Inferno Nettverk A/S

Technical Report

Performance analysis of Dante version 1.3.1

August 2, 2011

Contents

List of Figures

1 Introduction

This report documents the results of a performance analysis performed on version 1.3.1 of the Dante SOCKS server implementation from Inferno Nettverk A/S.

The load analysis was done over a period of roughly 50 hours, on two relatively highly loaded machines. One machine was serving up to 6,000 data-intensive SOCKS clients concurrently, while the other was serving up to 7,000.

The majority of the SOCKS sessions on these machine were using TCP, with less than 2% of the sessions at any time observed using UDP.

The ability of the Dante server to handle this load is examined in detail in the report, which looks at the resources consumed by Dante while handling the load, and the performance obtained.

2 Machine Description

The two machines evaluated in this document are referred to as machine *M* and machine *C*. Both machines run a standard 64-bit Linux distribution using kernel version 2.6.18, on the following hardware:

M One 2-core Intel Xeon CPU at 2GHz, with 3GB of RAM.

C Two 4-core Intel Xeon CPU at 2.53GHz, with 8GB of RAM.

Both machines have a single 100 Megabit network interface, meaning traffic between the SOCKS clients and the SOCKS server, and traffic between the SOCKS server and the remote hosts (Internet) go in and out on the same interface.

3 Dante Configuration

A non-released snapshot of the Dante source code was used for this testing, with code essentially identical to the subsequent official 1.3.1 Dante release.

Some compile time adjustments were made to the Dante server, using the following configure options:

–without-gssapi GSSAPI support was not needed on this machine and using this option removes GSSAPI support, which reduces the amount of memory used by Dante.

–with-iomax=32 This option increases the number of clients per I/O process from the default of 8 to 32.

The Dante I/O processes handle all traffic between a SOCKS client and its target after the initial SOCKS request has successfully completed.

Increasing the number of clients each I/O process can handle reduces the number of Dante I/O processes required, which further reduces the amount of memory used by Dante.

These adjustments were done in order to reduce the amount of memory used by Dante, as previous observations had indicated available RAM on at least the machine 'M' was not sufficient at times with high client loads.

Note that increasing the number of clients per I/O process may make it necessary to increase the permitted kernel/system socket buffer sizes also, due to larger amounts of control data being sent between Dante processes.

On the machines in question, these increases had already been done for other reasons, but if not, changes similar to the below may need to be done (if the existing settings are too small, the Dante server will complain and refuse to start):

Using the *sysctl* program, apply the following changes:

- *net.core.rmem_max = 16777216*
- *net.core.wmem_max = 16777216*

Note that the above settings are for Linux. Other platforms might have other ways of increasing these values.

Apart from these modifications, no special changes were made to either Dante or the machines, and Dante was started with the equivalent of these standard options: `sockd -Df /etc/sockd.conf`.

Only errors were logged by either Dante server, meaning no data was written to the log files during normal usage.

4 Performance and Load Analysis

Information about the system was obtained using standard system tools such as *ifconfig(8)*, *top(1)*, and *ps(1)*, sampled at 15 second intervals.

This approach places relatively little extra load on the system due to the monitoring, but the values obtained are not always entirely exact. For example, having the Dante server log data traffic would allow exact values for transmitted data to be obtained, but at the cost of increased system load due to the overhead from logging. Obtaining these values instead from *ifconfig(8)* means that traffic not going to or from the Dante server might inflate the number of transmitted bytes, so this approach is only usable if the vast majority of traffic on an interface is to or from the Dante server. For these machines, it was verified manually that there appeared to be very little traffic other than the traffic going to or from the Dante server, making the data obtained via *ifconfig(8)* be fairly accurate. The bandwidth measurements that are listed in the results with a ”per second” rate are based on averaging the bandwidth used during each sample period of 15 seconds, which should be unproblematic due to the measurements spanning more than two days.

The CPU values measured are likely to be slightly lower than the actual load; only CPU usage by each process since last sample point is calculated. This means that the CPU time spent by processes that terminate will only be measured up to the last sampling point before the process exits. Likewise, processes with a lifetime of less than the sampling rate may in some cases be ignored completely, if they both come alive and exit between two sampling points. Most CPU intensive processes were however long-running on these machines, so this should not have significant consequences for the measured results, except for probably giving slightly lower CPU usage values.

Some of the Dante processes running on the two machines belonged to an older instances of Dante; the I/O processes will typically be kept running until all connected clients have finished, in order to avoid existing client connections being suddenly aborted if the Dante server is restarted. On machine *M*, 35 processes like this were running at the beginning of the measurements, while 97 processes were running

at machine *C*. Most of these processes had a small number of clients (no more than three clients) and terminated during the first few hours that the measurements were performed. Since these old Dante processes only have a few clients and are unable to accept new ones, they result in a somewhat higher resource usage on the machines than what would otherwise be the case. This is commented in the results where relevant.

