Inferno Nettverk A/S

Technical Report

Analysis of Real-Time scheduling functionality in Dante version 1.3.2.2

March 7, 2012

Contents

List of Figures

1 Introduction

This report documents the results of an analysis of the performance and behavior of the real-time related functionality found in version 1.3.2.2 of the Dante SOCKS server implementation from Inferno Nettverk A/S. This version is not publicly available, but the functionality described in this document will likely be included in the upcoming public 1.4.0 version.

Two new keywords have been added in this version that makes it possible to specify the real-time scheduling algorithm and priority of Dante process types. Dante processes can also be bound to specific CPUs (CPU affinity).

2 Test environment

The tests described in this document are made in a simple test environment with three test machines and synthetic loads. Two different network switches are used. One switches traffic between the SOCKS clients and the Dante proxy, while the other switches traffic between the Dante proxy and the remote server. There is no other traffic of significance between the three machines at the time of testing. In some of the tests a second server machine is also used to receive traffic, or an extra client machine is used to generate traffic.

2.1 Machine Description

The machine used to run Dante has a 2 core 2.4 GHz *Intel Core2* processor and two one-Gigabit networking cards. Traffic to and from the client is transmitted over one interface and traffic to and from the server is transmitted over the other interface. The machine runs Linux, with kernel version 2.6.18 installed.

2.2 Dante Configuration

A snapshot of Dante 1.3.2.2 is used for testing. No special configure options are used and debugging is not enabled during compilation.

A simple configuration that forwards traffic is used in the server. Traffic information is at most only logged at the beginning and end of client sessions. This configuration is similar to what would be expected in a production environment.

Several variations of the real-time keywords are used in the tests. The following shows an example where the scheduling algorithm, priority, and the CPUs to use, is specified:

```
#enable real-time scheduling scheduling
cpu.schedule.mother: fifo/10
cpu.schedule.negotiate: fifo/10
cpu.schedule.request: fifo/10
cpu.schedule.io: fifo/15

#use only CPUs 0-2
cpu.mask.mother: 0 1 2
cpu.mask.negotiate: 0 1 2
cpu.mask.request: 0 1 2
cpu.mask.io: 0 1 2
```

3 Real-time behavior analysis

The purpose of these tests is to examine the effects of using the new real-time functionality in different usage scenarios. The three basic scenarios that will be looked at is having no load on the proxy machine, having a significant amount of CPU load from non-Dante processes, and having a significant amount IO traffic (both in other processes and passing through Dante). The behavior of Dante in these environments, when the real-time functionality is enabled, is compared to the behavior when the functionality is not enabled (the default/traditional behavior).

In addition to measuring the characteristics of traffic passing through Dante and other user-space processes, the performance when the traffic does not pass through Dante but is instead forwarded to the server via an *iptables* based NAT mechanism is also included. Because this mapping does not require traffic to pass via user space it gives an indication of what the optimal performance would be on the test machine (with only the hardware and kernel as performance limitations), and should closely match the performance of routing the traffic via a Linux-based layer 3 router (routing via the slow path).

Most tests are repeated at least three times, with the results from one of the test-runs shown, unless there are significant variations between the different test-runs.

Two types of traffic are generated in the tests. To measure latency a single-byte TCP request is sent to an *echo* server process on the server machine. The Round-Trip-Time (RTT) is calculated as the time from the the client sends a single-byte request until the reply has been received. The client then immediately sends a new request as soon as a reply has been received. This process is repeated for the test duration of 60 seconds, with information about the measured latency being logged each second. Values such as the lowest, average and median latency is stored.

To measure throughput, the client connects to a *chargen* server process on the server machine and the total amount of data that is received in the span of 60 seconds is measured. The transfer rate is also measured each second.

All tests are run in sequence but are shown in the same plot to allow for easier comparison.

3.1 Baseline (no CPU or IO-load)

To establish a baseline for comparison, the latency was examined with no other load on either of the test machines.

A single TCP connection to the *echo* server is established through the Dante server and the RTT of a single byte being transmitted to the server and back is measured. This traffic places only a minimal load on the system and should give the lowest RTT values possible for the given combination of hardware, operating system, and software.

