internet Achieve 5-Nines Availability?

Andrei Agapi, Ken Birman, Robert Broberg, Chase Cotton, Thilo Kielmann, Martin Millnert, Rick Payne, Robert Surton, and Robbert VanRenesse

Today's Internet often suffers transient outages, but as increasingly critical services migrate to the cloud, much higher levels of Internet availability will be necessary.

The stunning shift toward cloud computing has created new pressures on the Internet. Loads are soaring, and many applications increasingly depend on real-time data streaming. Unfortunately, the reliability of Internet data streaming leaves much to be desired. For example, at the University of Washington, the Hubble system (www.cs.washington.edu/research/networking/astronomy/hubble.html) monitors Internet health using all-to-all connectivity and throughput tests between hundred of end points through the Internet. The effort has revealed transient periods of very indirect routing, Internet "brownouts" (performance problems), and even "black holes." All these problems are surprisingly common, even when looking at routes entirely within the US or Europe.

Here, we focus on routing in the Internet's core, at extremely high data rates (all-to-all data rates of 40 Gbits per second are common today, with 100 Gbits/s within sight). These kinds of routers are typically implemented as clusters of computers and line cards: in effect a data center dedicated to network routing. The architecture is such that individual components can fail without bringing the whole operation to a halt. For example, network links are redundant; if one link fails, there will usually be a backup. Such a router could even run routing protocols of different types side-by-side, making the actual routing decisions by consensus — if some protocol instance malfunctions, its peers would simply outvote it.

But suppose that a routing protocol (for clarity, we focus on the Border Gateway Protocol [BGP], implemented by a BGP daemon [BGPD] hosted on some node within the router) needs to be restarted after a crash or updated with a software patch or migrated within the cluster. The resulting sequence of events can take several minutes, during which BGPD might be unavailable or not yet fully resynchronized. The resulting routing changes can ripple throughout the entire Internet, triggering routing events far from the one on which BGPD had to be restarted.

Could events of this kind account for the issues Hubble saw? On a typical core router, it can take two or three minutes to restart BGPD from scratch. Moreover, BGPD might need to be restarted as often as once per week. Thus, it's entirely possible that BGPD restarts are a significant factor.

In this article, we report on a new software architecture that can help mask BGPD outages, greatly reducing their disruptive impact. Moreover, the same techniques should be applicable to daemons associated with other important Internet routing protocols (we've already used the approach for two different BGP implementations, and an Intermediate System to Intermediate System [IS-IS] routing daemon).

A Close Look at BGP

Before drilling down on BGP availability, it might be helpful to be more precise about what availability means for a core Internet router. Routers drop packets during capacity overload (TCP flow control adapts based on overall path capacity), so it would make no sense to insist that a reliable router deliver every single packet. Accordingly, we adopt an approach first used in telephony, where availability measures the percentage of time when almost all calls go through (that is, only a small percentage are dropped, and in an uncorrelated way). The wired telephone infrastructure is engineered to guarantee

99.999 percent availability: the "5-nines" standard.

In a one-year reliability study of IP core routers in a regional IP service provider network conducted by the University of Michigan, router interface downtime averaged roughly 955 minutes per year, which doesn't even reach the "3-nines" level. Figure 1 shows the breakdown of problems that this study identified. The results support the view that redundant hardware has great potential: back in 2004, when the university conducted the study, most deployed routers were monolithic (nonclustered), and many links played unique, critical roles. Hardware and link failures jointly accounted for almost a third of outages. With redundant hardware and links, both factors have since been sharply reduced — putting ever greater emphasis on IP routing's reliability.

This need for software that can survive hardware outages is vital because we must minimize the percentage of time that the routes the router is using are inconsistent with those its neighbors use — for example, because the router has yet to apply routing updates that the neighbors are already employing. A more complete discussion of IP routing failures is available elsewhere.[1]

BGP is designed for use in networks composed of interconnected *autonomous systems* (ASs). An AS could be a network operated by some ISP, or might be a campus or corporate network. BGP maintains a table of IP networks, or "prefixes," that represent paths to a particular AS or set of ASs, tracking both direct neighbors and more remote ones. A BGPD instance runs on a router and uses path availability, network policies, or operator-defined databases of routing rules (patterns the operator has defined) to select preferred routes. It then advertises reachable prefixes by publishing sets of attributes that include the paths. As routing changes, BGPD exchanges updates with its peers that might add to the list of reachable prefixes or retract some prefixes; those peers are expected to update their own states accordingly. BGP allows BGPD instances to apply routing updates in an unsynchronized, distributed manner, but normally the delay between when one router applies an update and when its neighbor does is negligible, hence this asynchrony isn't noticed: most routers are working with very similar routing tables at any given moment. However, one important case exists where the lag can be larger: if BGPD is recovering when an update arrives.