4.1 Analysis Results

Being a network proxy, the actual network network performance is quite important. We first look at the total amount of data being transmitted and the number of simultaneous clients handled. Next we examine the resources that are consumed on each machine while handling this load. Finally, we look at how well the Dante version examined uses the processes it creates.

4.2 Network Performance

Figure 1 shows the transfer rates for the two machines. The majority of the data comes from the external network and goes to the SOCKS clients. The highest recorded transfer rate for machine *M* is 86 Mbit/s and occurs around four hours after the measurements were started. For machine *C*, the highest rate is 92 Mbit/s and occurs after roughly 45 hours.

Considering that 100 Mbit/s is the link limit, these values are quite high and indicate that Dante is able to provide good traffic utilization on both machines.

Another factor that can be observed from Figure 2 is that the clients receive considerably more data than they send. Machine *M* on average receives four times as much as it sends, while machine *C* receives 5 times as much as it sends. This difference is somewhat unexpected, as the type of clients served by both machines should be similar.

One can also observe from the same figure that the ratio fluctuates in a fairly regular pattern; the clients receive more and more data (relative to the amount they send), until a top is reached and the the clients start receiving less and less, until a bottom is reached, etc. Most of the reason for these fluctuations seems to be, as can be observed from Figure 1, that while the rate the clients sends data at remains fairly constant, there are significant variations in the rate that they receive traffic at.

An overview of the number of active, mostly TCP-based, clients are shown in Figure 3. The highest total number of clients is 6734 for machine *M* and 5850 for machine *C*.

On both machines, the majority of clients are in the *I/O* phase, meaning that both SOCKS protocol negotiation (handled by *Negotiate* processes) and SOCKS request processing (handled by *Request* processes) has completed, and the clients are now either ready to do I/O, or waiting for the new connection attempt to the target address to complete.

This will typically be the case when the majority of sessions are long-lived, by which we mean that the sessions lasts considerably longer than the time taken to finish the negotiate and request phases.

Both the transfer rates and the number of clients change over time, and as can be seen in Figure 4, there is a certain correlation between an increased number of clients and higher total transfer rates and increased network utilization.

This relationship is most easily seen for machine *M*, which sees the highest total transfer rates when the number of clients increases. Machine *C* sees higher total trans-

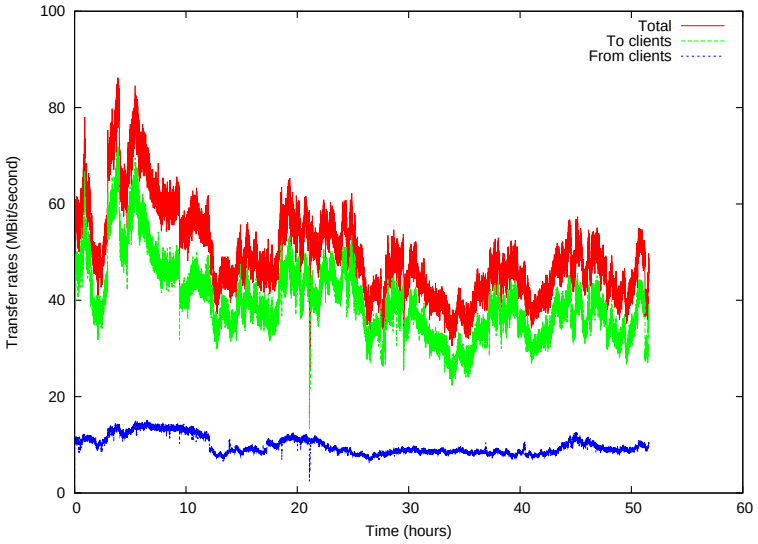
fer rates with fewer clients, and the relationship between the number of clients and the transfer rates seems less certain here.

The reasons for the higher transfer rates observed on machine *C* are unknown, but possibly due to external factors, such as better better Internet connectivity, or machine *C* having more powerful hardware than machine *M*. If the higher transfer rates are due to the latter, it could be expected that improving the hardware configuration of machine *M* would lead to improved performance for the SOCKS clients using the machine.

