The MÄK High-Performance RTI : Performance by Design

Shagoto Nandi *Felix Rodriguez* *Douglas D. Wood* *Len Granowetter*
snandi@mak.com, frodriguez@mak.com, dwood@mak.com, lengrano@mak.com

High-Performance is Our Middle Name

Although there are many criteria for evaluating and comparing RTI implementations, one of the most important is performance. Choosing an RTI that maximizes throughput and minimizes latency, bandwidth, and CPU usage can mean the difference between success and failure for an HLA simulation program.

Performance, however, is a difficult thing to quantify. There is not just one number that defines an RTI. There are many types of HLA exercises with wildly varying requirements. High performance on one exercise does not necessarily mean high performance on a different exercise. How many federates are you using? How many updates per second? Are you using a WAN configuration? Are you using any of the services such as DDM or time management? Are you using a Java or a C++ federate? Are you using HLA 1.3 or HLA Evolved?

The answer to all of these questions can have significant effects on performance. In order to provide the flexibility that meets the needs of most users, an RTI's configuration options must be robust. It should support the needs of most users out of the box. It must also provide the ability to reconfigure performance capabilities for the exceptional cases, if necessary.

In this paper we will present some of the design decisions we have made to provide an extremely fast RTI for nearly all HLA exercises. Before we get to *how* the design of the MÄK RTI has led to its unsurpassed performance, we'll present some real numbers on common situations for each of three major performance criteria.

Latency

Much of the literature on distributed simulations indicates that latencies of up to 30-100 milliseconds are tolerable without losing the feeling of real-time interactivity. Even a 3D graphics-based application running at 60Hz has 16 milliseconds in which to compute and draw each frame, meaning that latencies of 5-10 milliseconds may not even effect the time at which a particular event is drawn. Meanwhile, typical latencies for the MÄK RTI are closer to 100 *microseconds* on our gigabit network – fast enough to meet the needs of even the most sensitive real-time simulations.

In previous versions of the MÄK RTI, we optimized this low-latency level for HLA 1.3 federations and 100-150 byte payloads. However, the MÄK RTI now maintains near 100 microsecond latency regardless of what version of HLA you choose to use. Additionally, we have improved latency for large packets. In this latest version of the MÄK RTI, even HLA payloads of 5,000 bytes top out at around 140 microsecond latency.

Throughput

Throughput is a measure of how fast an RTI can write to and read from the network. Because throughput tells you how well an RTI can handle federations with large numbers of objects that are frequently sending updates, it is often an even more important metric of RTI performance than latency. In many real-time platform-level simulations, updates or interactions that contain 100-150 bytes of data are fairly typical. For packets this size, we have demonstrated a throughput of over 170 thousand packets per second on our test system.

For larger packets, we do even better. In fact, for packets with 5000 bytes of payload data, we have achieved a throughput of over 22 thousand packets per second, around 90% of the theoretical maximum for a 1 gigabit network. In our original test system, we consistently topped out at 90% payload usage (not counting our minimal HLA overhead); we re-ran all our tests in a 10 gigabit network to get a better idea of what our limit is and we measured over 60 thousand messages over 5,000 bytes per second.

It should be noted that these benchmarks test an individual federate pushing packets out on the network. Most distributed exercises would want to have multiple federates sending packets. In this case, a complete exercise could be much larger than 170 thousand packets per second.

CPU Usage

As far as CPU usage goes, we are so proud of the efficiency of our RTI code that we added a panel to the RTI Assistant GUI to show the time spent in each RTI service call. On our test system, an average call to subscribeObjectClassAttributes() takes 15 microseconds.

Just the simple act of sending over 150 thousand messages takes a lot of processing power. However, it is extremely unlikely that any currently existing simulator can handle even close to as many entities as that number of messages would imply. The simple act of updating position on our F-18 example took more CPU than the MÄK RTI uses when sending messages.

The MÄK RTI is unlikely to be your CPU bottleneck and because we use non-blocking I/O in separate threads by default, sending and receiving will not affect the performance of the main thread of your simulation.

A Word about Benchmarks

Of course, like all benchmark tests, these were performed under lab conditions: only two federates, no other traffic on the network, federates that do nothing but send and receive packets, etc. We used off-the-shelf PCs without any special networking hardware. The computers used were driven by an Intel Core i7-3770 CPU (3.4 Ghz), using Windows 7 64-bit, in a 10 Gigabit network. These numbers can certainly be used as a starting point to evaluate the raw performance of the MÄK RTI. Of course, most federations will want to execute side-by-side comparisons with real federates and real-life network topologies. Such 3rd party studies and comparisons have consistently proven the outstanding efficiency and usability of the MÄK RTI.

For the purposes of providing our customers the ability to prove performance in their communication, the RTI Assistant should make things easy. If you have Network Statistics Monitoring turned on, you can track every message sent by your federates, as well as how long different callbacks take to process. This is a great resource to find any unexpected bottlenecks in your exercise as well as a benchmarking tool. You don’t have to take our word for any of this, you can try it yourself!

What Does Performance By Design Really Mean?

Early RTI implementations had performance characteristics that were unsuitable for high performance simulations. These problems created a perception that HLA was inherently slow, and that perception hindered the adoption of HLA. When we began to develop the MÄK RTI, our foremost goal was to create an RTI that met the performance needs of the Real-Time community. By designing a "wire format" specifically for the purposes of the MÄK RTI, we ensured that our packets were as compact as possible.

As the MÄK RTI evolved, we applied the same performance-focused approach to implementing additional services like MOM and DDM. After all, flight-simulators aren't the only applications that care about performance. As we implemented these services, we adhered rigidly to the principle that new functionality was not allowed to impact the performance of the initial subset. RID file parameters allow you to disable services that you do not need. If you do not have Time Management or DDM enabled, for example, the MÄK RTI will skip that section of code and perform as if the service never existed. In general, latencies are not affected by additional services, since our communications strategies have not significantly changed since the original subset RTI was released. We have also taken great pains to make sure that you do not pay a penalty for unused services in terms of computation time. For example, if MOM is not enabled, we avoid collecting the information that would be necessary to send MOM updates.