Figure 1 shows the resulting RTT values. There is little variation in either the median or average latency and both values are essentially identical regardless of whether the real-time functionality has been enabled or not. Both the median and average RTT values are at around 500 μs. The latency for the traffic forwarded via NAT is as expected lower, at 460 μs, so the cost of sending the traffic (a single byte) via Dante appears to be around 40 μs, compared to kernel based routing/NAT.

There is some variability in the lowest RTT values measured, but this is to be expected because the traffic passes between three machines, two switches and three different user-space applications.

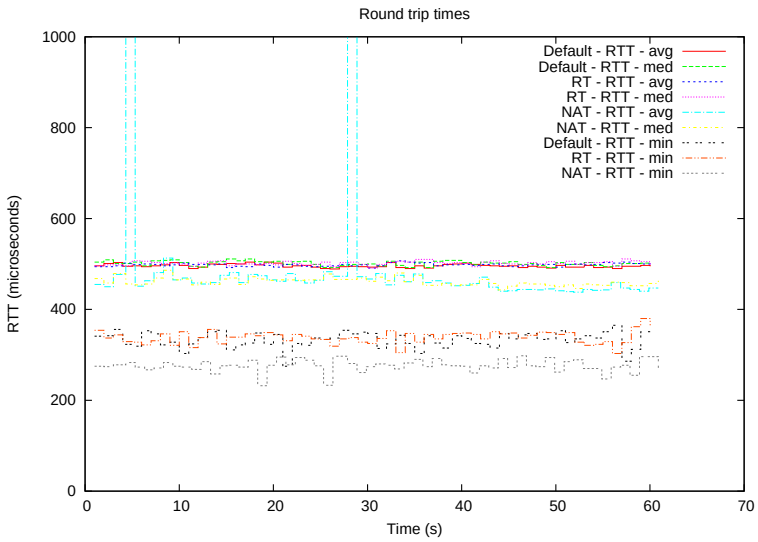

Figure 1: Baseline RTT

The baseline for the transmitted data is shown in Figure 2. In this test, 100 TCP clients are run in parallel and receive data from the *chargen* server, with 10 clients started each second from the start of the test. The sum of the transfer rates of all clients (as measured on the client side) is shown in the plot.

As with latency, there is no significant difference between using real-time scheduling or the default settings when there is no other load on the machine. The aggregated transfer rates lie at around 895 $MBit/s$. The measured rates for the traffic forwarded via NAT shows less variation than for the traffic passing through Dante, but the total rate lies at around the same value.

3.2 Effects of CPU load

To create a scenario where real-time scheduling is more likely to have an effect, a CPU based load is added to the machine running Dante. The load basically consists of a program that maintains a set of up to ten CPU intensive processes that run for roughly one second before terminating and being replaced by a new process. Each process performs a simple mathematical operation in order to use the CPU but does not use the disk or network. The number of CPU intensive processes exceeds the number of CPU cores/processors on the machine, and the processes are also given a *nice(1)* value of -3 to give them priority over the Dante processes, which run with the default *nice(1)* value of zero.

Figure 3 shows the resulting latency values. As with Figure 1, the values are measured using a single TCP connection; there is no other load on the proxy. The primary difference is that the average latency measured often diverges from the median value when the default (i.e., not real-time) scheduling is used, indicating that there are several spikes with higher latency when the default configuration is used. This is not observed