Imagine that some router experiences an event that forces it to restart BGPD. When BGPD fails or migrates, the TCP links from it to the BGPDs on neighboring routers disconnect (break). Those neighbors will sense the failure and try to route around the affected router, but the alternative routes might be poor ones, and sometimes no backup routes are available (recall that we're focused on the Internet's core, where data rates are so high that only Internet "backbone" links and routers can handle the load). This, in turn, can trigger secondary routing decisions at routers further away, and so forth.

So, how can we make BGPD more available? Currently, the main approach is to activate a BGP feature called *graceful restart*, which exploits routing tables that were downloaded into the hardware line cards prior to the crash. Assuming the crash left the routing tables intact, when the new BGP service starts up, the router will still be running using the old routing table, a bit like an airplane on autopilot. The router won't be adapting to new routing updates and is thus frozen in time, but at least it was initially in a consistent state. Graceful restart tells the neighboring routers to continue to route packets through the impacted router, even as the restarting BGPD resynchronizes with its peers. The problem, however, is that while this is happening, BGP updates continue to stream in at a furious pace, so routing tables can become inconsistent within seconds.

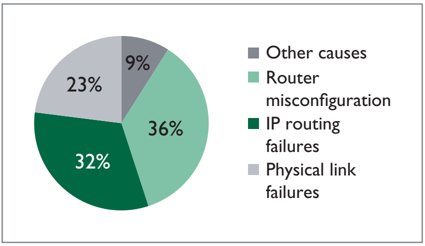

Figure 1. Causes of network downtime. In a 2004 study of router reliability, hardware outages, link failures, and Border Gateway Protocol (BGP) outages were dominant causes of downtime. The use of software fault-tolerance mechanisms to protect BGP and routers with redundant hardware and network links dramatically improves availability.

This creates a strong motivation to improve routing daemon availability. For example, some work has aimed at running BGP in a movable virtual machine (but VM migration is slow, and offers no help for fault tolerance), and some hand-tuned BGP migration mechanisms exist.[2] Our approach offers fault tolerance, can support BGP upgrades (patching), and works with routing daemons other than BGPD, yet is fast and built from surprisingly simple technologies.

Fault-Tolerant BGP

Our new approach uses software to transform a standard BGPD implementation into a fault-tolerant service. It involves minimal changes to the existing BGPD, the operating system, and existing protocols such as TCP, IP, and UDP. The first step is to "wrap" BGPD in a fault-tolerance layer, the fault-tolerance *shim*. The shim helps the underlying routing protocol handle failures in ways invisible to remote peers.

Figure 2 illustrates the approach. The solution combines the existing BGPD with several new components. The first is *fault-tolerant state storage* (FTSS), in which the shim stores BGP state and other data that must be