Performance by Design also means a realization that different simulations may have widely varying requirements, and therefore might be willing to make differing performance tradeoffs:

- If the simulation must run across the WAN or via low-bandwidth network links (such as satellite or radio), then minimal bandwidth usage may be the primary concern.
- For processor intensive simulations such as a Stealth, low CPU utilization will be of paramount importance.
- Voice transmission is highly dependent on latency that is both low and predictable.
- For simulations involving very large numbers of entities, high throughput and low bandwidth overhead may be a necessity.
- Finally, for large exercises incorporating many individual players and federates, scalability may be the deciding factor.

In many cases, the MÄK RTI offers a choice between several algorithms, and puts the power to make performance tradeoff decisions in a *user's* hands. Although most federations can expect outstanding performance out of the box, various RTI settings offer a high degree of configurability. For example, bundling can be enabled to increase throughput and minimize bandwidth usage, at the expense of a little extra CPU processing and latency. In addition, MÄK offers the customer the ability to fully customize the RTI to meet the needs of a specific federation through the Plug-in API.

HLA is a standard, so how different can the implementations be?

HLA provides a well-defined API and in some cases there is an obvious mapping from the API to a particular implementation. However, there are many areas of the HLA Specification that allow for a variety of possible design choices. In fact, this was the intent of HLA – to allow vendors to compete and create multiple implementations that make different trade-offs so that a user can pick the one that best suits the needs of his federation.

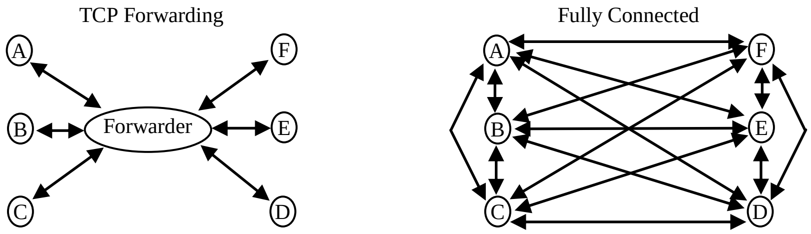
Some of the biggest differences between existing implementations are in the following areas:

- 3rd party networking layer vs. custom wire format
- Reliable transport implementation
- Federate Ambassador callback strategy
- Availability of a light-weight mode
- WAN efficiency
- DDM and Time Management implementations

Let's look at a few of the specific implementation choices made by the developers of the MÄK RTI, and examine the effect of these choices on performance.

Reliable Transport – the TCP Forwarder Approach

Because the TCP protocol supports only one recipient at a time, RTI developers need to choose between two strategies when packets need to be sent over TCP to multiple recipients. Either each federate's LRC needs a connection to every other federate, or some forwarding scheme must be used where a packet is sent to a forwarder, who sends a copy to each recipient. The diagrams below show the basic architecture of both designs in a federation of 6 federates.



The biggest advantage of the Fully Connected scheme is that it minimizes latency by requiring only a single network hop for a packet to reach a recipient. However, the extra latency associated with the second network hop in the TCP Forwarder approach amounts to only a fraction of a millisecond. The MÄK RTI is able to achieve latencies for reliable traffic that are well under a millisecond, even though we employ a TCP Forwarder. As we will see, this minimal amount of extra latency is a small price to pay in order to gain the other benefits of the Forwarder approach. Besides, federations typically send their most latency-critical data via Best-Effort in order to take advantage of UDP's multicast capability.

The other advantage of the Fully Connected design is that it requires the minimum total number of transmissions of each message. The TCP Forwarder requires *one* extra transmission to get a copy from the sender to the forwarder. With federations of only two federates, the TCP Forwarder approach actually requires twice the bandwidth – two transmissions instead of one for each message. (This is why the TCP Forwarder often appears to compare poorly in benchmarks - benchmarks are generally run with only 2 federates.) However, the impact of the extra transmission decreases as the number of federates increases. For example, in a federation with 20 interested federates, an RTI that uses a TCP Forwarder requires 21 transmissions, as opposed to 20 for a Fully Connected RTI.

The TCP Forwarding architecture chosen by the MÄK RTI provides several major advantages over a Fully Connected network. The most important is that a TCP Forwarding approach moves the computation burden of sending multiple copies of each packet out of the real-time application and into a dedicated process on a separate machine. Adding federates to your federation has *no* impact on the time it takes a federate to send a reliable message in a TCP Forwarding architecture. In contrast, unless you have a multi-processor machine for each federate, the amount of CPU time required for a federate to send a message increases linearly with the number of recipients. Of course, this leaves fewer cycles available for the federate to actually do its work. This makes the TCP Forwarding approach far more scalable than the Fully Connected approach. As the number of federates increases and a single TCP Forwarder can no longer keep up with forwarding all of the federation's reliable traffic, additional Forwarders may be added to distribute the message load. No such load balancing is possible in a Fully Connected network.

The TCP Forwarding approach also greatly simplifies the network map of the federation and leads to many internal benefits to the RTI. It simplifies tasks such as federate fault tolerance and greatly improves the speed of tasks such as federate join since each Local RTI Component only needs to establish a single TCP connection (to the Forwarder). It also allows RTI implementers to easily add features that require centralized message processing or filtering. For example, the forwarder can easily filter messages based on subscription interest to lower the bandwidth required by reliable traffic. The MÄK RTI does exactly that when “Smart Forwarding” is enabled.

