Test Your Network's Multicast Capability

November 9, 2010

Table of Contents

Tools and instruction to test your network's multicast connectivity and performance. The tools package contains documentation, source code and pre-compiled executables for Windows and a variety of Unixes.

1. Introduction

Many organizations are looking to switch their high-speed messaging from TCP to Multicast. This can drastically improve their performance in streaming data applications, such as live market data, where you have many subscriber programs. However, many corporate network infrastructures provide limited multicast connectivity or are not able to sustain high message rates. Even sites that are successfully using multicast with legacy messaging systems encounter problems after an upgrade to a higher-performing messaging layer.

Fortunately, most modern networking hardware is capable of passing wire-speed multicast with appropriate configuration parameters. The key is discovering configuration problems early and diagnosing them. The "msend", "mdump", and "mpong" tools can help. We at Informatica invite you to use them to evaluate your network's capacity to carry multicast traffic.

> **Warning**: If used carelessly, the msend tool can produce significant network loading and congestion, to the point of rendering switches and routers virtually non-functional. When using large burst counts (-b>500) and/or small pause times (-p<100), make sure to keep the number of bursts (-n) small enough to limit the duration of the test to a few seconds.

RELATED TO THIS AGREEMENT OR THE TRANSACTIONS CONTEMPLATED HEREUNDER, EVEN IF INFORMATICA HAS BEEN APPRISED OF THE LIKELIHOOD OF SUCH DAMAGES.

2. Tool Notes

The source and a variety of binaries of the tools are available free of charge on the Informatica Marketplace (https://community.informatica.com/solutions/1470). After expanding the files, the appropriate binary then can be invoked from a command shell with the "-h" option to get a help screen. The binary directory also contains a script that was used to build the binaries.

```
msend -h

Usage: msend [-1|2|3|4|5] [-b burst_count] [-d] [-h] [-l loops] [-m msg_len] [-n num_bursts] [-p pau
Where:
  -1 : pre-load opts for basic connectivity (1 short msg per sec for 10 min)
  -2 : pre-load opts for long msg len (1 5k msg each sec for 5 seconds)
  -3 : pre-load opts for moderate load (bursts of 100 8K msgs for 5 seconds)
  -4 : pre-load opts for heavy load (1 burst of 5000 short msgs)
  -5 : pre-load opts for VERY heavy load (1 burst of 50,000 800-byte msgs)
  -b burst_count : number of messages per burst [1]
  -d : decimal numbers in messages [hex])
  -h : help
  -l loops : number of times to loop test [1]
  -m msg_len : length of each message (0=use length of sequence number) [0]
  -n num_bursts : number of bursts to send (0=infinite) [0]
  -p pause : pause (milliseconds) between bursts [1000]
  -P payload : hex digits for message content (implicit -m)
  -q : loop more quietly (can use '-qq' for complete silence)
  -s stat_pause : pause (milliseconds) before sending stat msg (0=no stat) [0]
  -S Sndbuf_size : size (bytes) of UDP send buffer (SO_SNDBUF) [65536]
                   (use 0 for system default buff size)
  -t : tcp ('group' becomes destination IP) [multicast]
  -u : unicast udp ('group' becomes destination IP) [multicast]

  group : multicast group or IP address to send to (required)
  port : destination port (required)
  ttl : time-to-live (limits transition through routers) [2]
  interface : optional IP addr of local interface (for multi-homed hosts)

mdump -h

Usage: mdump [-h] [-q] [-Q Quiet_lvl] [-r rcvbuf_size] [-p pause_ms/num] [-v] [-s] group port [inter
Where:
  -h : help
  -o ofile : print results to file (in addition to stdout)
  -p pause_ms[/num] : milliseconds to pause after each receive [0: no pause]
                      and number of loops to apply the pause [0: all loops]
  -Q Quiet_lvl : set quiet level [0] :
                 0 - print full datagram contents
```

```
                1 - print datagram summaries
                2 - no print per datagram (same as '-q')
  -q : no print per datagram (same as '-Q 2')
  -r rcvbuf_size : size (bytes) of UDP receive buffer (SO_RCVBUF) [4194304]
                   (use 0 for system default buff size)
  -s : stop execution when status msg received
  -t : Use TCP (use '0.0.0.0' for group)
  -v : verify the sequence numbers

  group : multicast address to receive (required, use '0.0.0.0' for unicast)
  port : destination port (required)
  interface : optional IP addr of local interface (for multi-homed hosts) [INADDR_ANY]

mpong -h

Usage: ./mtools/FreeBSD-6-i386/mpong [-h] [-i] [-o ofile] [-r rcvbuf_size] [-S Sndbuf_size] [-s samp
Where:
  -h : help
  -i : initiator (sends first packet) [reflector]
  -o ofile : print results to file (in addition to stdout)
  -r rcvbuf_size : size (bytes) of UDP receive buffer (SO_RCVBUF) [4194304]
                   (use 0 for system default buff size)
  -S Sndbuf_size : size (bytes) of UDP send buffer (SO_SNDBUF) [65536]
                   (use 0 for system default buff size)
  -s samples : number of cycles to measure [65536]
  -v : verbose (print each RTT sample)

  group : multicast address to send on (use '0.0.0.0' for unicast)
  port : destination port
  interface : optional IP addr of local interface (for multi-homed hosts) [INADDR_ANY]

Note: initiator sends on supplied port + 1, reflector replies on supplied port
```