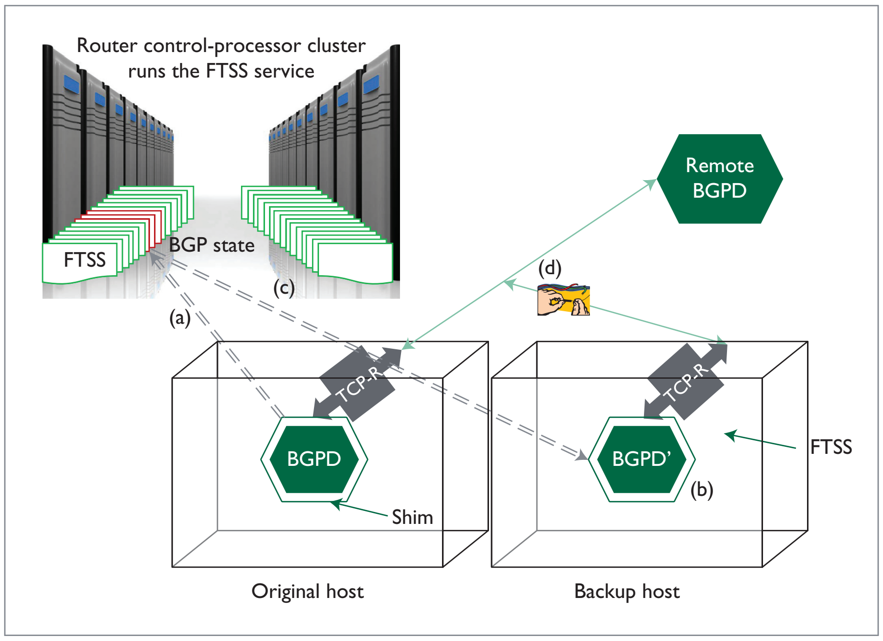

Figure 2. The fault-tolerant Border Gateway Protocol daemon (BGPD) architecture. The architecture runs on a router constructed as a cluster of host computers that control racks of network line cards, with the actual routing done mostly in the hardware. The shim (a) replicates BGPD's runtime state into the fault-tolerant storage service (FTSS), which is implemented as a distributed hash table (DHT; shown as an oval). The figure portrays the BGPD state as if it resided at a single node, but in fact the solution spreads it over multiple nodes for fault tolerance and also to speed things up by enabling parallel PUT and GET operations. After a failure, (b) BGPD can restart on some healthy node, (c) recover its precise state, and resume seamlessly. The TCPR layer (d) splices connections from the new BGPD to the still-open end points its remote peers hold, masking the BGPD failure/recovery event.

preserved across failures. The second component is the shim itself. The solution routes BGP connections between BGPD and its peers through the shim, so that the shim can see all incoming and outgoing updates as well as any changes to the routing table. This lets the shim checkpoint all this information so that any incoming update will be securely logged in FTSS before our BGPD actually sees it, and any outgoing or routing table update will be securely logged before being sent to a neighboring peer or installed into the hardware.

The shim can also support multiple routing protocols running side-by-side, a configuration that often arises in the core Internet, where an AS might have internal routing protocols that it uses to manage its own network, and a separate BGP routing layer that talks to neighboring ASs. It uses a form of voting to select among competing routing "proposals" in such cases, combining the routing protocol outputs to create the routing table that will be downloaded into hardware.

Of course, the shim itself can experience a failure, so we've designed it to store its state in the FTSS, enabling it to recover rapidly on a different node. The last component of our solution can "splice" the new TCP connections (which the shim creates) to the old TCP connections that it was previously using to connect to remote peers. Called TCPR (for "TCP with session recovery"), this splicing technology works somewhat like network address translation (NAT), but rather than translating source and destination addresses in NAT-style, TCPR also updates the TCP sequence numbers. The effect is to connect the new connection to an existing, active, TCP connection that is open at a peer, in a manner that won't lose any data and imposes just milliseconds of delay.

We've focused primarily on the shim; let's next look at our approach's other components in more detail.

FTSS

FTSS is a fault-tolerant storage solution that saves and replicates state so that in the event of a failure, a state-dependent component can recover its previous configuration. In our architecture, the shim is the only component that interacts directly with FTSS, using it to store the wrapped BGPD's state, incoming and outgoing BGP updates, the routing information table, and a small amount of additional state associated with TCPR. FTSS runs on all nodes within the router; in our target setting, this would range from a few dozen nodes to several hundred.

FTSS is implemented as a one-hop, in-memory, performance-optimized distributed hash table (DHT). Each state record has a unique ID (basically, a file name and a block number), and FTSS uses this as a key. The component maps the key to a few nodes within the router (recall that the router is a cluster), and FTSS agents on these nodes replicate the update. Lookup works the same way.

FTSS maintains full membership tables (with at most a few hundred nodes in each router, and often far fewer, the full address list easily fits in memory). Consequently, FTSS can perform requests with a single RPC to each target node. FTSS also leverages parallelism: we break the BGP state into a large number of small chunks and spread these over many machines, doing PUT and GET operations in parallel, and in this way gain roughly an order of magnitude in speed. Even when we take into account delays associated with the need to replicate data for robustness, this yields a fast, flexible store. In fact accessing remote memory in this manner is approximately two orders of magnitude cheaper than file I/O to a standard local disk, and many orders of magnitude faster than remote file I/O. To support checkpoints and complex object storage, FTSS extends the usual DHT key-value model to also support record linking and offers efficient ways to traverse linked data structures.