Additionally, the TCP Forwarding approach allows us to limit internal messages to only the federates that need them. For example, In the MÄK RTI 4.3 and later, if a new federate joins the exercise, or a new object is created, update requests and responses will be sent directly to that new federate and the rest of the network will not be overwhelmed with request messages. This is particularly useful for high load networks since the internal messages are sent reliably with only minimal overhead.

Finally, a TCP Forwarder approach is more suitable for Wide Area Networks (WANs). Through the use of multiple forwarders, we can ensure that only one copy of each message is sent between each pair of LANs. (The forwarder on the receiving side resends to each local federate.) With a Fully Connected approach, a separate copy is sent between each pair of federates, meaning that many copies of a single message might be sent across the same link between two particular LANs.

Which callback strategy is best?

The major RTI implementations offer 3 basic strategies for how to pass incoming data to a federate through Federate Ambassador callbacks: single-threaded synchronous I/O, fully asynchronous callbacks, and tick with asynchronous I/O.

The single-threaded method is the simplest of the three major callback strategies. The federate calls the RTI Ambassador tick() method from the main federate thread. At that time, the RTI reads any pending messages from the network and calls the corresponding callbacks sequentially. This method entirely avoids the overhead of communication between multiple threads. However, in return for this simplicity the federate must accept the overhead of making network I/O calls within the tick function. In addition, federate code may need to be tuned to call tick regularly enough that the network layer's buffers do not overflow and drop messages.

In the asynchronous callback method, federate callbacks are invoked from a secondary thread as messages are received rather than waiting for a tick call. Perhaps the biggest advantage of the asynchronous callback method is that the overhead of network I/O is moved to a secondary thread, so the federate code is free to perform simulation work even when the network is heavily loaded. For some applications, the use of asynchronous callbacks also offers a way to reduce the latency inherent in the callback mechanism. In particular, in those cases where a federate is able to handle the new data entirely within the secondary thread, latency may be improved.

However, in most cases, the perceived latency savings are an illusion. It is true that a callback may be executed earlier if it is allowed to happen asynchronously. But most federates design their callbacks to merely queue the data that they receive, or to store new attribute values in a database of objects that will be accessed by other parts of the federate code later. If this is the case, nothing is really gained by using asynchronous callbacks. Like a car that has raced ahead only to arrive at a red light sooner, data will reach the federate's queue or object database faster, only to sit there longer before it can actually be used by the federate.

The dubious gains associated with asynchronous callbacks come at the cost of (possibly greatly) increased complexity in federate design. Asynchronous callbacks put the burden on the federate developer to make callbacks thread-safe. The main simulation thread no longer has control over when to accept data from the RTI and locking must be employed to ensure that callbacks are not writing to the federate's object database while the main federate thread is reading from it.

Asynchronous I/O offers a compromise between the single-threaded tick and multi-threaded asynchronous callback methods. For asynchronous I/O, the RTI creates two separate threads to handle network I/O. Messages are read from the network, and stored in a queue within the RTI, so that they are available to the federate immediately when tick is called. Since the threads are internal to the RTI, asynchronous I/O can be enabled with no change to federate code, and there is no burden on the federate to maintain thread-safety. At the same time, it provides many of the advantages of asynchronous Callbacks: Messages are read from the limited network buffer immediately as they arrive, which can greatly reduce the number of dropped messages in a heavily loaded federation, and the overhead of reading from the network is moved out of the federate's main thread.

Because asynchronous I/O provides most of the benefits of asynchronous callbacks without any of the extra complexity, we have implemented this approach by default in the MÄK RTI. We still provide the ability to use the simple single-threaded tick model, because it can be easier to debug and because it requires the least total run-time overhead (no need to copy messages into and out of a queue). We also provide the ability to have full asynchronous callbacks for customers who prefer that method of data transfer and do not mind the extra complexity.

What are some of the MÄK RTI's Unique Performance-Enhancing Features?

Under the hood of the HLA API, RTI implementers are free to add their own solutions to common federation problems. For example, on a single LAN, RTIs typically use multicast or broadcast to reach a number of recipients with a single packet transmission. However, most WANs do not support broadcast or multicast. Like with TCP, only point-to-point traffic is supported. The MÄK RTI allows users to minimize best-effort traffic across the WAN via a UDP Forwarder. Like a TCP Forwarder, the UDP forwarder acts as a single point of contact for reaching all federates on a remote LAN. Rather than sending a separate UDP message across the WAN to each remote federate, a single UDP message is sent over the WAN to a forwarder, which then redistributes the message via multicast on its LAN.

Another key feature of the MÄK RTI is its class-based multicast filtering solution. Often described as “poor-man's DDM”, class-based multicast filtering offers many of the advantages of DDM without any of the overhead and without requiring changes to federate code. Federation designers map FOM classes to multicast addresses through RID configuration file changes. For example, all radio traffic can be confined to a specific multicast channel that is different from the one used for entity state data. Filtering of unwanted data occurs at the network card level without any processing by federate or RTI code, greatly reducing the CPU load imposed on each federate by the RTI.