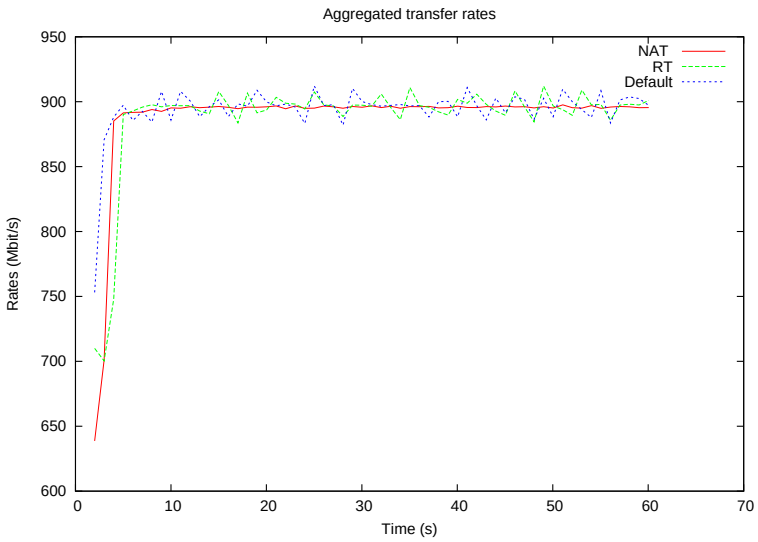

Figure 2: Baseline data transfer rates

when the real-time functionality is enabled (here the *fifo* scheduling algorithm is used). This effect was also seen with a slightly higher numbers of parallel connections.

The traffic load here is quite light, but the difference between using and not using real-time scheduling indicates that using real-time scheduling might reduce the likelihood of other processes having a negative impact on the latency of traffic passing through Dante. Using real-time scheduling appears to provide a latency that averages very close to the median, without sudden latency increases.

The effect of CPU load on accumulated transfer rates is shown in Figure 4. Here the CPU load clearly has an impact on the performance of Dante and results in much less consistent transfer rates than when real-time scheduling is used. At the lowest point the aggregated transfer rate falls to $245\ Mbit/s$, when the default scheduling algorithm is used. When real-time scheduling is used (here the *rr* algorithm), the rate is not significantly changed compared to when there is no CPU load (see Figure 2).

As expected, having less access to the CPU results in less data being transmitted. Figure 5 shows the aggregated number of bytes transmitted via Dante. Comparing the NAT forwarded traffic and the traffic that passes through Dante when real-time scheduling is used, the default scheduling algorithm gives worse performance; $5.4\ GByte$ is transmitted during the test period, compared to $6.3\ GByte$ when real-time scheduling is used.

Note that some experimentation was needed to find a way to generate a CPU load that would actually give a negative impact when real-time scheduling was not used. Unless other processes that are very CPU intensive or use real-time scheduling run on the same machine as Dante, the benefits from using real-time scheduling might be less than what is seen here.

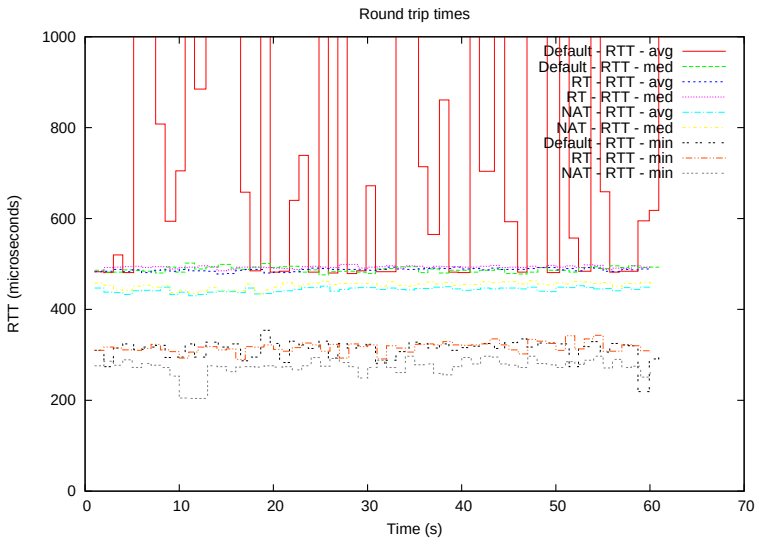

Figure 3: RTT with CPU load

3.3 Effects of external network load

This test looks at the effects of external network traffic on latency and basically creates a worst case scenario where bulk traffic saturates the link. Two Dante servers are run on the proxy machine, each with a separate type of traffic passing through it. A single TCP connection is opened via one of the Dante servers in order to measure latency as above. The other Dante server is only used to create the network load, by having 100 TCP connections opened via it to the *chargen* server. Different client machines are used for latency measurements and the bulk data transfers, and there are similarly two different machines used for the *chargen* and *echo* servers; each client connects to a different server via different Dante servers, but both Dante servers run on the same machine. The data from the *chargen* server essentially saturates the link and results in most of the CPU time being spent in the kernel or processing interrupts. The bulk traffic load is started roughly ten seconds after the latency measurements start.