BGPD

As noted, we made only minor changes to the existing BGPDs with which we worked (we've applied our methodology to two, so far: Quagga BGPD and a proprietary Cisco BGPD). The main change was to have BGPD connect to the shim rather than directly to its remote peers. A side effect is that without further modification, when BGPD restarts, the shim can supply the initial routing state: rather than informing remote peers of the restart, the shim itself senses the restart, pulls the needed state from FTSS, and pushes it into BGPD at a very high data rate. In our experiments, using state typical of real core-Internet routing conditions, this took as little as 1.5 to 4 seconds. The remote peers, of course, remain completely unaware of the event. Finally, when the remote peer set changes, BGPD informs the shim so that it can manage the associated connections.

TCPR

TCPR is a TCP-splicing technology. The approach is best understood by first considering the behavior of a standard NAT box: it has the effect of grafting a TCP end point that thinks itself to be connected to server *X* on port *P* to a server that might really be running on machine *Y* using port *S*. The NAT box translates back and forth. TCPR works in much the same way but at the level of the byte-sequence numbering used within TCP's sliding window protocol.

The key idea is very NAT-like: when a restarting BGPD's shim wrapper tries to connect to a peer, TCPR intercepts the three-way handshake so that the remote peer won't see a connection reset. Instead, it computes the "delta" between the randomly chosen initial sequence number for the new connection and the sequence numbering used in the old connection. As packets are sent back and forth, TCPR adds or subtracts the delta, depending on which way the packets are going. Thus the new connection end point finds itself talking to the old remote end point. TCPR handles the TCP options used in routing protocols such as BGP, including the MD5 signatures. In our experiments, TCPR splicing takes as little as 350 microseconds, and having TCPR on the path has a negligible impact on TCP connection performance.

TCPR and the shim cooperate in several ways. First, TCPR delays outgoing acknowledgments until the shim confirms that it's backed up the associated incoming data; this ensures that, after a crash, the new BGPD won't see any gaps or duplicated bytes in the incoming data stream. Similarly, the shim backs up any outgoing data so that, after a node crash, the recovered shim/BGPD pair can finish transmitting any data that was being sent at the time of the crash. Finally, the shim backs up parts of the TCPR state, enabling TCPR itself to recover if a node running it crashes and the TCPR daemon must restart.

Solution Performance

As this article was going to press, we were just finishing our port of the full fault-tolerant BGP implementation to an actual CRS-1 router and hadn't yet measured recovery times or the corresponding router-availability levels in a true Internet deployment. However, we do have a full implementation running on a testbed, and were able to experiment with it using realistic BGP routing tables and update traffic. The results are encouraging: complete recovery finished in as little as 30 ms for a BGPD that had no routes to recover (for instance, one with an empty routing table) and 405 ms for a BGPD with a large routing table containing 157,975 entries. These numbers were essentially unchanged when we tested with BGP updates arriving every 130 ms, and fall within the window of normal asynchrony between BGP peers in the core Internet. Overall, the ability to fail and recover transparently, coupled with the ability to test new versions and configurations of routing software in production without risk, eliminates many of what used to be the biggest causes of downtime.

Today's cloud computing systems are appealing for their low cost of ownership, amazing scalability, and flexibility. The cloud even brings environmental benefits: users share computing resources, which are used more efficiently, and the data centers are typically located near power-generating sources: by using the network to move data to a data center, the need to move electricity to widely scattered computing devices is reduced. However, for many applications, network routing instabilities make the cloud less reliable than it needs to be.

Our work tackles a root cause for this problem, and by dramatically improving router availability, offers a path toward better stability in the Internet as a whole. The technique is incrementally deployable (meaning that it can be rolled out without change to routers that run existing protocols) and brings immediate benefit to any path that traverses even just a few routers using our approach. With enough routers using the method, we could imagine that VoIP telephony could achieve the same

An Internet for the Cloud

Cloud computing, particularly in conjunction with increased device mobility, is reshaping the Internet. We're seeing unprecedented shifts in demand patterns, a broad spectrum of new quality expectations, and a realignment of the entire field's economics. The implications are far-reaching.