Of course, the most unique tool that the MÄK RTI provides for optimizing performance is the

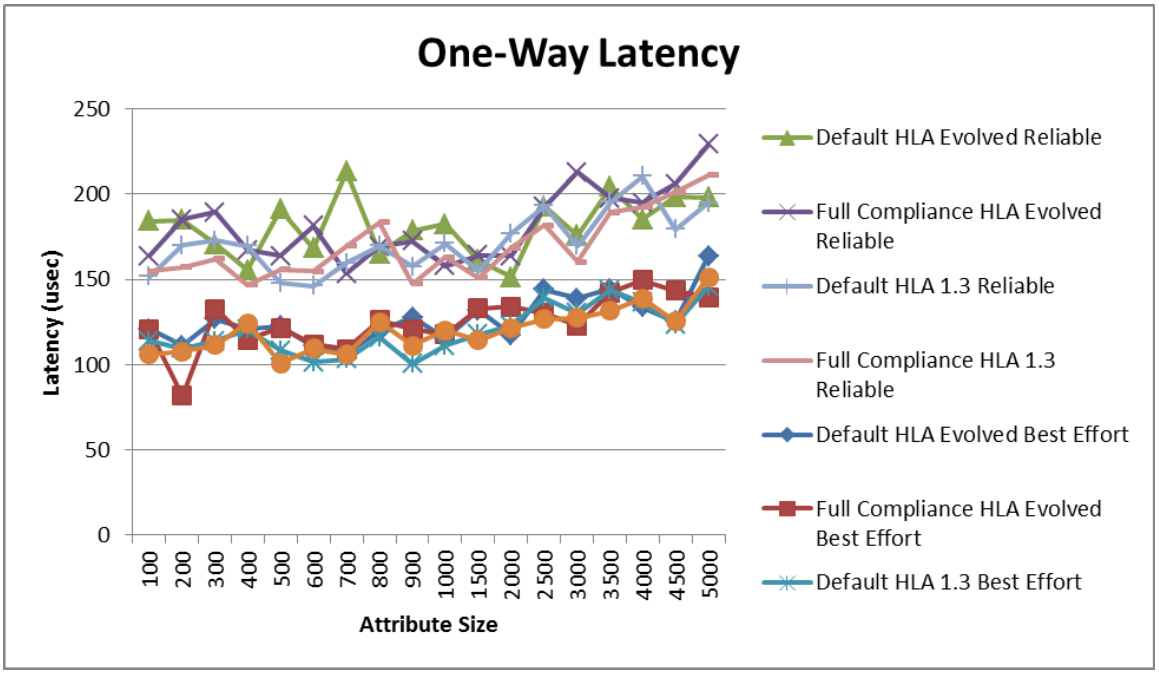
RTIspy plug-in API. The API allows federation designers to truly open the black box of the RTI and customize almost any aspect of the RTI's functionality to meet the specific needs of their federations.

How Does MÄK Plan to Help Customers Achieve their Future Performance Goals?

The MÄK team understands that performance requirements for simulations will only become more ambitious and in order for our customers to succeed, the MÄK RTI must continue to rise to the challenge. The MÄK RTI is continually evolving as features and optimizations are added to the software based on customer needs. New features such as packet bundling and “Smart TCP Forwarding” have been added to recent releases and new solutions are on the way. For example, shared memory implementations have already been demonstrated within the MÄK RTI using the RTIspy plug-in API and these will soon be wrapped into the RTI itself to help our customers take advantage of multi-processor machines. “Implicit DDM” will allow the RTI to take advantage of federation-specific knowledge to intelligently route data only to those federates who need it. Research is already underway into additional communication technologies such as RUDP (a fast, reliable protocol built on top of UDP), compression, Quality of Service differentiation, and more.

Latency Benchmark Information

Below is a graph comparing the latency observed in a variety of test situations. In all cases, the tests were run on 3.4 GHz i7-based PCs running Windows 7, connected over a 10 gigabit ethernet through a 10 gigabit switch. The lines for latencies through the RTI are actually very close to the expected latency for raw UDP data, indicating that the RTI itself adds little overhead. The overhead that is present is largely due to the fact that the RTI must serialize data from an AttributeHandleValuePairSet into a network buffer before sending and must reconstruct an AttributeHandleValuePairSet from the serialized data on the other side.

