Benchmarking the RTI for Use in a Simulated Radio Environment

David Nemeth
Advanced Simulation Technology, Inc.
441-A Carlisle Drive
Herndon, VA 20170
703-471-2104
davidn@asti-usa.com

Keywords:
Audio, Radio Environment, RTI performance, Back Channels

ABSTRACT: *The RTI provides a general purpose network communications mechanism. This paper examines how well suited it is to the particular problem of a simulated radio environment. This environment must pass both low bandwidth transmitter/receiver information and high bandwidth audio information. Low latency, efficiency, and scaleability are all requirements that the RTI must provide.*

The ability of the RTI to meet these needs is discussed. Results on measurements for latencies, CPU usage, and bandwidth overhead are presented and analyzed in light of the requirements of a simulated radio environment. An analysis of whether back channels are required for the audio is given.

Measurements were performed on the DMSO RTI 1.3v4, and the MAK RTI 1.3, running on a 233 MHz AMD K6 machine with the Red Hat Linux 5.1 operating system. Comparisons are made to the current state of the art for DIS radios.

1. Requirements for a radio environment

The requirements for the radio environment are simple: to fool the operator into thinking they are using a real radio, and not a simulated one. Practically, this places two requirements on the transport mechanism.

The first requirement is low latency - the voice must be transferred with a minimum of delay. In the real world, radio signals travel with no perceptible delay. In a simulation, experience has shown that a latency of less than 100 ms is sufficient for training purposes. Latency much greater than 150-200 ms becomes noticeable to two users carrying on a conversation. This delay comes both from the latency in the transport layer, as well as the latency inherent in buffering up the audio. If the transport layer latency has any jitter in it, the worst latency and not the average latency will become the latency of the voice stream.

The second requirement is for a large number of audio streams to be carried. In the real world, there is no limit to the number of radio conversations that can be carried out at once. In a simulation, the number of audio streams that can be carried will limit the number of players in the simulation. Large team training simulators presently fielded can have well over a hundred operators, and hence a hundred voice streams at once are possible. While one machine might not be required to handle this volume of traffic, the architecture should be scaleable enough to handle it. Hence, a large background traffic should not effect the latency or performance of any machines. A cost efficient architecture (not just one or two operators per computer) requires the individual machine to process a large number of voice streams. Also, each operator on a given machine often needs the ability to receive more than one voice stream (from different virtual radios or from network intercom buses), so the number of voice streams is often a multiple of the number of operators on a machine.

2. Test procedures

All tests were run on an AMD 233 processor

running Red Hat Linux 5.1. The RTIs tested were the DMSO 1.3v4 and the MAK 1.3. All data was sent Best Effort.

Three tests were carried out on each RTI. In the first, attribute updates and interactions were sent out with various numbers of parameters/attributes of varying sizes. The packets produced were examined using the tcpdump utility on Linux. The sizes of the packets as a function of the various parameters were examined, and found to follow a simple formula.

The second test involved a sending and a receiving federate. Again, the sending federate sent out attribute updates or interactions with varying sizes and numbers of attributes or parameters. The receiving federate was running by itself on a separate machine. It called tick() at a 1 Hz rate, and the time in tick was measured.

The receiving processing load was the one measured, because there is a natural imbalance in the flow of audio data. An operator can have at most one voice stream going out, but is often listening to several incoming audio streams at the same time (which may be intercoms, radios, etc.). This makes the receiving efficiency more critical than the sending efficiency.

Care was taken so that only the time spent by tick() in receiving the messages was measured. gettimeofday() was called immediately before and after tick(). There were no print statements during any of the callbacks. There was no keyboard or mouse activity, changes in the graphics, screen savers turning on or off, etc. The callbacks in the federate ambassador did nothing with the data, except increment a counter to ensure the proper number of packets were received.
Furthermore, after tick() was called, the process paused by making a blocking read on the real time clock. Hence, the operating system was in control when the incoming UDP packets were received, allowing them to be processed to the point of being read by the RTI.

Because of these precautions, the resulting time measured was only the amount of time required for the RTI to handle its part of the transport process - the time for the network layers to process the incoming data, and the time for the federate to act on the data are NOT included. In any real application, of course, the received data would be acted upon in some way.

