Understanding NTP Reachability Statistics

Jan 05, 2004 By Todd Jacobs (/user/801247)
in

How to make sense out of reachability statistics to learn what and where the problem is.

You're a typical system administrator, with a million and two urgent things to do before you get to go home for the day. Next on your list is bringing up a new NTP server for your department so your systems can use the Network Time Protocol to synchronize their clocks with a reference time source (see Sidebar). You've already compiled and installed ntpd for your platform and are ready to take the next step.

NTP: The Network Time Protocol

Keeping accurate time is an important aspect of both system administration and network security. If you've ever had the dubious pleasure of trying to match up log events on multiple systems, you know how critical accurate timestamps can be in tracking an event as it flows through your network.

On a busy server, you may have dozens of similar log entries every second. If your system clocks are even a few seconds apart, you may have tremendous difficulty determining which event on Server A matches up with the same event on Server B. NTP addresses these issues by keeping system clocks synchronized with a reference time source, often to within a few milliseconds even over the open Internet.

Any system clock can be designated as the master clock in an NTP infrastructure, regardless of whether or not it keeps accurate time. Rather than using an arbitrary time source, most enterprise-class networks synchronize a local master clock with Universal Coordinated Time (UTC) as maintained by the National Institute of Standards and Time (NIST) or the US Naval Observatory (USNO).

Systems within an enterprise then should synchronize with the local master clock rather than directly to a remote time server. This reduces network latency and jitter for local time clients, conserves bandwidth on WAN links and limits the load on public time servers that otherwise might be overwhelmed by an excess of individual client connections.

The proper use of NTP ensures a consistent time base for all internal systems. In addition, synchronizing to UTC enables you to share the same time base with any external systems that have done the same.

Start your NTP daemon. Curious to see how well it's syncing up, you monitor the output of ntpq (see Listing 1) and watch the reachability statistics in the reach field climb towards the mysterious upper bound of 377. At last, the system has reached that exalted state, and all is well with the world.

```
$ watch ntpq -pn

Every 2s: ntpq -pn                                    Thu Apr 17 13:41:29 200

     remote           refid      st t when poll reach   delay   offset  jitte
==============================================================================
*192.168.38.127  .GPS.            1 -   46   64  377   43.763   -5.586   0.70
```

You go celebrate with a mocha latte, only to return to find the reach field now reads 357 (see Listing 2). Concerned, you decide to investigate.

```
     remote           refid      st t when poll reach   delay   offset  jitte
==============================================================================
*192.168.38.127  .GPS.            1 -   16   64  357   23.994  -10.649   2.56
```

The poll field tells you the daemon polls the remote server every 64 seconds. The when field shows that the last poll was 16 seconds ago. That leaves you plenty of time to set up a packet capture to see what's happening to your precious time packets.

```
# The "-n" flag turns off name resolution for both
# speed and readability, and to prevent DNS traffic
# from cluttering the results if you don't specify a
# specific port. The "-i" flag specifies which
# interface to monitor (usually eth0 or eth1). The
# host and port flags limit the output of tcpdump to
# NTP traffic to and from the specified IP address.
tcpdump -n -i eth1 host 192.168.38.127 port 123
```

Use tcpdump (Listing 3) to look for NTP packets. You let tcpdump run for awhile, and see several NTP transactions similar to this one:

```
21:04:45.446566 172.16.24.16.ntp > 192.168.38.127.ntp:  v4 client  strat 2 po

21:04:45.485103 192.168.38.127.ntp > 172.16.24.16.ntp:  v4 server  strat 1 po
```

Hmmm. Looks like the NTP packets are flowing freely. But a quick check of ntpq shows that reachability has now dropped to 177. What's going on here?

It turns out that the explanation relates back to the strange upper bound of 377 for the reach field. This field is not, as one would assume, a sequential counter of successful connections. Instead, it is a circular log buffer containing a set of eight bit-flags.

Each remote server or peer is assigned its own buffer by ntpd. This buffer represents the status of the last eight NTP transactions between the NTP daemon and a given remote time server. Each bit is a boolean value, where a 1 indicates a successful transaction and a 0 indicates a failure. Each time a new packet is sent, the entire eight-bit register is shifted one bit left as the newest bit enters from the right.

The net result is that dropped packets can be tracked over eight poll intervals before falling off the end of the register to make room for new data. This recycling of space in the register is why it's called a circular buffer, but it may make more sense to think of it in linear terms, as a steady, leftward march--eight small steps, and then the bit ends up wherever bits go when they die.

For reasons that seemed good to the developers, this register is displayed to the user in octal values instead of binary, decimal or even hex. The maximum value of an eight-bit binary number is 11111111, which is 377 in octal, 255 in decimal and 0xFF in hex.

So why does the value of the reach field drop when packets are being successfully sent and received? For those of you who dream in octals, this next part may seem obvious. For ordinary mortals, it requires closer scrutiny.

The answer is that the lower numerical values are caused by the left-shifting of the reachability register. Remember, the buffer is not a metric, it is a FIFO log. So, if you have received the last eight NTP packets successfully, the log contains all 1s and the reach field contains the octal value of 377.