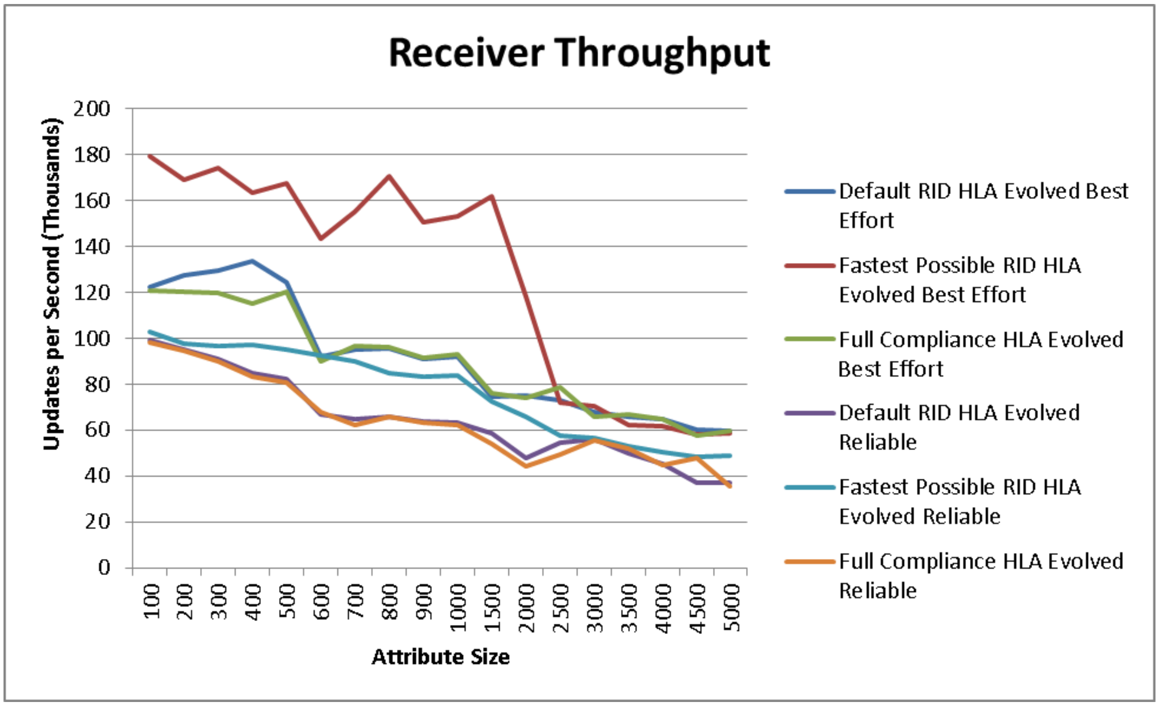


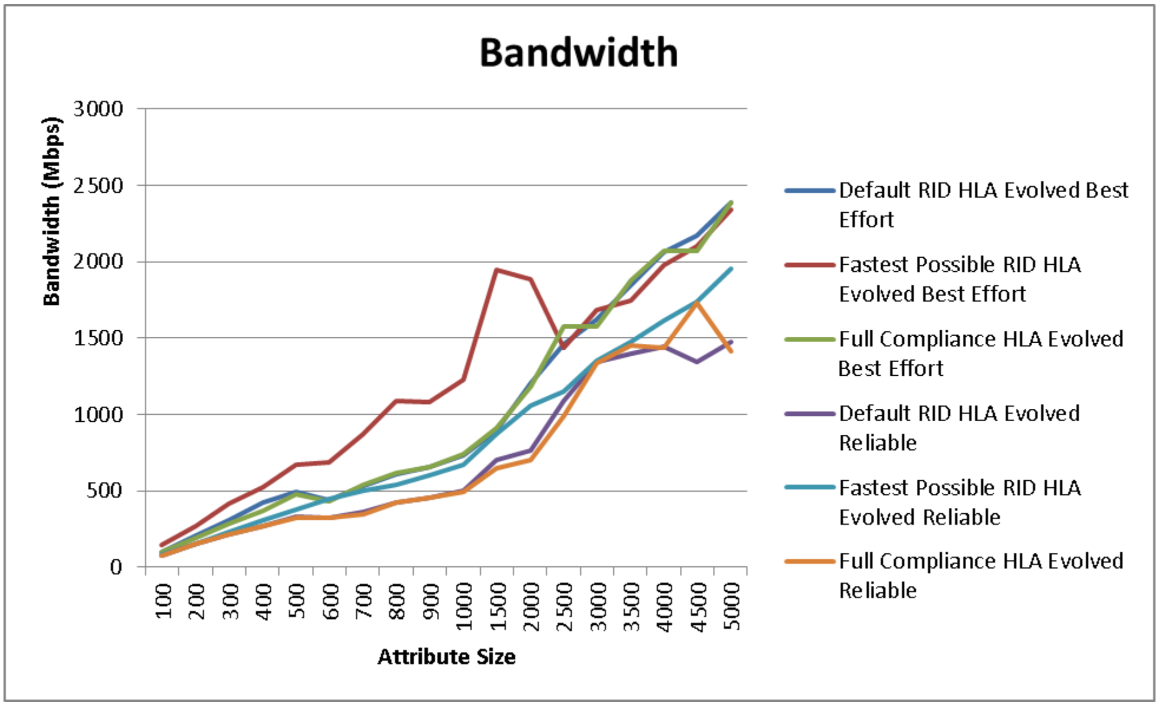
For reliable transport, the relevant comparison is the RTI's latency against the raw network latency for two TCP network hops. Even though the numbers include the time a packet spends inside the TCP Forwarder, total end-to-end federate-to-federate latency for reliable transport is well under a quarter of a millisecond, even for the larger packets.

For an RTI using TCP forwarding (as does the MÄK RTI), it is critical that the TCP Forwarder run on a dedicated machine. If the TCP forwarder is run on the same machine as a federate, then the federate process will dominate the CPU, the TCP forwarder is starved for CPU time, and messages will become backlogged. It is our experience that any time a third party benchmark result seems to show the MÄK RTI having poor performance for reliable transport, it turns out that they have been running the rtiexec (where the TCP Forwarder lives) on the same machine as a test federate.

Of note in this chart is the fact that the protocol, HLA 1.3, or HLA Evolved, has little effect on latency. We did not fully run the tests on HLA 1516-2000, because we found they tracked HLA Evolved closely. Also noticeable is the fact that forcing the RTI to be fully compliant with all the HLA services turned on also has little effect on latency. The only thing that really has any significance is changing the transport protocol between best effort and reliable.