Figure 6 shows the latency for this usage scenario. The point at which the network load is added is clearly visible on the graph as the latency starts rising. There is no clear reliable benefit from using real-time scheduling in this case, which is not surprising as the resource being contended is not the CPU but the network/IO-bus. Even the traffic forwarded via NAT does not show significantly lower latency than the traffic that passes through Dante; the bottleneck here appears to be hardware related.

To examine the effects that the external network traffic has on transfer rates through Dante, the external load is reduced to 50 TCP connections and an identical load of 50 TCP connections are opened via Dante. The effects can be seen in Figure 7. The total rate is halved, which is not surprising because both client machines have the same number of connections open to the same *chargen* server, splitting the available network capacity between the connections that receive data. Use of real-time scheduling does

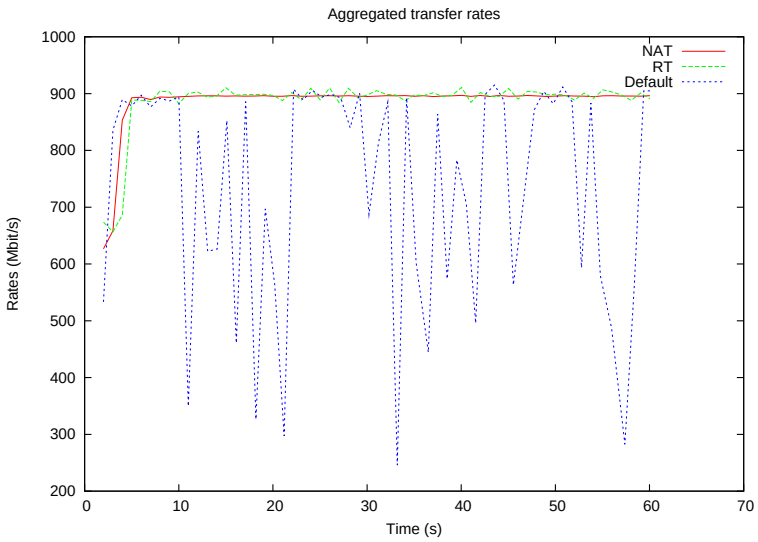

Figure 4: Data transfer rates with CPU load

not provide any difference in this case either, again most likely because the bottleneck is not the CPU.

Interestingly, the NAT traffic shows markedly degraded performance in this scenario. The reason for this is unknown, but possibly the kernel based forwarding mechanism does not perform well when there are user space processes that compete for the networking capacity, even when real-time scheduling is not used by the user space processes.

3.4 Effects on latency from internal network load

Finally we look at the effects that bulk data transfers passing through Dante can have on the latency of traffic also passing through the same Dante processes. 100 TCP connections are established to the *chargen* server to create the network load, and one TCP connection is established to the *echo* server to measure latency. As the traffic from the *chargen* server will saturate the link and only a single byte is transmitted each time the latency is measured, this represents a worst-case scenario with regards to the influence the bulk traffic can have on the measured latency.

Figure 8 shows the results for this usage scenario, and here real-time scheduling has a positive effect on latency, giving lower RTT values. The *rr* scheduling algorithm was used in the server, along with CPU affinity for the Dante *io* process (the *io* process was also bound to one of the two cores on the machine).

The baseline rate plot in Figure 2 shows that enabling real-time scheduling does not improve transfer rates, but having real-time scheduling enabled appears to have a positive effect on the measured RTT when there is also a significant bulk data transfer load.