The tools, `msend` in particular, have many options which are useful in diagnosing a variety of multicast problems. To make the initial evaluation of the network easy, the `msend` command has 5 numbered command-line options (*`-1`* through *`-5`*) which pre-load other options to values which test for the most common network problems.

Both the `msend` and `mdump` commands take zero or more Unix-style options, followed by two required positional parameters: multicast address (group), and destination port. In the examples listed in this document, addresses and ports are used by default by Informatica's **UMS** product (formerly called "LBM"), but any addresses and ports can be used. However, for each step in the initial 5-step test, it is important to use *different* multicast addresses for each step. In fact, if you desire to run a particular test a second time, it is a good idea to chose a multicast address that is different from the previous run of the same test. (This is because the tests are verifying different ways that a multicast stream is initiated, so re-running a test on a multicast address that the switch and router is already initialized for will prevent the test from verifying the stream initiation condition.)

The `msend` command has an optional third positional parameter which is the TTL (Time To Live) for the sent packets. This field controls how widely the multicast datagrams can be distributed in an interconnected group of networks. A TTL of 1 prevents the datagram from crossing any routers, restricting the datagrams to a single network segment (LAN). However, many switches and routers actually operate *less* efficiently with a TTL of 1, so the tool defaults to a TTL of 2. It is assumed that you want to verify the performance of your routers, so the examples below

use a TTL of 2. For some organizations, multiple routers may need to be crossed, which would require the TTL to be increased.

The `msend` command has an optional forth positional parameter which is the IP address of the desired network interface. The `mdump` command also can specify a network interface as its third positional parameter. This parameter is only needed on a multi-homed host (machine with more than one network interface). With normal unicast destination addresses, IP uses routing tables to determine the correct interface to send packets. With multicast, there is no "correct" interface - the application should specify which interface to use. (It is possible for an application to let the operating system choose an interface, but this can lead to systems that work properly for a time, and then mysteriously stop working due to minor hardware or operating system upgrades. It is a much better idea to specify the correct interface for your network architecture.) In the examples below, it is assumed that the sending and receiving hosts are single-homed, and therefore do not require the interface.

The `mdump` command attempts to set its socket to have a 4 MB UDP receive buffer. Many operating systems will not grant a request that large (the tool will inform you how large a buffer it was able to get). However, if your sending machine is a reasonably high-power machine, the receiver may very well need a large receive buffer. You can use the "netstat" command (usually with the "-s" option) to see UDP statistics, including packets dropped due to the receive buffer overflowing. If you consistently get loss with tests 3 - 5, it may be necessary to have your system administrator increase the maximum allowable receive buffer (it's a kernel tuning parameter). See our THPM document (http://www.29west.com/docs/THPM/thpm.html#SETTING-KERNEL-UDP-BUFFER-LIMITS) for advice in this area.

The `msend` command attempts to set its socket to have a 64 KB UDP send buffer. Many operating systems will not grant a request that large (the tool will inform you how large a buffer it was able to get). However, we have found that especially when sending fairly large datagrams, less than 64 KB of send buffer prevents the sending machine from reaching its maximum send rate. In fact, if your sending datagrams are themselves approaching 64 KB in size, you may need an even larger send buffer, perhaps three times the maximum datagram size. However, due to a possible bug in Linux, you should not set your send socket buffer more than 192 KB (196608).