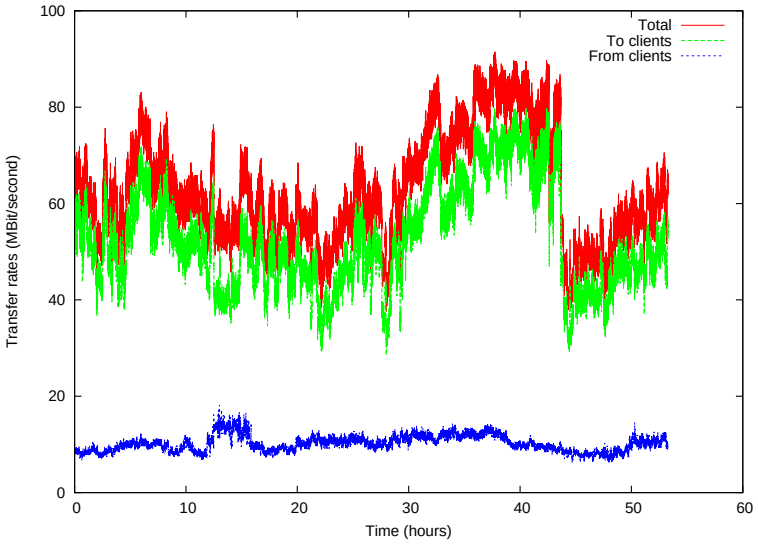
For both machines, the relatively high number of clients and high transfer rates result in significant amounts of data being transmitted during the measurement period.

Figure 5 shows the total amount of data transferred by both machines; $1066\,GBytes$ for machine *M* and $1397\,GB$ for machine *C*. Machine *M* receives 854 GB during the measurement period, and sends 212 GB for an average traffic rate of 47 Mbit/s.

The corresponding values for machine *C* are 1170 GBytes received, 227 GBytes sent, and 60 Mbit/s as the average traffic rate.