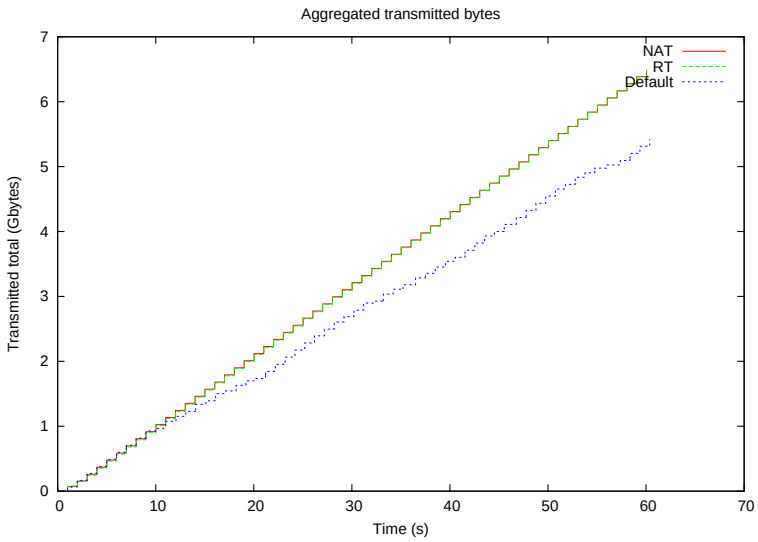

Figure 5: Aggregated data transfers with CPU load

3.5 Observed effects of bulk data transfers on RTT

The Linux machine used for testing shows what appears to be a correlation between sustained bulk data transfers and increasing RTT values. This can be seen in Figure 6 and Figure 8.

In Figure 6, the measured latency is shown to increase gradually. This increase is very gradual and does not appear to be random. It also occurs even if the TCP connection used to measure latency does not pass through Dante but is forwarded via NAT without passing via a user space application. The bulk data in this experiment is always passed through user space, while we experimented with letting the data used for calculating latency pass either through Dante or through the NAT layer when analysing the behaviour observed here.

Figure 8 shows that the same behavior is seen when real-time scheduling is not used, and that real-time scheduling has a positive effect on latency by reducing the angle of the slope. The NAT traffic does however show a different behavior. In this case the latency increases quickly as new connections are being added and then remains flat. Both the connections to the *chargen* server and the connection to the *echo* server are forwarded via NAT so no user-space applications are used in this case.

This behavior does not appear to be Dante-specific and was only observed when the link is fully utilised (transfer rates at around 900 $Mbit/s$). Most likely this behavior is the result of the machine becoming overloaded; this type of behavior was not seen on a different and slightly more powerful machine running an essentially identical Linux kernel.

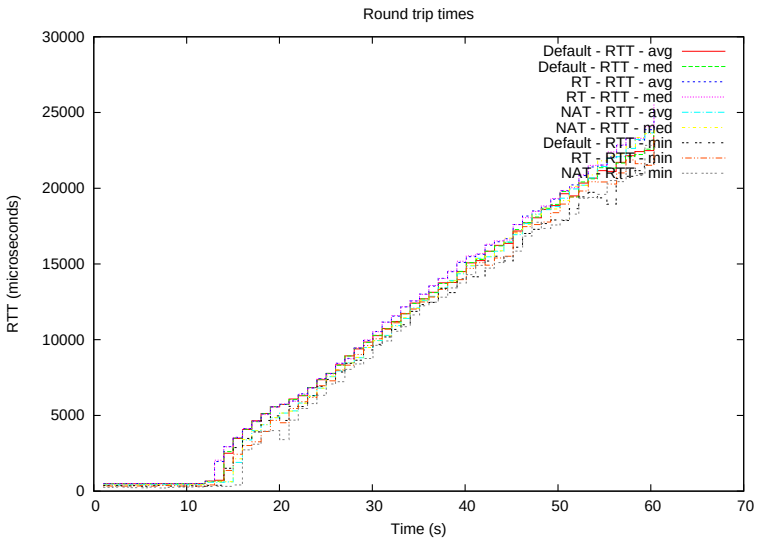

Figure 6: RTT with external traffic load

4 Configuring Dante for Real-time scheduling and usage comments

Dante uses multiple processes with dedicated tasks. The primary process types are as follows:

Mother Accepts connections from clients and forwards clients between the other three Dante process types.

Negotiate Handles SOCKS protocol negotiation.

Request Opens connections to external hosts.

IO Handles I/O.

The first three handle different phases of session setup and the last handles communication between the client and the remote server after the session setup has completed.