The tick() function was configured/called in such a way that it would process all incoming packets in a single call. The mechanism for doing this is different between the MAK and the DMSO RTI implementations.

The third test involved sending audio over the RTI and measuring the latency. Latency was measured by placing a microphone next to the sender's microphone and another microphone next to the receivers speaker. Clicks were generated and sent, and the two microphone traces were displayed on an oscilloscope. These measurements were performed on ASTi's HLA Coordinator™, a voice over HLA product.

3. Results

3.1 Results of packet size test

The size of the data portion of the UDP packets (not counting the UDP, IP, and ethernet headers) that the DMSO RTI sent out depended on whether bundling was off or on. With bundling on, the UDP packets each contained two attribute updates or two interactions. With bundling off, the UDP packets only contained one attribute update or one interaction. Bundling was controlled by changing the RTI.rid file for the DMSO RTI. The MAK RTI does not appear to bundle data in this way.

The packet sizes followed the following rules for the DMSO RTI:

For attribute updates, the packet size was found to be:
1 update per UDP packet:
packet size in bytes = 176 [RTI overhead] + (number of attributes)*4 + (total attribute data size)
2 updates per UDP packet:
packet size = 344 [RTI overhead]+ (number of attributes)*4 + (total attribute data size)

For interactions, the results were:
1 interaction per UDP packet
packet size in bytes = 92 [RTI overhead] + (number of parameters)*4 + (total parameter data

size)
2 interactions per UDP packet
packet size in bytes = 176 [RTI overhead] + (number of parameters)*4 + (total parameter data size)

For example, for an interaction with 3 parameters with 10, 30, and 40 bytes of data, the size of the data portion of the UDP packet would be:
92 [RTI overhead] + 4*3 [parameter overhead] + 10 + 30 + 40 [data size] = 184 bytes.

The results for the MAK RTI are as follows:
For attribute updates, the packet size was found to be:
1 update per UDP packet:
packet size in bytes = 26 [RTI overhead] + (number of attributes)*4 + (total attribute data size)

For interactions, the results were:
1 interaction per UDP packet
packet size in bytes = 14 [RTI overhead] + (number of parameters)*4 + (total parameter data size)

These formulas break down when the total packet size exceeds the maximum udp packet size of 1500 bytes. The DMSO RTI handles this by breaking down the data structure into two udp packets, while the MAK RTI counts on the IP layer to fragment the data packets.

For the DMSO RTI, we see that interactions carry a 92 byte packet overhead, plus 4 bytes for every parameter the data is broken into. For attribute updates, the overhead is even larger: 176 bytes. This doesn't include the IP, UDP, and ethernet headers.

For the MAK RTI, the packet overheads were much smaller: 26 bytes for attribute updates, and 14 bytes for interactions.

Table 1. Packet overheads for the DMSO and MAK RTIs

	Interaction packet overhead	Attribute Update packet overhead
DMSO RTI	92 bytes	176 bytes
MAK RTI	14 bytes	26 bytes

3.2 Results of tick timing test

The timing tests required a more careful analysis than the packet size test, because there was significant variation in the tick times in a given run for receiving the same amount of data. All tests were performed with bundling off.

The tick timing test measures the amount of time the RTI takes to receive data and pass it to the federate through a federate ambassador callback. As mentioned above, care was taken to measure only the time in took the RTI to get the data from the network layers and pass it back to the federate ambassador. It's important to remember what is NOT included in this time: the time the OS spends handling the network layers, and the time the federate spends actually doing things with this data. Hence, these results represent an upper bound to how much data the RTI can handle - the actual amount is going to be much less.

We would expect the time spent in tick to follow the following formula:
total time = t1 + Np*(t2 + t3*Na + t4*Nd)
where
Np = number of packets received in a tick()
Na = number of attributes/parameters per packet
Nd = total amount of data per packet (the sum of the data from ALL the attributes/parameters)

t1,t2,t3, and t4 are times which are fit to the data, which can be interpreted as follows:

t1 = time spent in tick doing things other than data handling. The RTI, in a normal simulation, will be doing many other things: providing DDM services, discovering and publishing objects, etc. These things would all fall under t1. In this experiment, the federate used none of these services, hence this is a nominal value.

t2 = the overhead time spent dealing with a packet of data containing an attribute update or interaction.

t3 = the overhead time spent dealing with each attribute or parameter within one packet

t4 = the time spent handling each byte of actual data

Table 2 shows the values for t1,t2,t3 and t4 for interactions and attribute updates for the DMSO RTI. Table 3 shows similar results for the MAK RTI. Figures 1 and 2 show graphs of the predicted versus actual values for several points (note the difference in scales on the y axes for the two graphs.) The formula predict the average time to within 5%. Note, however, that the time measured deviates from the average by as much as a millisecond in either direction - the exact cause of this is unknown.