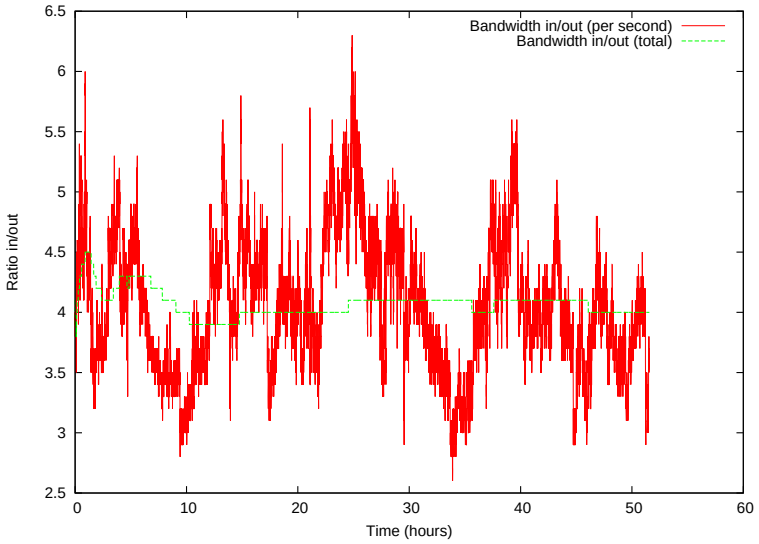


(a) Machine *M*

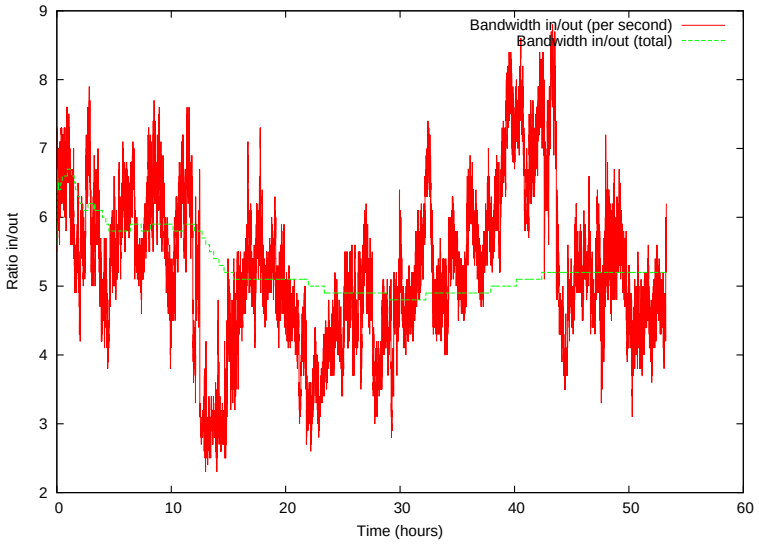


(b) Machine *C*

Figure 1: Transfer rates

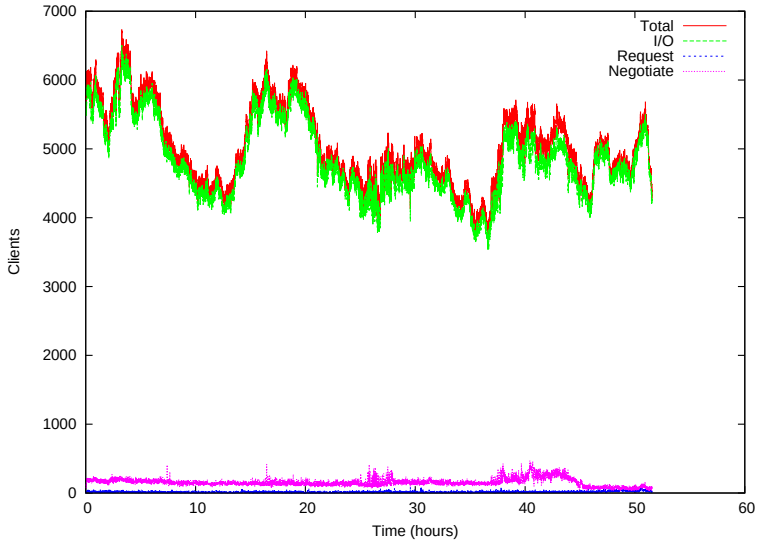


(a) Machine *M*

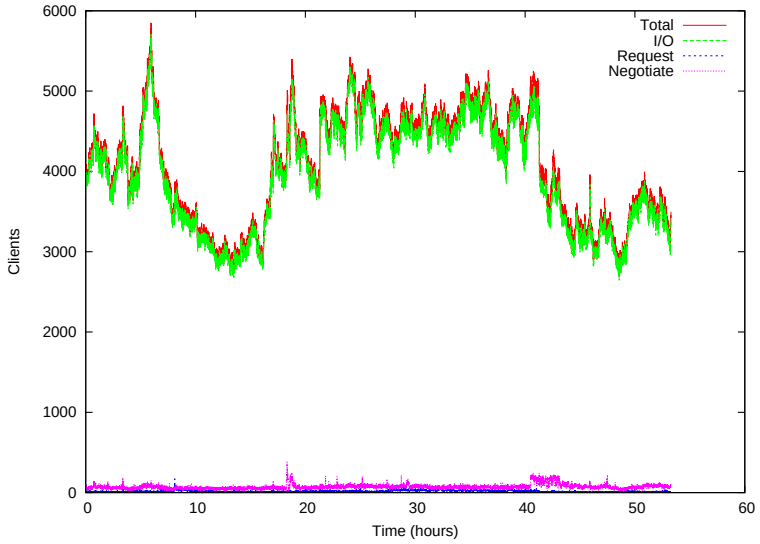


(b) Machine *C*

Figure 2: Traffic ratio, in to out



(a) Machine *M*



(b) Machine *C*

Figure 3: Active clients

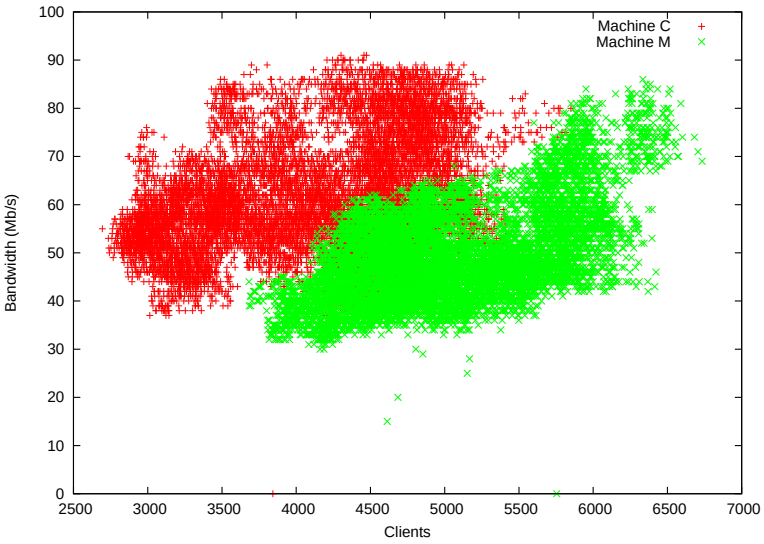
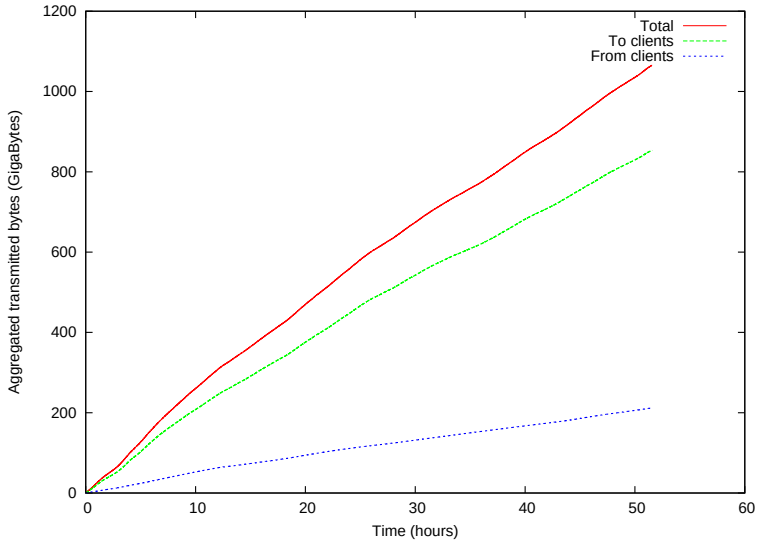
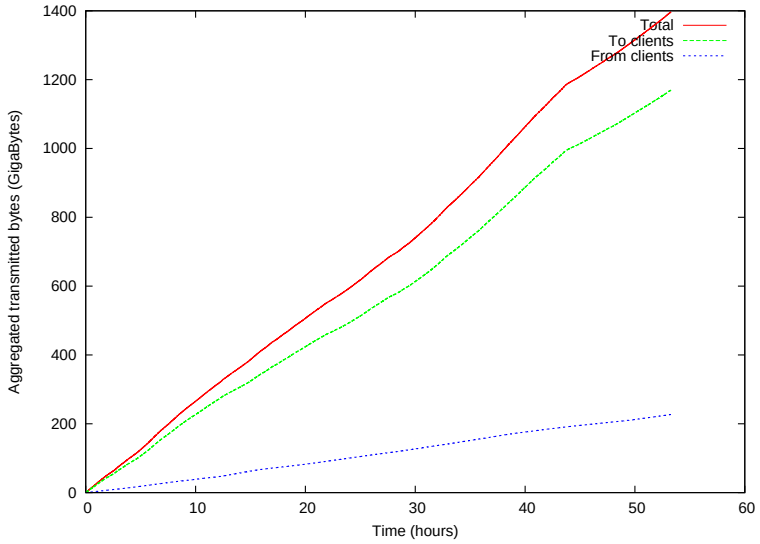