To influence the time required for connection setup and latency, real-time scheduling must be set for all four process types. This can be done as follows, giving a slight preference to reducing the latency of clients that have already finished establishing their sessions via the Dante server:

```
cpu.schedule.mother: rr/10
cpu.schedule.negotiate: rr/10
cpu.schedule.request: rr/10
cpu.schedule.io: rr/15
```

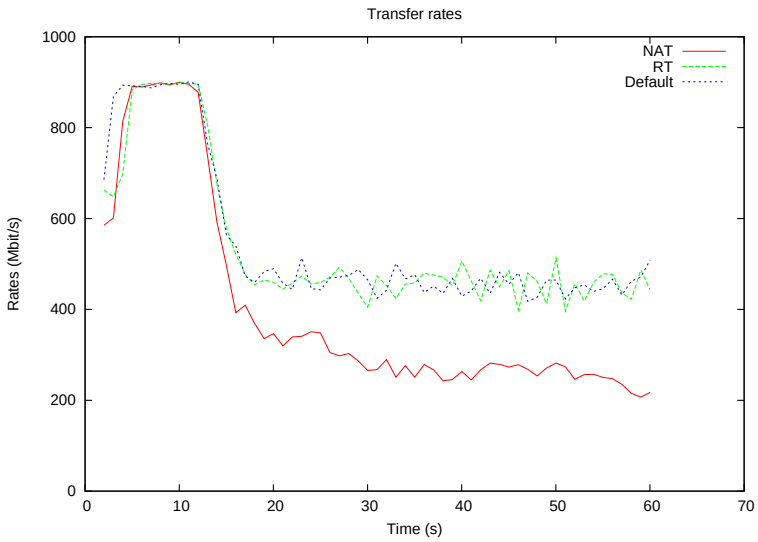

Figure 7: Data transfer rates with external traffic load

Most of the time is however spent in the IO process, which means that unless the connection setup time is important it should suffice to specify real-time scheduling for this process type, for example in this way:

```
cpu.schedule.io: rr/15
```

In general, we recommend that CPU affinity is also used when a real-time scheduling algorithm is used and that at least one CPU/CPU core is left free as a safety mechanism to prevent Dante from potentially consuming all available CPU capacity and preventing other processes on the machine from running properly.

Assuming a machine with four CPUs is used, this can be done in the following way, by specifying that only the first three CPUs should be used:

```
cpu.mask.mother: 0 1 2
cpu.mask.negotiate: 0 1 2
cpu.mask.request: 0 1 2
cpu.mask.io: 0 1 2
```

Or alternatively, if only the IO processes use real-time scheduling:

```
cpu.mask.io: 0 1 2
```

5 Summary

This technical report has examined the performance of Dante when real-time scheduling and CPU affinity is used in the Dante server.

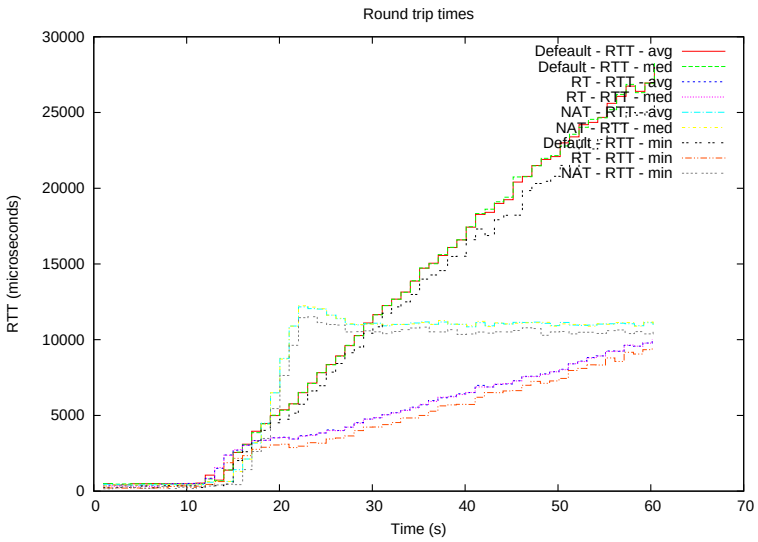

Figure 8: RTT with internal traffic load

Using real-time scheduling can improve performance, both with regards to lower latency and higher throughput, but the benefits depend on the traffic and system load. We recommend doing comparison tests to examine whether there are benefits in a given usage scenario.

Feedback to this paper can be sent to *misc-feedback@inet.no*.