Table 2. Parameters for formulae for the DMSO RTI. All values in ms.

	t1	t2	t3	t4
Attribute Updates	1.7	0.52	0.011	3.1×10^{-5}
Interactions	1.8	0.38	0.009	5.8×10^{-5}

Table 3. Parameters for formulae for the MAK RTI. All values in ms.

	t1	t2	t3	t4
Attribute Updates	0.21	0.08	0.0065	5.1×10^{-5}
Interactions	0.15	0.07	0.007	5.3×10^{-5}

In both the MAK and DMSO RTIs, tick() would only process around 36 incoming packets. The remaining packets were lost. 36 was not a hard number - sometimes it would be more, and sometimes less. 32 packets per tick seemed to pose no problems. Because this was true for both the MAK and the DMSO RTI, the problem probably lies in the network layer configuration of the system, and had nothing to do with the RTIs themselves. It would be expected that reconfiguring the systems network layers (which is certainly possible with Linux) would solve this problem. For the purposes of this paper we will assume this is not a fundamental limitation.

The time spent processing by the network layers can be measured by sending a large number of interactions over the network and watch the CPU usage with the tool xosview. For interactions with 256 bytes of data, the network layers spend about 0.04 ms per packet. This is small on the scale of the DMSO RTI cpu usage, but significant for the MAK RTI.

3.3 Results of the Latency test

The latency is the amount of delay in an audio stream sent from one machine to another over the RTI. As noted above, latencies should be kept to 100 ms or below to prevent them from being too noticeable to the operator. Audio was sent out as interactions, each containing 1/32 seconds of audio data. Audio was sent out best effort. For both the MAK and DMSO RTIs, the latency was unaffected by adding additional audio streams, up to a limit of about 35. At this point, the audio packets started to get dropped, resulting in voice breakup. (It is fair to assume, as was mentioned above, that the losing of packets was due to the network layer, and was not inherent to either RTIs.)

Latency has two components - the latency of the transport mechanism, and the latency inherent in the audio buffering scheme. By way of comparison, a currently available DIS radio environment[1] has an 80 ms latency end to end on a LAN, and sends out audio packets containing about 30 ms of audio data. This latency includes a built in delay of 20 ms, which is effected by buffering up audio in the receiver to provide robustness in the face of network latency jitter.

It should be noted that the Red Hat Linux Operating system is not real time. Because of the non-real time nature of the operating system, it made a difference what the process priorities were set to. Running at normal priority ran with a 10 ms greater latency for both RTIs than running at a boosted priority level. The non real time nature becomes more problematic if there are disk accesses or other time consuming processes going on at the same time. When sending audio over networks, care must be taken to minimize the effect of the non-real time nature of an operating system.

[1] The radio environment referred to is the one used in the ASTi Digital Audio Communications System (DACS). The author knows of no other DIS radio implementations with this level of performance, but he may be biased.

To further reduce the latency, the receiving federate was called at 64 Hz, twice the rate of the sending federate. Running the receiver at twice the rate of the transmitter reduces the latency with little additional processing overhead, because the same number of packets are being received.

At a boosted priority level with the sending federate sending audio at 32 Hz and the receiving federate running at 64 Hz, the latency in the audio was 65 ms for the MAK RTI and 70 ms for the DMSO RTI. As with the DIS system, some extra buffering on the receive side would be desirable to reduce the sensitivity to jitter in the latency. Table 4 reflects what the latency would be with this added guard:

Table 3 Latencies under optimum conditions, 31 ms audio packets

DMSO RTI	70 ms + 20 ms buffering = 90 ms.
MAK RTI	65 ms + 20 ms buffering = 85 ms
Current DIS, with guard buffering	= 80 ms

We see that the latency is acceptable for both RTIs, as long as the audio transport is done correctly. Without proper timing and priority management it can rapidly degrade.