Figure 4: Number of clients and bandwidth usage relationship



(a) Machine *M*



(b) Machine *C*

Figure 5: Aggregated data transfers

4.3 System Load

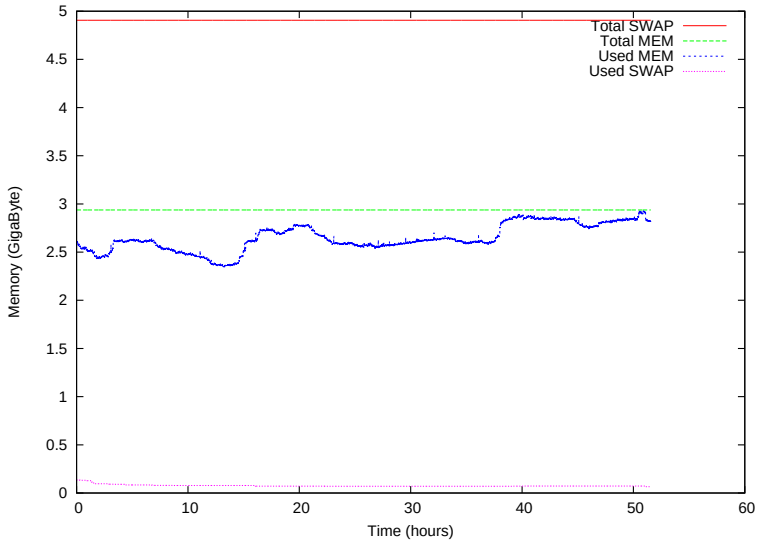
The memory and swap usage on the machine, as reported by *top(1)*, is shown in Figure 6.

It should be noted that these values should be interpreted with a certain amount of caution. Especially the amount of memory reported by *top(1)* as used can include memory used for caching by the kernel; the actual amount of memory available for application use might actually be higher.

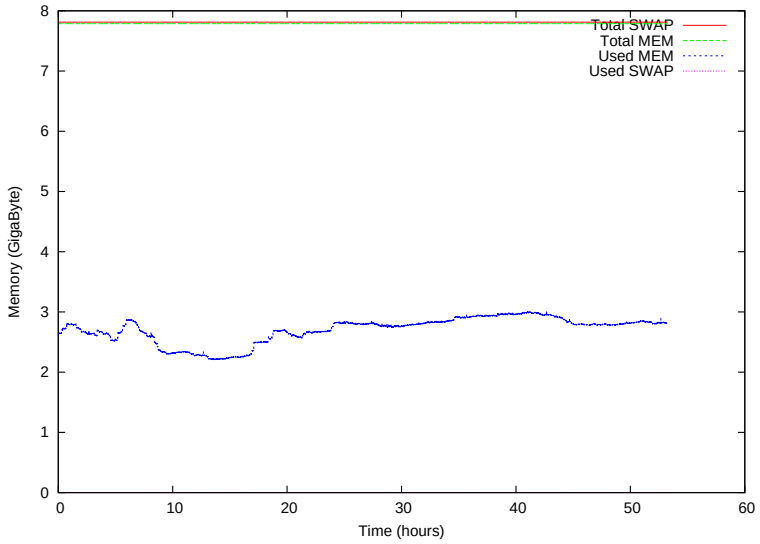
The figure does however show some differences between the two machines that corresponds to our expectations; memory usage is roughly similar, at between two and three gigabytes of memory. For machine *M*, only three gigabytes of memory is available on the machine, and it shows a small amount of swap space being used. Machine *C* has eight gigabytes of memory available and does as a result never use more than half of the available memory and swap space is never used.