Throughput Benchmark Information





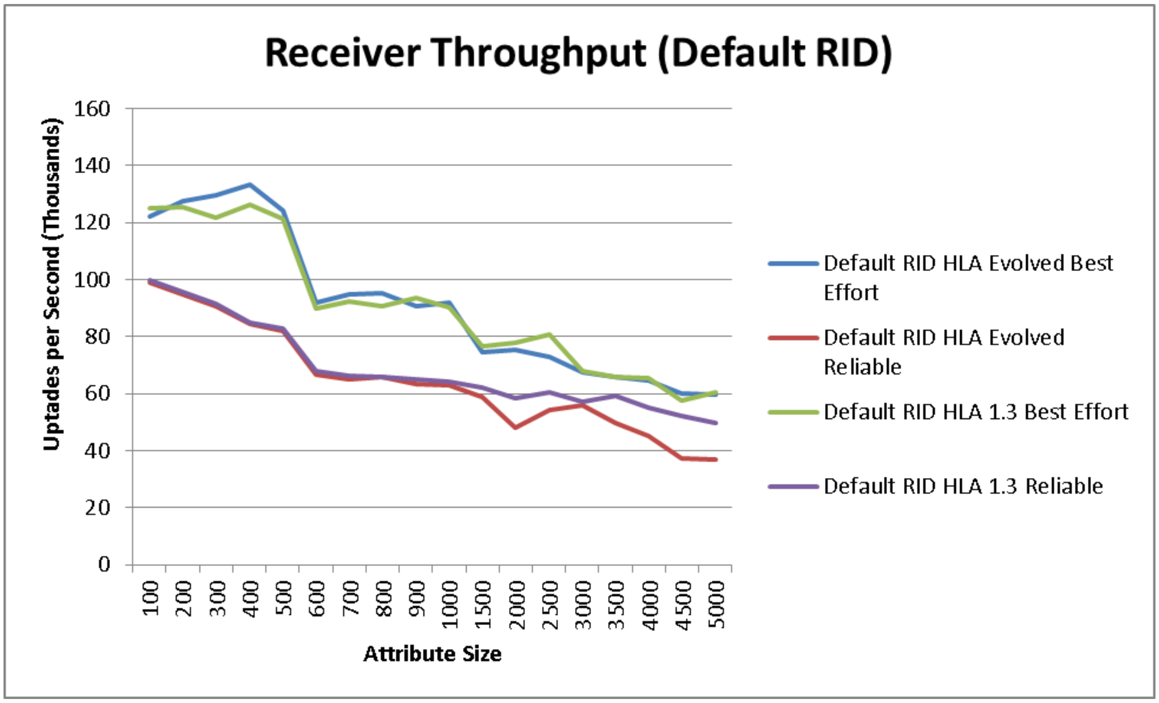
The two graphs above show the results of our throughput tests using three different RID files. The first one is the default RID, that ships with the RTI. The second RID is a fully compliant RID, which is what you would use if the "force full compliance" checkbox is selected. As one can see, both of these RID files provide very fast throughput, over 120 thousand messages with small data sizes (100 bytes).

It's hard to visualize how large 120 thousand updates a second are, so it's best to put this into perspective. Spatial information of an entity takes exactly 96 bytes. That includes position, velocity, and acceleration for both linear and rotational components. At any given moment there are approximately 9 thousand commercial aircraft flying in the world. This means that a single MÄK RTI federate could transfer positional information of every one of these aircraft a little over 13 times a second.

That doesn't take into account dead reckoning and smoothing, however. Unless every single one of these aircraft is doing fast maneuvering exercises, they would really need to update a few times a second. Let's reduce the update rate to 4 times a second, a number which is quite reasonable. This gives enough time for the federate to transfer spatial data for every single one of the approximately 35,000 military aircraft around the world as well, in the unlikely event that they were all activated at the same time.

The third RID, which in the charts above we call "Fastest Possible" is a modification of the default RID with the following options: bundling on and at 5,000 bytes. MOM, DDM, update rate reduction, and network statistics monitoring all turned off. We found that this RID, provided the fastest numbers, at well over 170 thousand messages a second (again, 100 byte payload). In this RID, we also modified the socket send buffer to be 20MB as opposed to the default 2MB. This change only affects HLA Evolved exercises running in reliable mode, with large attribute sizes. Otherwise, there is no need to make the send queue larger. This is explained a little more below. It should be noted, however, that this RID was the fastest results we found in our exercise, but depending on your configuration, there may be a RID that is more optimal for you.

As data sizes increase, network bandwidth generally increases due to lower processing requirements and proportionally lower per-packet bandwidth overhead. Of particular note is that a single federate running the MÄK RTI can reach 1 gigabit bandwidth at packet sizes of just 800 bytes. At even larger byte counts of 5,000 the federate was approaching 2.5 gigabits. Originally, our tests were made with a gigabit network, but we found the MÄK RTI could handle 90-95% the theoretical maximum bandwidth of the network too easily. For this reason, all performance tests were done on a 10 gigabit network.



The graph above shows the results of our throughput tests comparing transport types. Results are shown for best effort and reliable transport. For reliable transport, effective throughput drops by roughly 20% because each packet is actually sent to the network twice – once from sender to Forwarder, and once from Forwarder to recipient, as well as TCP overhead versus UDP. If you have multiple federates, then reliable transport comes at a higher network usage cost, as forwarders need to send one message per federate. However, the individual federates still only send one message, so it will have no effect on your local simulation unless the forwarder is fighting for the same CPU time.

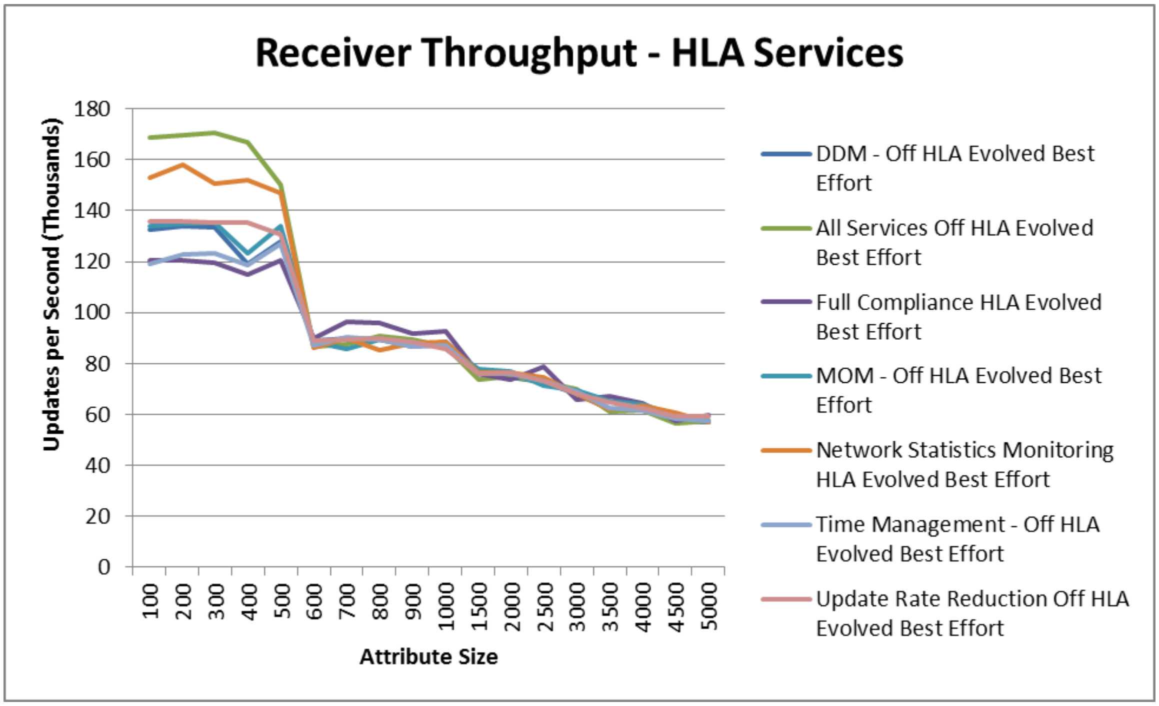
Like the latency tests, and unlike previous MÄK RTI versions, the protocol chosen, HLA 1.3 or HLA Evolved has a minimal effect on performance when using best effort. Using reliable mode, however, HLA Evolved becomes somewhat slower when sending large packets. There is an easy work around however. If you set the send buffer queue to 20mb instead of the default of 2mb, HLA Evolved regains the lost speed and matches up to the HLA 1.3 reliable results. This will result in an increase in memory usage for each federate, but otherwise there is no other negative side effect of this change.