4. Discussion

Two RTIs - the MAK RTI and the DMSO RTI - have been benchmarked for use with a real time radio environment. An accurate analysis of the results cannot be given with out some further observations about the two RTIs.

The MAK RTI does not currently implement all of the services required by the HLA specification. These include time management, reliable transport, data distribution management, and others. Some of these, including time management and reliable transport, are of little or no use in a real time radio environment. Some of them, particularly data distribution management, will play a crucial role in scaling systems up to accommodate a large number of voice streams. Because it lacks these services, it is not really a full-up RTI yet.

The DMSO RTI performs all of these services, but as we will discuss, it pays a price for doing so. The price is paid both in network bandwidth because of the large packet headers, and in the CPU usage, as reflected in the tick() timing measurements.

This paper also does not address the question of how well the DMSO data distribution services work, which is crucial to the question of scaleability. It is important that the unwanted audio streams be effectively filtered out in hardware, not software. How well the DMSO RTI can assign the multicast address based on the "routing space" API was not examined in this paper. Of course, the MAK RTI does not provide the services at all at this point.

With these caveats in mind, we can proceed with the discussion.
Keeping packet overhead low is important in the transport of audio data over computer networks. When the packet overhead is comparable in size to the audio data, the scaleability of the system is directly affected. This can be a problem in situations where bandwidth is limited (WAN's and T1 lines, for example) and when a large number of voice streams are put onto the system.

Voice compression is of limited use in this situation, because it does not compress the header. For 31 ms packets, the audio data is 248 bytes long (mulaw) and the header for the DMSO RTI is 92 bytes. Even if you were to achieve 10:1 compression over mulaw, the final packet would be 117 bytes, giving you less than 3 : 1 compression overall. Sophisticated compression schemes also take a good deal of processing time, and have varying voice quality and varying sensitivity to background noise. So small packet overheads are important to audio communications over networks, and large packet overheads make voice compression much less effective in reducing bandwidth.

The packet overhead on the DMSO RTI packets was quite large, and would have a significant effect on the bandwidth usage of the system. Furthermore, the CPU usage places a fundamental limit of around 70 voice streams. At 70 voice streams, the RTI would only have time to pull the packets from the sockets, and not leave the federate any time to do anything. Of course, the federate in a radio environment has a lot of other things to do, so the number of voice streams actually handled would be much less than 70.

The MAK RTI could handle a larger number of voice streams, with a theoretical maximum of 200 to 300 voice streams. Again, this is limit where the RTI takes all the CPU time to pass the data to the federate ambassador, leaving no time for the federate to do anything. In a real radio environment, where the federate has other computational tasks, this number would be much lower. Also, without hardware filtering of the unwanted audio data (which a well implemented DDM should provide) this places a fundamental limit on the total number of voice streams in an exercise.

By contrast, the current DIS radio products[2] available reach this limit at 1000 voice streams (ignoring 240 voice streams with 31 ms packets requires 25% of the CPU time on a 233 MHz machine without hardware filtering.) In addition, the current DIS radio products have been extended to use multicast addresses to filter unwanted audio in the hardware.

5. Conclusion

The DMSO RTI is capable of handling a modest number of voice streams, and can do so with an acceptable latency. A close examination of the network packets produced and of the time consumed by the CPU in receiving these packets show that audio is not handled very efficiently by the DMSO RTI. Furthermore, audio compression will not significantly help with the bandwidth consumption, and will consume even more CPU processing. Because of this, the DMSO RTI is not suitable for handling a large amount of audio data, especially if the audio federate has other time-consuming tasks to perform, such as managing a radio environment associated with those audio streams. It is also poorly suited in situations where bandwidth is limited. Because of this, for serious applications, back channels would be required for audio transport.

The MAK RTI has much more acceptable packet overheads and processing requirements. However, it is not yet a fully compliant RTI. Furthermore, without data distribution management services that filters at the hardware level, it places a fundamental limitation on the number of audio streams that can be used in an exercise. In its current form, back channels for the audio would also be required for any serious application.

Contact Information

A copy of this paper can be found at www.asti-usa.com. The author can be reached by e-mail at davidn@asti-usa.com. The DMSO RTI can be downloaded from hla.DMSO.mil, and the MAK RTI is available from www.MAK.com.

