Smart Proxy - Bug #20474

Multiple free IPs returned after record deletion

08/01/2017 09:33 AM - Lukas Zapletal

Status:	Closed		
Priority:	High		
Assignee:			
Category:	DHCP		
Target version:			
Difficulty:		**Fixed in Releases:**	
Triaged:		**Found in Releases:**	
Bugzilla link:	1459644	**Red Hat JIRA:**	
Pull request:	https://github.com/theforeman/smart-proxy/pull/543		

Description

In #20173 we fixed possible race condition, but there is one more:

- create reservation
- call unused_ip and X is returned
- delete the above reservation
- call unused_ip and Y is returned

Now, the contract is expected to satisfy X != Y but it's not the case right now thanks to the design of our unused_ip method (rotating list shifts to the left on delete returning already returned IP). I think we need to rethink how this is done.

We should also not use /tmp as the directory, it's getting deleted via systemd timer in RHEL7 or other distros as well.

Related issues:

Related to Foreman - Bug #20475: Implement Random DB IPAM	**Closed**	**08/01/2017**
Related to Smart Proxy - Bug #20173: Concurrent calls to Subnet#unused_ip may...	**Closed**	**06/30/2017**

Associated revisions

Revision 54f44d80 - 11/03/2017 08:18 AM - Dmitri Dolguikh

Fixes #20474 - return different ips on sequential unused_ip calls

History

#1 - 08/01/2017 09:34 AM - Lukas Zapletal

- Description updated

#2 - 08/01/2017 09:49 AM - Lukas Zapletal

- Related to Bug #20475: Implement Random DB IPAM added

#3 - 08/01/2017 09:49 AM - Lukas Zapletal

- Related to Bug #20173: Concurrent calls to Subnet#unused_ip may return the same ip address added

#4 - 08/08/2017 10:21 AM - Lukas Zapletal

- Bugzilla link set to 1459644

Randomizing will look like the best solution. We can seed random generator from MAC address as we did in #20475.

#5 - 09/01/2017 05:42 AM - Lukas Zapletal

Foreman core now contains randomized IPAM DB implementation, we need similar for Smart Proxy. Ideally this would be an implementation that Ivan this is customer-facing, can you put this on your scrum?

```
module IPAM
  # Internal DB IPAM returning all IPs in random order to minimize race conditions
```

```
  class RandomDb < Base
    def generator
       @generator ||= Random.new(mac ? mac.gsub(':', '').to_i(16) : Random.new_seed)
    end

    def random_ip
      IPAddr.new(generator.rand(subnet_range.first.to_i..subnet_range.last.to_i), subnet.family)
    end

    # Safety check not to spend much CPU time when there are no many free IPs left. This gives up
    # in about a second on Ryzen 1700 running with Ruby 2.4.
    MAX_ITERATIONS = 100_000
    def suggest_ip
      iterations = 0
      loop do
        # next random IP from the sequence generated by MAC seed
        candidate = random_ip
        iterations += 1
        break if iterations >= MAX_ITERATIONS
        # try to match it
        ip = candidate.to_s
        if !excluded_ips.include?(ip) && !subnet.known_ips.include?(ip)
          logger.debug("Found #{ip} in #{iterations} iterations")
          return ip
        end
      end
      logger.debug("Not suggesting IP Address for #{subnet} as no free IP found in reasonable time (#{iteratio
ns} iterations)")
      errors.add(:subnet, _('no random free IP could be found in our DB, enlarge subnet range'))
      nil
    end
  end
end
```

The following unit tests simulates the race condition:

```
  def test_unused_ip_with_concurrent_record_add
    @subnet.stubs(:icmp_pingable?)
    @subnet.stubs(:tcp_pingable?)
    records = []
    records << Proxy::DHCP::Reservation.new('test', "192.168.0.1", "aa:bb:cc:dd:ee:01", @subnet, :hostname =>'
test_01')
    assert_equal "192.168.0.2", @subnet.unused_ip(records)
    assert_equal "192.168.0.3", @subnet.unused_ip(records)
    records << Proxy::DHCP::Reservation.new('test', "192.168.0.2", "aa:bb:cc:dd:ee:02", @subnet, :hostname =>'
test_02')
    assert_equal "192.168.0.4", @subnet.unused_ip(records) # this fails - returns 192.168.0.5
  ensure
    File.delete('test/tmp/foreman-proxy_192.168.0.0_24.tmp')
  end
```

```
  def test_unused_ip_with_concurrent_record_delete
    @subnet.stubs(:icmp_pingable?)
    @subnet.stubs(:tcp_pingable?)
    records = []
    records << Proxy::DHCP::Reservation.new('test', "192.168.0.1", "aa:bb:cc:dd:ee:01", @subnet, :hostname =>'
test_01')
    records << Proxy::DHCP::Reservation.new('test', "192.168.0.2", "aa:bb:cc:dd:ee:02", @subnet, :hostname =>'
test_02')
    assert_equal "192.168.0.3", @subnet.unused_ip(records)
    records.pop
    assert_equal "192.168.0.4", @subnet.unused_ip(records)  # this fails - returns 192.168.0.3
  ensure
    File.delete('test/tmp/foreman-proxy_192.168.0.0_24.tmp')
  end
```

And the following unit test never returns (CPU spikes at 100%) because Ruby is busy with creating 16 million records array in memory. The randomized can solve that:

```
  def test_unused_aclass_ip_beware_this_never_finishes
    @network = "10.0.0.0"
    @netmask = "255.0.0.0"
    @subnet = Proxy::DHCP::Subnet.new @network, @netmask
    @subnet.stubs(:icmp_pingable?)
    @subnet.stubs(:tcp_pingable?)
```

```
    r = Proxy::DHCP::Reservation.new('test', "10.0.0.1", "aa:bb:cc:dd:ee:ff", @subnet, :hostname =>'test1')
    assert_equal "10.0.0.2", @subnet.unused_ip([r])
  ensure
    File.delete('test/tmp/foreman-proxy_10.0.0.0_8.tmp')
  end
```

Ideally the code is refactored so unused_ip has it's own providers injected into subnet/server and we provide two implementations: the current one (perhaps fixed so it uses less memory) and random which I believe is very easy to implement (see the code bit from core) and workarounds the issue nicely.

#6 - 09/01/2017 05:42 AM - Lukas Zapletal

- Priority changed from Normal to High

BZ set, customer facing.

#7 - 09/07/2017 05:58 PM - Anonymous

- Status changed from New to Assigned

- Assignee set to Anonymous

#8 - 09/13/2017 04:38 PM - The Foreman Bot

- Status changed from Assigned to Ready For Testing

- Pull request https://github.com/theforeman/smart-proxy/pull/543 added

#9 - 11/03/2017 09:01 AM - Anonymous

- Status changed from Ready For Testing to Closed

- % Done changed from 0 to 100

Applied in changeset 54f44d807b8540fd3561a0d16ab68c68deb88c0f.