Memory usage is one area where the Dante processes left over from a previously run Dante server likely resulted in increased memory consumption. Most of these process only have a small number of clients but still consume memory. The used memory values in the plot are however fairly constant, while most of the left over processes disappear during the first hours, so most likely the effect is not that significant.

An overview of CPU usage is shown in Figure 7, with Dante as the primary source of CPU usage; around 40% of CPU usage comes from Dante on machine *M*. Machine *C* is significantly more powerful and shows only around 10% CPU usage from Dante. One other CPU intensive process (*Process I*) runs on both machines, having relatively constant CPU usage at roughly half the CPU usage of Dante. Both machines are able to handle the load they have, but there is not much free capacity on machine *M*.

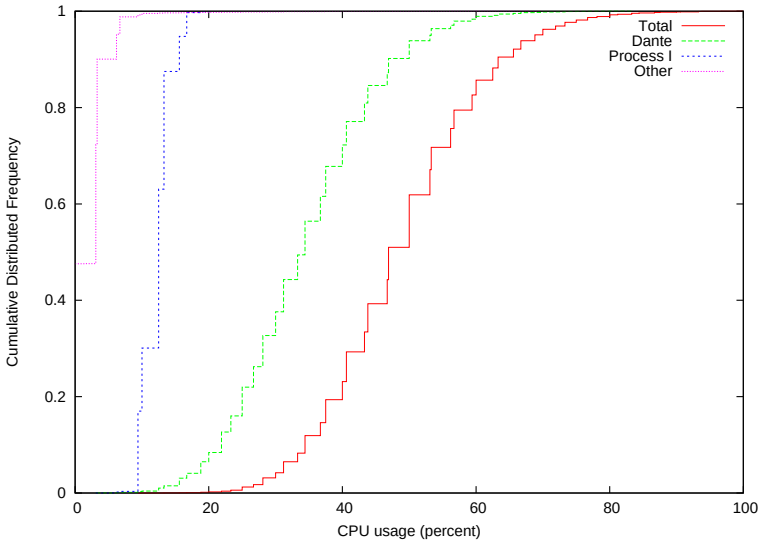


(a) Machine *M*

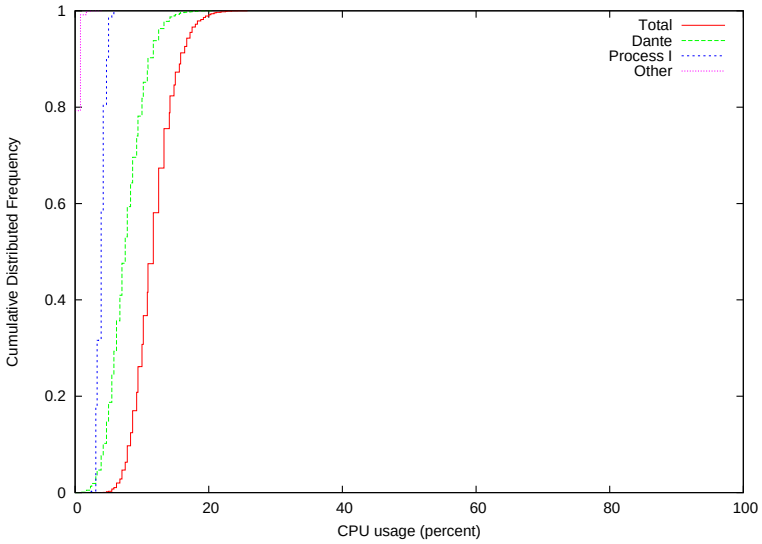


(b) Machine *C*

Figure 6: System memory usage



(a) Machine *M*



(b) Machine *C*

Figure 7: System CPU usage

4.4 Dante Behavior

As seen in Figure 8, the Dante I/O processes are most CPU intensive, as they handle the task of moving data between two connected machines. For machine *M*, the I/O processes uses between 20 and 60 percent of the available CPU time, while around 10 percent is used on machine *C*. The other Dante use very little CPU.

The I/O processes also make up the majority of the Dante processes that are running at any time, as shown by Figure 9. Some of the processes shown are left over from an older Dante process.

On both machines, the total number of I/O processes varies between 150 and 250 most of the time, while the other process types rarely exceed 50 in number. The highest variance comes from the request processes, where the number of processes more directly corresponds to the load at any given time.