[2] The ASTi DACS, again

Figures

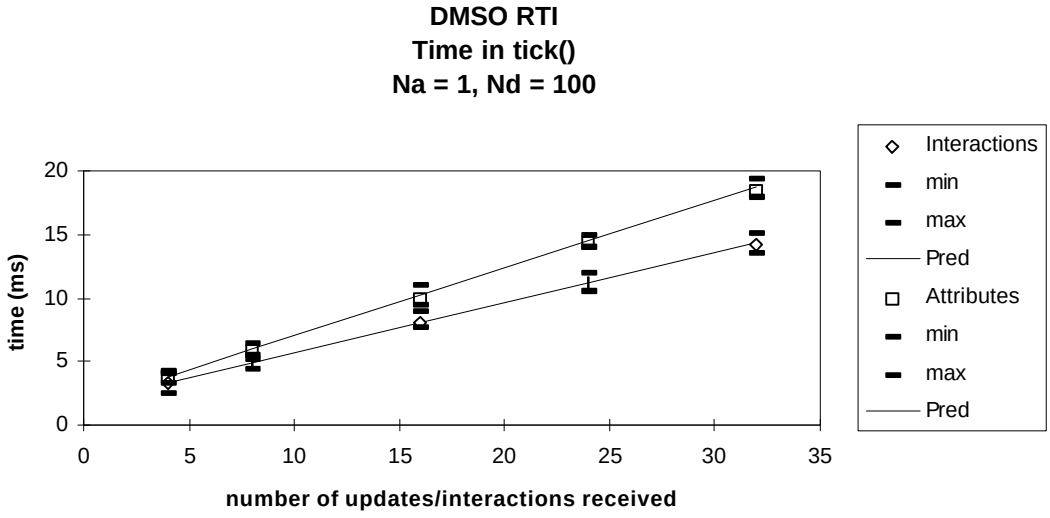

Fig 1.1 Processing time vs. Packets received, DMSO RTI

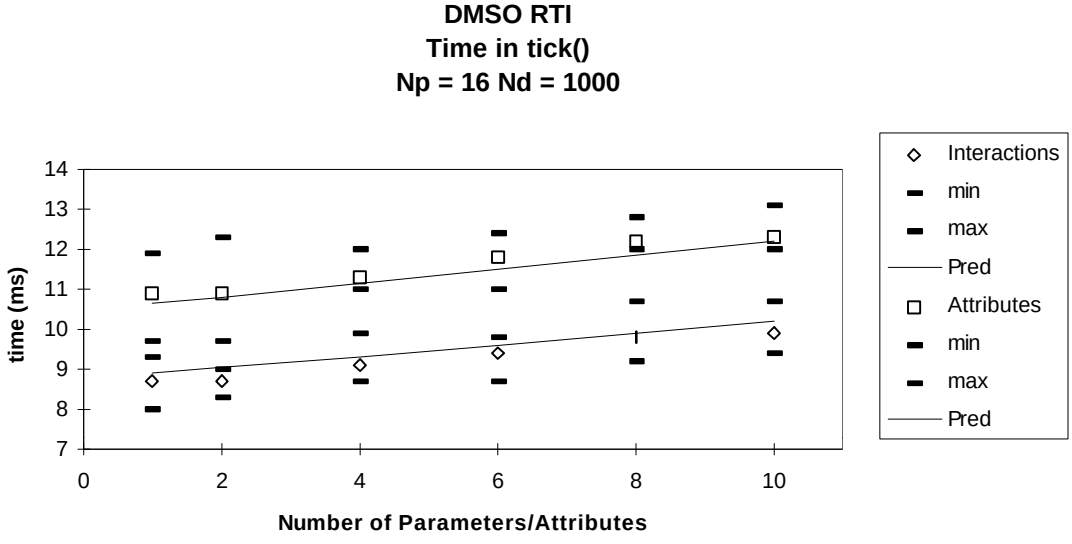

Fig 1.2 Processing time vs. The number of attributes/parameters in the packet, DMSO RTI