In conclusion, if you are using HLA Evolved, and reliable mode, and want to send messages larger than 2 thousand bytes, we recommend that you increase the send buffer. Under any other circumstance, there is no need to modify this value for performance.

Individual HLA Services Benchmarks

One major advantage of the MÄK RTI is its ability to turn HLA services on and off. If you are not using DDM, for example, you can have the RTI turn that feature off to get a performance increase.

Two things need to be noted when using this feature. First, even with all services turned on, the MÄK RTI is very fast. The test federate could still send over 120 thousand updates per second. That is much more than every simulator that we know of, so users really should not fear leaving all services on. Second, every service has its own overhead cost, as is shown in the following chart:



As you can see in this graph, turning off HLA services produces the most performance improvements when sending smaller attributes. At larger than 500 bytes, there doesn't seem to be any significant advantage in turning off any services.

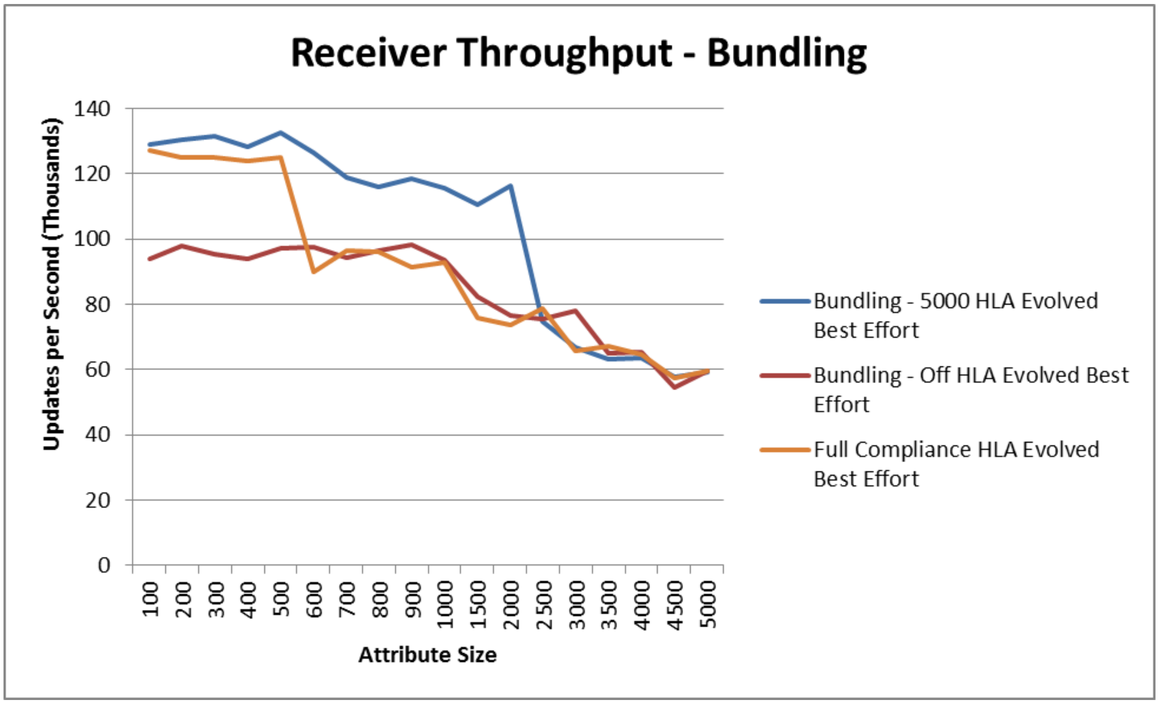
The most time consuming service is the network statistics monitoring feature that the RTI Assistant provides. At 100 byte payloads, turning off this feature causes a 30% increase in performance. (From 120 to 155 thousand updates per second). We think it's worth it and 120 thousand updates is more than enough for 99% of HLA Exercises, but when you really need more throughput, this is an easy thing to turn off.

Most other HLA services provide a similar 10% improvement when turned off. This is the case for update rate reduction, MOM, and DDM. On the other hand, turning off time management provided no benefit whatsoever. When time management is not in use, there is no overhead for it even if it is turned on. So there's really no reason to ever turn that setting off.

All in all, in the best case scenario turning off all services, provides about a 45% increase in performance for small updates, but a minimal increase for large updates.