This is due to a larger amount of capacity being kept available in the I/O processes. Figure 10 shows the available client capacity in the different process types, which corresponds to how many new clients Dante is able to handle at any given point, without creating any new processes. The capacity does not include the free slots at the left over processes because these cannot be used by the currently running Dante server.

The figure shows that I/O processes at many points have far in excess of 1000 free slots. The client capacity for I/O processes is also highly variable, while the number of I/O processes is fairly stable, as shown in Figure 9.

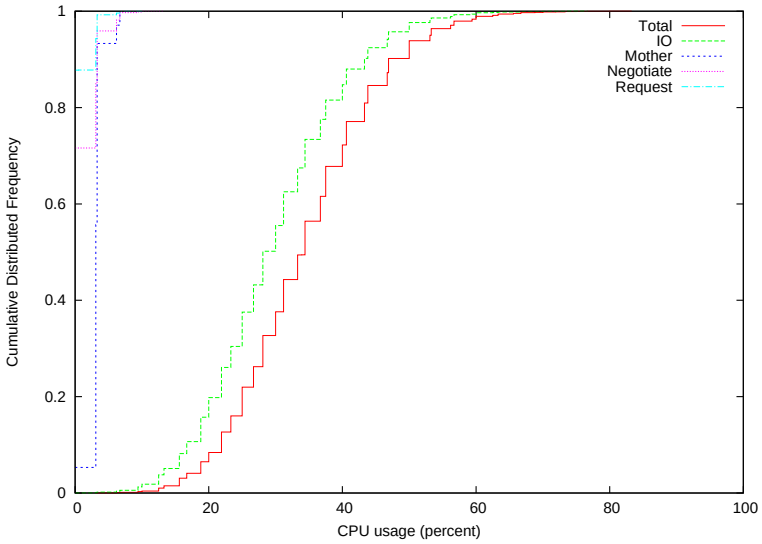
Most likely, the current version of Dante maintains a higher number of free slots than is necessary. Reducing this number by making more efficient use of the available processes would likely make it possible to reduce the number of running I/O processes, and thereby reducing the resource consumption of Dante.

5 Summary

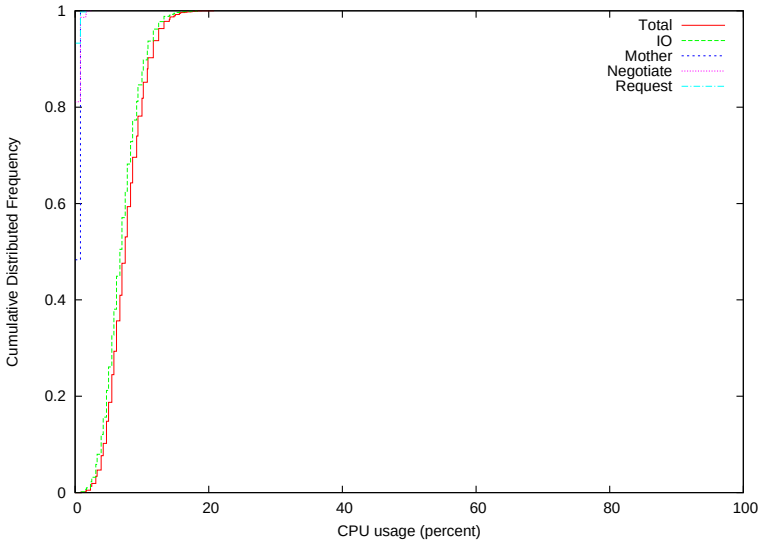
This technical report has examined the performance of Dante on two machines where the Dante server is placed under a relatively high load with up to 7,000 concurrent data-intensive clients. The results show that Dante is able to handle this load without problems on these two machines, but the less powerful machine must be said to be relatively highly loaded.

The results also show some areas where Dante can be improved; making more efficient use of existing processes and reducing the number of free client slots available at any time will likely reduce the resource requirements of Dante.