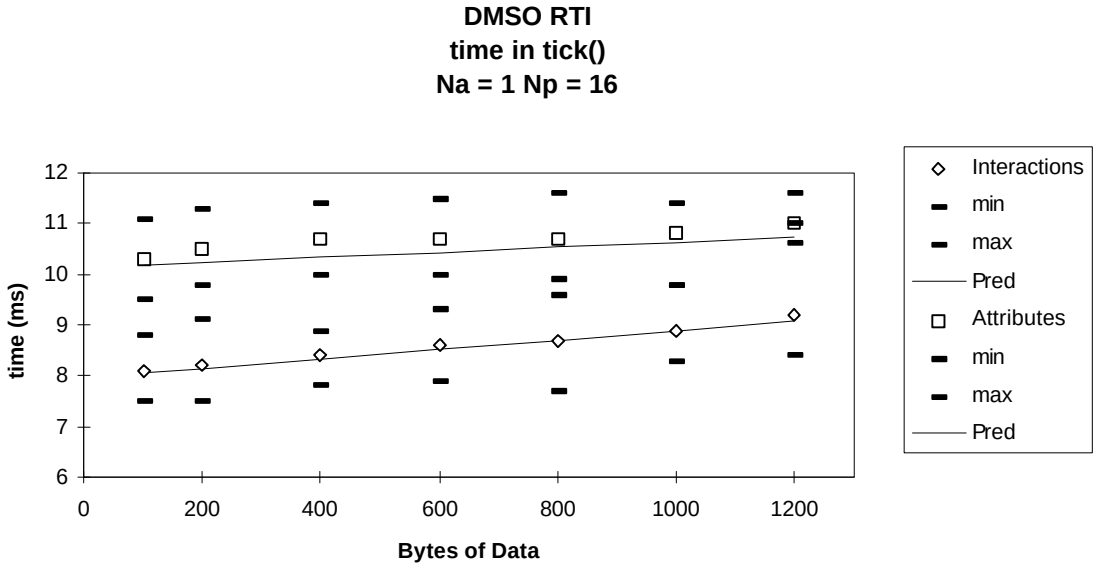

Fig 1.3 Processing time vs. The data size of the packet, DMSO RTI

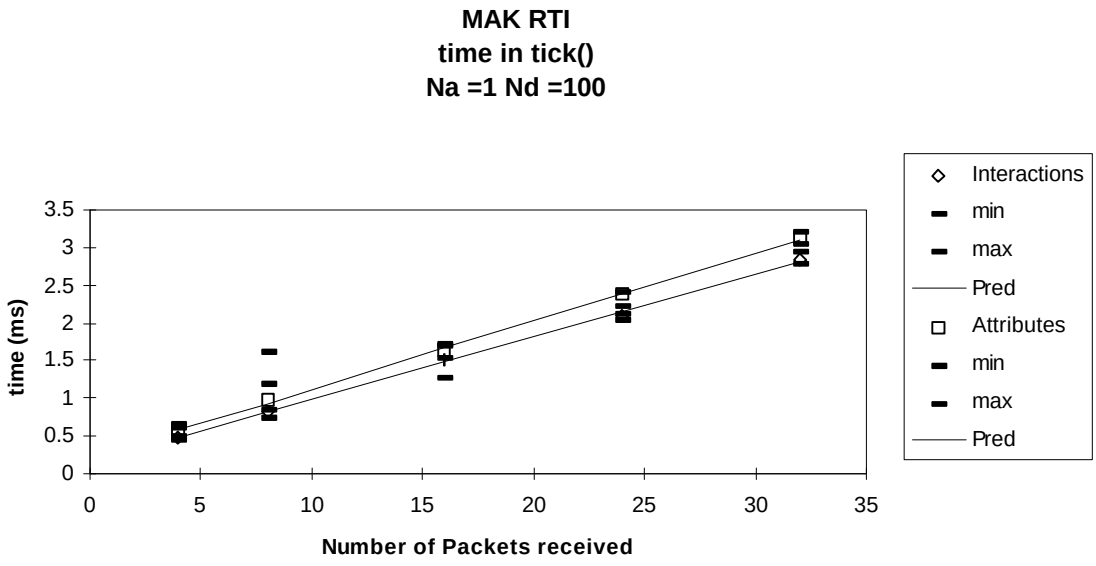

Fig 2.1 Processing time vs. Packets received, MAK RTI. Note the difference in scale on the y axis with Fig 1.1