The main text of this article focuses on *high availability*, one of several key properties today's cloud computing applications demand. The need is most obvious in voice-over-IP (VoIP) telephony and video streaming: for such uses, even the briefest disruptions can cause connections to seize up or fail in ways that are highly visible to the end user. If we can crack the "high-availability barrier," we can imagine a future in which the Internet carries all such traffic.

Yet high availability is merely the first step in what will be an evolutionary process. Cloud applications also need better techniques for guaranteeing steady, very high data rates; the ability to prioritize traffic; and robustness under routing-level attacks. Content-distribution networks have been central to the static Web's success: What will be the analogous paradigm for the Web of dynamic content, such as video streams shared by large numbers of users, gaming applications, or virtual reality immersion? The answers to such questions could transform the Internet's roles.

Indeed, many cloud computing uses are so important (both in the terms of their scale and the associated revenue streams) that unless the Internet can evolve to meet the demands, the associated cloud computing enterprises might consider building new networks that would be dedicated to their use. Companies such as Google, Netflix, Amazon, Microsoft, and others are insisting on the need to craft virtual enterprise networks. If these are to share the same optical fibers used for other purposes, these and other cloud computing providers will need guarantees of disruption-free bandwidth, predictable latencies, and hands-on control of routing policy control: "my traffic from A to B will traverse such-and-such a route," or "requests from user X will be routed to data center Y," to list just a few examples. A new network-control paradigm has emerged (the so-called Open Flow standard; www.openflow.org) with enthusiastic backing from the cloud computing community. Moreover, with such a large part of the economy Internet-dependent, there are growing calls to harden the network so that it can offer rock-solid defense against attackers, be they hackers or cyber warriors under command of national adversaries.

The challenges are significant, but the payoff will also be big. Today, many of the top technical people in the field are racing to offer competing ideas. For many of the topics listed, rather than having no solutions, we might soon have a buffet of choices to pick from. These are exciting times to work in the field of networking, and the best part of the story is that so much of it has yet to be written.

(or even better) quality of service seen in wired telephone networks, and that other kinds of streaming media applications could be deployed with sharply improved quality guarantees relative to what's feasible today. High-availability routers, however, are just one of many developments that will slowly reshape the Internet in response to the challenge and opportunity cloud computing represents — the sidebar "An Internet for the Cloud" describes how our efforts fit into this shifting computing landscape.

Acknowledgments

We are deeply grateful to professors Jonathan Smith (University of Pennsylvania) and Doug Comer (Purdue University) for helping us identify this research topic, and for their encouragement and advice at many stages. We also thank John Denisco for his invaluable assistance. Our effort was supported by Cisco and is a part of the NEBULA project within the US National Science Foundation's Future Internet Architectures (FIA) program (see http://r3.cis.upenn.edu/paperspdfs/R3_WP_Full.pdf).

References

1. C. Labovitz, G.R. Malan, and F. Jahanian, "Internet Routing Instability," *IEEE/ACM Trans. Networking*, vol. 6, no. 5, 1998, pp. 515–526.
2. E. Keller, J. Rexford, and J. van der Merwe, "Seamless BGP Migration with Router Grafting," *Proc. Networked Systems Design and Implementation* (NSDI 10), Usenix Assoc., 2010, pp. 16–30.

Andrei Agapi is a PhD student at Vrije Universiteit, Amsterdam, and a software engineer with Cisco Systems. Contact him at aagapi@few.vu.nl.

Ken Birman is the N. Rama Rao Professor of Computer Science at Cornell University. Contact him at ken@cs.cornell.edu.

Robert M. Broberg leads the Reliable Router Research Effort and is a Distinguished Engineer at Cisco Systems. Contact him at rbroberg@cisco.com.

Chase Cotton is a senior scientist with the University of Delaware. Contact him at ccotton@udel.edu.

Thilo Kielmann is an associate professor at Vrije Universiteit, Amsterdam. Contact him at kielmann@cs.vu.nl.

Martin Millnert is writing his master thesis at Cisco Systems. Contact him at martin@millnert.se.

Rick Payne is a software engineer at Cisco Systems. Contact him at rpayne@cisco.com.

Robert Surton is a PhD student at Cornell University. Contact him at burgess@cs.cornell.edu.

Robbert van Renesse is a principal research scientist with the Department of Computer Science at Cornell University. Contact him at rvr@cs.cornell.edu.