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

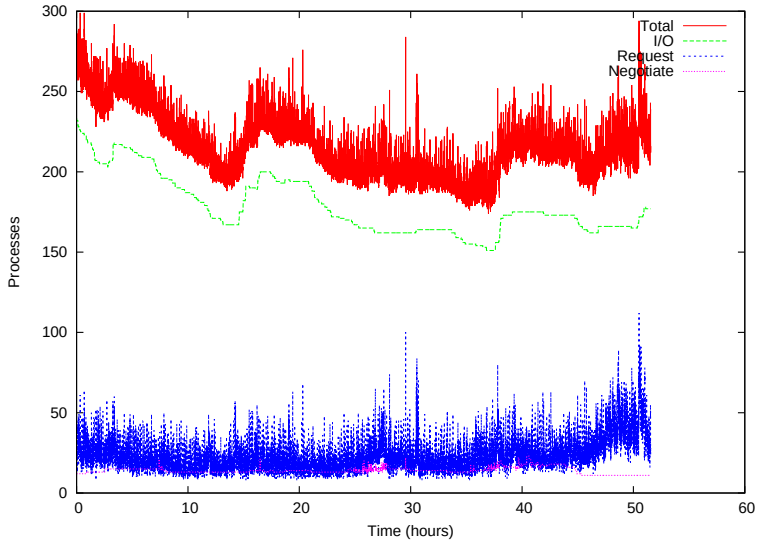


(a) Machine *M*

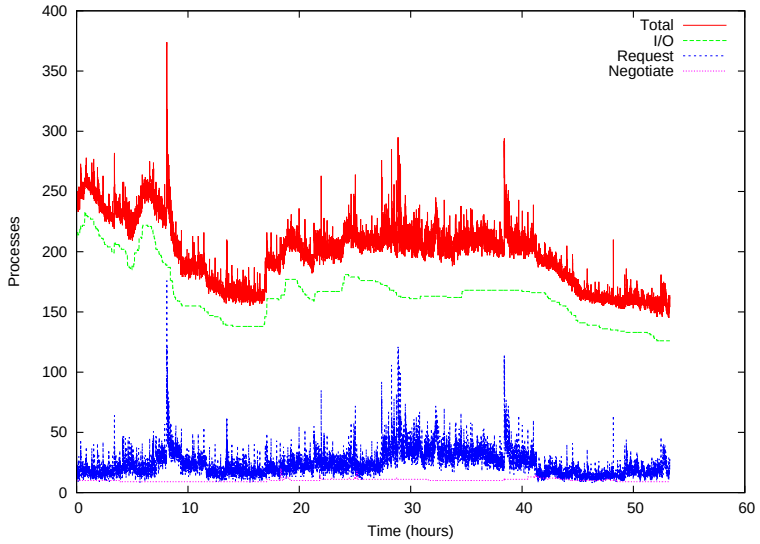


(b) Machine *C*

Figure 8: Dante process CPU usage

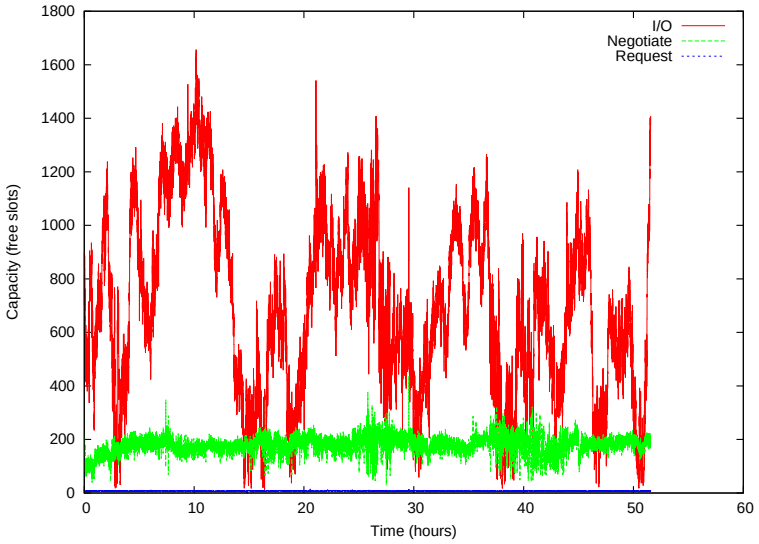


(a) Machine *M*

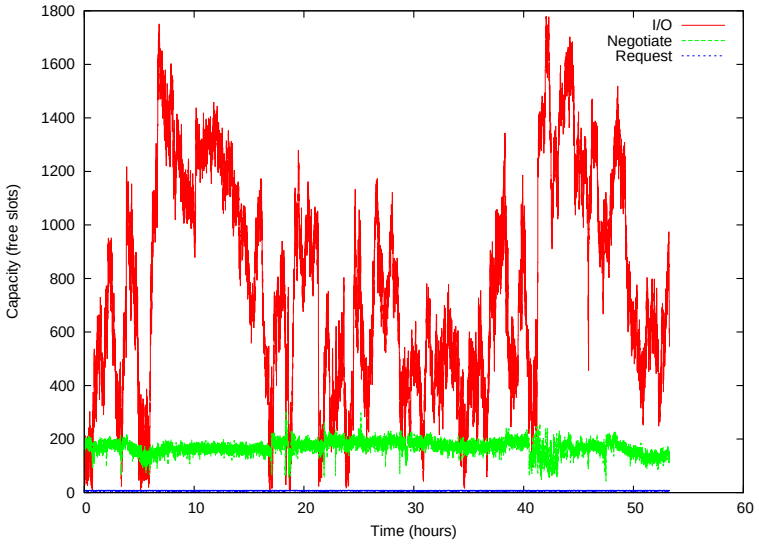


(b) Machine *C*

Figure 9: Dante processes



(a) Machine *M*



(b) Machine *C*

Figure 10: Free client capacity