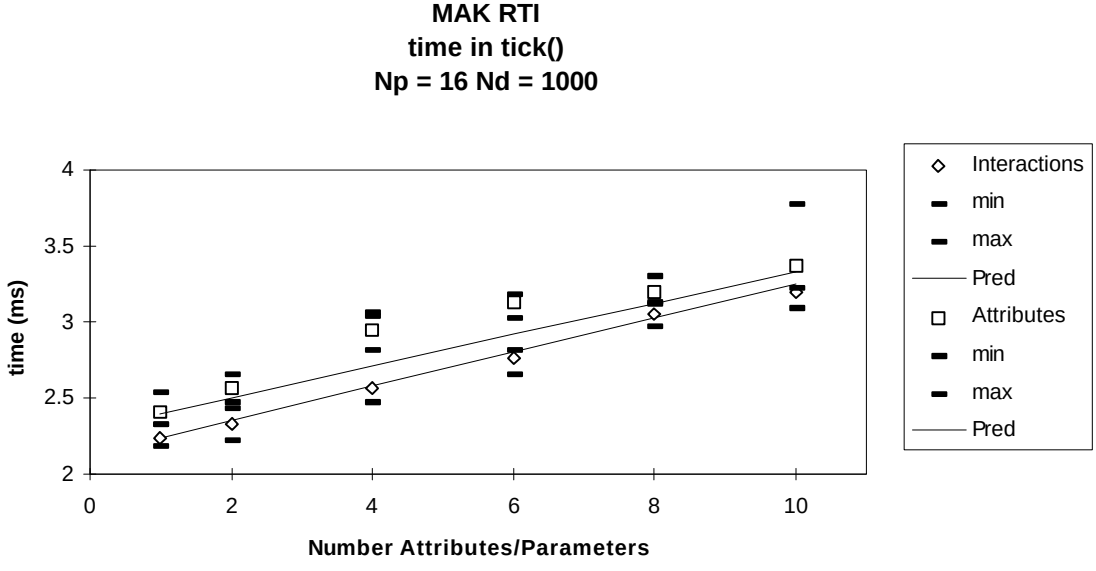

Fig 2.2 Processing time vs. Attributes/Parameters per packet, MAK RTI

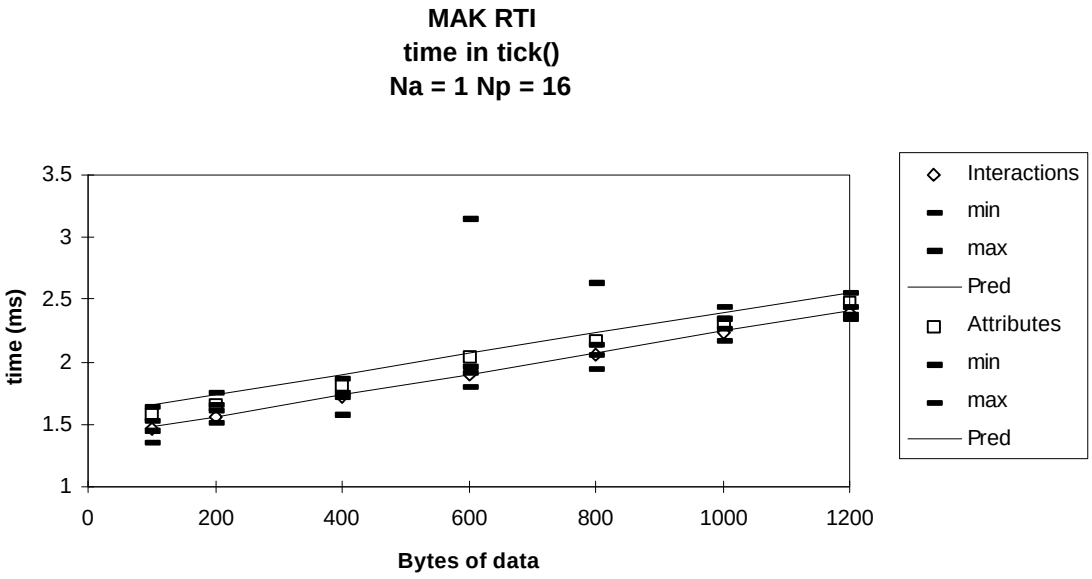

Fig 2.3 Processing time vs. Data size of packet, MAK RTI