Let's assume that on the next update, a packet is dropped. Because NTP is a UDP-based protocol with no delivery guarantees, this is not necessarily a cause for alarm. But the NTP daemon dutifully logs the failure in the circular buffer and waits for the next poll period. The log now contains 11111110 and a reach field value of 376.

If the next seven polls are successful, seven 1s are added from the right-hand side of the register, pushing the 0 representing the dropped packet further towards the left (and digital oblivion). Listing 4 shows the progression of a single dropped packet through the reachability register.

```
 Buffer     Octal    Decimal
--------    -----    -------
11111110     376       254
11111101     375       253
11111011     373       251
11110111     367       247
11101111     357       239
11011111     337       223
10111111     277       191
01111111     177       127
11111111     377       255
```

As you can see, the 0 representing the dropped packet is moved one bit to the left every time the daemon polls its server. The numerical value assigned to each bit is higher on the left than on the right. In the binary system, the leftmost digit holds a value of 128, while the rightmost digit represents a value of 1. So, as the zero moves leftwards, it actually produces a lower numerical value, despite the fact that its distance from the right-hand side of the register represents an increase in the time since the packet was dropped.

If the zero falls off the end of the register, and no other packets have been dropped, the value of the reach field jumps back to 377 with no intervening steps. This can be very confusing if you insist on viewing the numbers as a connection metric rather than as a history log.

For a simple script that can parse your reach fields without all that dangerous mucking around with rusty bits and pointy octals, see the ntp_packets.sh script (Listing 5). Listing 6 shows the script's output from an NTP server that failed three consecutive polls but appears to be back on track.

```
#!/bin/bash

################################################
# ntp_packets.sh
# --------------
#
#       Show FIFO log of the last 8 packets for each
#       NTP server or peer in a more human-readable
#       format.
################################################
```

```
# Convert circular buffer's octal value to decimal,
# and then compute the binary value that represents
# the boolean received status for the past 8 NTP
# packets.
display_buffer () {
    IP=$1
    unset packets
    let buffer=0$2
    while [ $buffer -ne 0 ]; do
        packets=$(( buffer %  2 ))$packets
        buffer=$((  buffer >> 1 ))
    done
    # Indented EOF flag at end of here-document must
    # be preceded only by a tab, or the script
    # breaks.
    cat <<-EOF
        Last 8 packets received from $IP:
            8......1
            $packets
        EOF
}

# Parse the output of ntpq to get the IP address and
# reachability value for all peers and servers.
# xargs won't exec functions, so we'll store the
# values as positional parameters instead.
set -- $(
    /usr/sbin/ntpq -pn  | awk '{print $1, $7}' |
    sed -e 's/^[*+-]//' | grep '^[[:digit:]]'
)

# Display output.
while [ $# -ge 2 ]; do
    display_buffer $1 $2
    shift 2
done
```

```
Last 8 packets received from 192.168.38.127:
    8......1
    10001111
```

Now that we've unlocked this deep, dark mystery, you can finally relax. There's nothing wrong with your NTP server, which only leaves a busy sysadmin like you with a million and one urgent matters left to solve before your next mocha latte.

Todd A. Jacobs, CISSP CCNA LPIC (**articles@codegnome.org** (mailto:articles@codegnome.org)) is president of **CodeGnome Consulting, LTD** (http://www.codegnome.org) . In addition to spending way too much of his free time researching obscure technical topics, Todd likes riding his Honda Shadow, reading fantasy novels and listening to the latest *Doctor Who* audio dramas from Big Finish Productions.

Comments

Comment viewing options

Threaded list - expanded | Date - newest first | 50 comments per page | Save settings

Select your preferred way to display the comments and click "Save settings" to activate your changes.

Superb! Thanks for the (/article/6812#comment-334043)
Submitted by KevinB (not verified) on Mon, 03/02/2009 - 13:07.

Superb! Thanks for the explanation.

superb explanation (/article/6812#comment-206918)
Submitted by Anonymous (not verified) on Mon, 12/18/2006 - 02:18.

I once got all the details out docs, much labour. Your article is much more concise and a so much easier to understand.

Great work!

Re: Understanding NTP Reachability Statistics (/article/6812#comment-8095)
Submitted by Anonymous on Tue, 08/24/2004 - 02:00.

Excellent article. Nice explaination, just right on the details. Thank you !

anupam

Re: Understanding NTP Reachability Statistics (/article/6812#comment-8092)
Submitted by Anonymous on Wed, 01/07/2004 - 03:00.

Excellent article! Simple, consise with the right level of technical. One more building block in the wall of NTP understanding.

Re: Understanding NTP Reachability Statistics (/article/6812#comment-8094)
Submitted by Anonymous on Fri, 02/20/2004 - 03:00.

Finally the "ntpq -p" command makes sense...
Great explanation, documentation, example & script..

Regards from **Rio de Janeiro, Brazil**

Re: Understanding NTP Reachability Statistics (/article/6812#comment-8093)
Submitted by Anonymous on Sun, 02/08/2004 - 03:00.

Great Insight ... Difficult to find such explanations on net .