Finally, it is very possible to experience loss due to a sufficiently-large capability mismatch between the sending and receiving machines. It might be because the sending machine has a faster network interface than the receiving machine, or a faster CPU, or an operating system that is more efficient at processing UDP. For the purposes of evaluating your network, it is suggested that the sending and receiving machines be as closely-matched as possible.

3. The Initial Five Tests

3.1. Test 1

Test 1 sends one small datagram per second for 10 minutes. It initially tests simple connectivity, and after several minutes verifies that multicast streams are properly maintained over several IGMP query timeouts. Start the `mdump` first, then after a second or two start the `msend`.

Receiving Host:

```
mdump -omdump1.log 224.9.10.11 12965 10.1.2.3
```

Sending Host:

```
msend -1 224.9.10.11 12965 15 10.1.2.4
```

(Note: host interface addresses 10.1.2.* should be changed to reflect your hosts.) This test will run for 10 minutes and then report the percentage of dropped messages (datagrams). During that time, `mdump` will display the received messages in a hex dump form. At the end of the test run, `msend` will tell `mdump` to display statistics. You should use `ctrl-C` to exit `mdump` when test 1 finishes.

Be sure to run the test a second time, switching the roles of sender and receiver, and remember to use a different multicast address for that second run. Also, note that other tests use the "-q" option on the mdump command. However, test 1 does not use "-q".

3.2. Test 2

Test 2 sends one large datagram per second for 5 seconds. It tests the ability of the network hardware to establish a multicast stream from a fragmented datagram. Start the `mdump` first, then after a second or two start the `msend`.

Receiving Host:

```
mdump -q -omdump2.log 224.10.10.10 14400 10.1.2.3
```

Sending Host:

```
msend -2 224.10.10.10 14400 15 10.1.2.4
```

(Note: host interface addresses 10.1.2.* should be changed to reflect your hosts.) Notice that this and subsequent `mdump` commands include the "-q" option to prevent the hex dump of each datagram. In this test it is for convenience; in future tests it is necessary to prevent slow receiver operation.

Be sure to run the test a second time, switching the roles of sender and receiver, and remember to use a different multicast address for that second run.

3.3. Test 3

Test 3 sends 50 bursts of 100 datagrams (8K each). Each burst of 100 is sent at the maximum possible send rate for the machine usually fully saturating the wire), and the bursts are separated by a tenth of a second. This is a pretty heavy load that tests the ability of the network hardware to establish a wire-speed multicast stream from a fragmented datagram. Start the `mdump` first, then after a second or two start the `msend`.

Receiving Host:

```
mdump -q -omdump3.log 224.10.10.14 14400 10.1.2.3
```

Sending Host:

```
msend -3 224.10.10.14 14400 14 10.1.2.4
```

(Note: host interface addresses 10.1.2.* should be changed to reflect your hosts.) Depending on the speed of the machine, this test should not run much longer than 7 seconds, usually much shorter.

Be sure to run the test a second time, switching the roles of sender and receiver, and remember to use a different multicast address for that second run.

3.4. Test 4

Test 4 sends a single burst of 5000 datagrams (20 bytes each). The burst is sent at the maximum possible send rate for the machine. It may not fully saturate the wire, but does lead to a very high message rate during the burst. This is a another heavy load that tests the ability of the network hardware to sustain a high message rate multicast stream. Start the `mdump` first, then after a second or two start the `msend`.

Receiving Host:

```
mdump -q -omdump4.log 224.10.10.18 14400 10.1.2.3
```

Sending Host:

```
msend -4 224.10.10.18 14400 15 10.1.2.4
```

(Note: host interface addresses 10.1.2.* should be changed to reflect your hosts.) Depending on the speed of the sending machine, this test should not run much more than 5 seconds, often much less.

Be sure to run the test a second time, switching the roles of sender and receiver, and remember to use a different multicast address for that second run.

3.5. Test 5

Test 5 sends a single burst of 50,000 datagrams (800 bytes each). The burst is sent at the maximum possible send rate for the machine. This test generates the heaviest load of the 5 tests, and should saturate a 1-gig link. Start the `mdump` first, then after a second or two start the `msend`.

Receiving Host:

```
mdump -q -omdump5.log 224.10.10.18 14400 10.1.2.3
```

Sending Host:

```
msend -5 224.10.10.18 14400 15 10.1.2.4
```

(Note: host interface addresses 10.1.2.* should be changed to reflect your hosts.) Depending on the speed of the sending machine, this test should not run much more than 5 seconds, often much less.

If this test experiences loss, re-run the msend command with the option "-S65536". If this option removes the loss, then your system default UDP send buffer size is too large. Many Linux systems suffer from this if the UDP send buffer is larger than a few hundred KB. We recommend setting the default to either three times your maximum datagram size, or 64 KB, whichever is larger. If that is not desirable, then we recommend configuring your multicast applications to override the system default UDP send buffer size.

It is a good idea during the execution of this test for the network hardware administration team to monitor switch CPU usage. We have seen cases where switches that handle multicast in hardware still overload the switch CPU when high-rate multicast is used. For example, we saw one user of Cisco hardware enable an ACL, with the result that the CPU had to examine each multicast packet. This left his Cisco switch at 90% CPU utilization even though he was only using about half of the gigabit bandwidth. It is always better to discover this kind of CPU loading early, rather than on the "go-live" day.

Be sure to run the test a second time, switching the roles of sender and receiver, and remember to use a different multicast address for that second run.

3.6. Next Steps

It is beyond the scope of this simple document to attempt to fully diagnose and describe the treatments of various multicast networking maladies. Networking routers and switches are too diverse. However, you can find a wealth of general information in our THPM document (http://www.29west.com/docs/THPM/thpm.html#SETTING-KERNEL-UDP-BUFFER-LIMITS). If you suspect that your network infrastructure is not able to handle high-speed multicast traffic, there is a very good chance that it is simply a matter of switch and router configuration. We have found that network administrators, working with the network hardware's support team, are usually successful at enabling the proper hardware multicast routing parameters. It sometimes requires a bit of patience and digging, but the scaling advantages of multicast are well worth the effort.

4. Mpong

The `mpong` commad can be used to get a very rough idea of the round-trip latency between two machines. However, be aware that this tool does not contain optimizations which could further decrease latencies; for example, it does not set CPU affinity or modify thread priority. It also does not contain logic to detect lost packets and retransmit them. Nor does it provide topic-based PUB/SUB model of usage.

However, the tool can still be useful in reproducing latency problems with a simple tool which can be provided in source form to support organizations for diagnosis. For example:

```
MACHINE 1: mpong 224.1.3.5 12000

MACHINE 2: mpong -iv -ompong.raw 224.1.3.5 12000
```

Machine 2, the initiator (`-i`) will start the test by sending a packet to Machine 1, the reflector, over group 224.1.3.5 port 12001. The packet will contain the sending timestamp. Machine 1 will receive it and send it back (reflect) over the same group but with port 12000. This is one round-trip cycle. The first 20 cycles are for "warm-up", followed by the measurement phase, consisting of 65536 (default) cycles. The initiator prints the results and exits. The reflector does not exit and must be killed.

Notice that both commands are provided port number 12000; the code takes care of incrementing it appropriately. The `-v` option forces verbose output, resulting in a large dataset being written to file `plot.raw`.

> **Warning**: Since this tool's multicast traffic is UDP, it is possible to lose a packet during the test. This will result in a hang of the test since no timeouts are programmed. If several tries continue to result in loss, there is something seriously wrong with your multicast connectivity, and that needs to be diagnosed and resolved before meaningful latency measurements are possible. You should use the initial 5 tests to verify adequate connectivity before using mpong.

A quick and easy plot of the data can produced with gnuplot (http://www.gnuplot.info/). Here is a simple unix shell script:

```
#!/bin/sh
# remove text lines (select lines starting with numerics)
grep "^[0-9]" mpong.raw >mpong.dat
gnuplot <<__EOF__
```

```
## IF TYPE eq ps   set terminal postscript landscape color
## IF TYPE eq eps  set terminal postscript eps color
set terminal jpeg
## IF TYPE eq png  set terminal png
## IF TYPE eq pbm  set terminal pbm
set output "mpong.jpg"
set xlabel "time (sec)"
set ylabel "RTT (usec)"
set multiplot
set autoscale
set data style lines
set border 3
set xtics border nomirror
set ytics border nomirror
set origin 0.0,0.0
set title "mpong results"
set style line 10 lt 1 lw 1 pt 5 ps 0.65
plot 'mpong.dat' using 1:2 title 'mpong.dat' with linespoints linestyle 10
__EOF__
```

This should produce `mpong.jpg`