Bundling and Compression

In addition to turning services on and off, the MÄK RTI provides a few ways to reduce the traffic in the network. The two most commonly used methods to do this are bundling and compression. The ideal value to set both of these features varies by the type of simulation being done. Thus it is best to understand their effects on traffic to use effectively. The following graphs show the effects of bundling on network throughput:



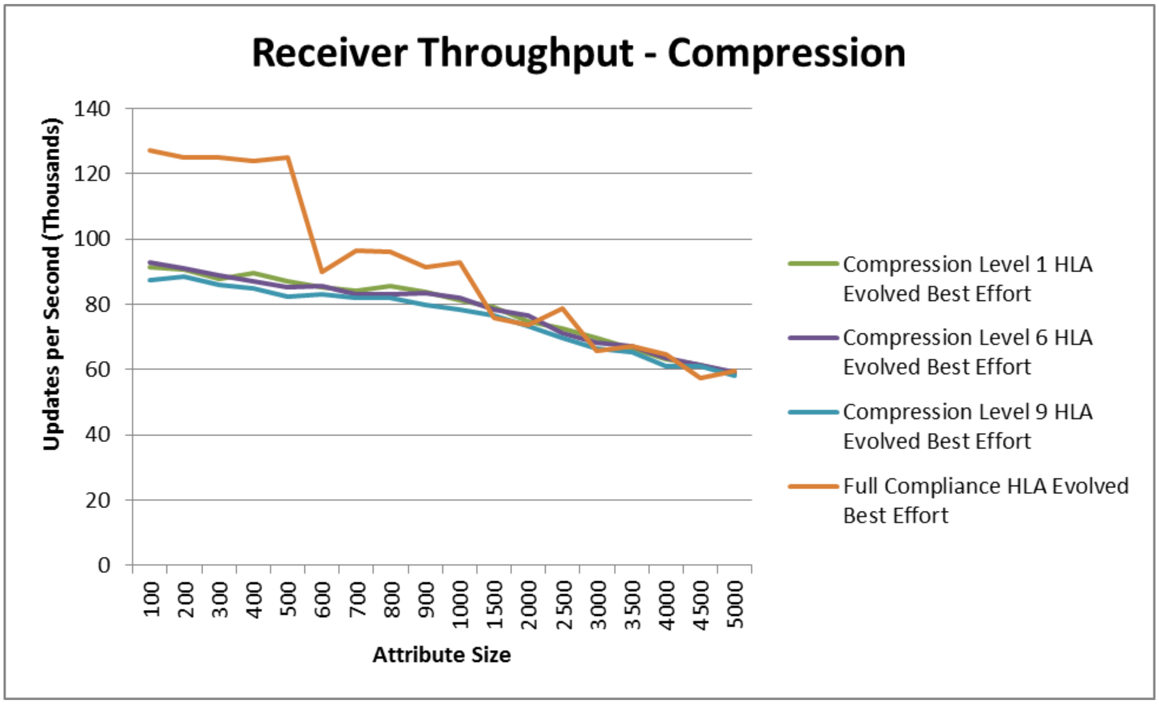
The above graphs show our test application with message bundling turned on and off. For this test, the bundling was set at the default 1400 bytes, or just a little under the UDP packet maximum. We also show bundling at 5,000 bytes for comparison.

A cursory look at the graph will show a significant speed improvement on small message counts. The improvements then start decreasing until you actually get a small penalty under medium message counts. Once messages become bigger than the bundle value, bundling stops occurring and the performance results are the same as not bundling at all.

Understanding why this happens is the trick to knowing when to set message bundling. On small and very frequent messages, message bundling dramatically improves performance by reducing the significant cost of the TCP or UDP socket. This is at a price of an extra copy, but as you can see by the graph, it is worth the cost. However, once you reach the point where you can only send one message per packet, now you actually are doing worse than you would with message bundling turned off, since the extra copy is still there. You get the entire penalty and none of the improvements.

It should be noted that if your simulation is using RPR and is mostly sending entity updates - those are perfect for bundling. In fact, it has been our experience that most HLA exercises send small sized updates, therefore we recommend that unless you know you are doing something special, you should keep message bundling turned on.

When looking at this chart, one might think that even larger bundling values would be better, as a bundling value of 5000 provides a small but clear improvement in throughput even at smaller counts. There are drawbacks with going bigger, however. Larger packets have a higher tendency to drop while being sent through UDP. At 5,000 bytes, the drops were still minimal, but increasing that value further, will certainly degrade your packet completion count. For that reason, we have defaulted bundling to 1400 bytes, which should still cause significant speed up but is well below the safety line for dropped packets.



As you can see above, message compression is almost the exact opposite of bundling. For small-sized messages, compression doesn't really provide much improvement in payload size and there is a very real CPU cost to compression.

If processing time is your limitation, as is the case for most federates, then compression would be counterproductive. You might send smaller sized packets, but your throughput will be negatively affected. Even if you are sending larger messages of easily compressible data as we are in this test, there is no throughput gain at any scale. Most large RPR-based simulations will probably not benefit from compression either, given the expected small size of the average messages.

Compression does reduce the number of bytes sent on the network. Therefore it can provide a faster exercise when faced with an overloaded network. Therefore, if your exercise is running in a 3G environment or large packet 1 gigabit exercises, then you should definitely experiment with this setting to find your optimal results.

Conclusions

There is no standard set of benchmarks available for independent 3rd party RTI benchmarking.

The numbers presented in this paper cannot be directly compared to numbers generated using different tests since they will use different methodology. In addition, many factors in federation performance are beyond the scope of these benchmarks, such as an RTI's DDM and time management implementations that strongly impact scalability when used. However, the source to the MÄK RTI benchmark suite will be made available and customers are welcome to use it to perform their own comparisons.

The benchmarks demonstrate the excellent performance of the MÄK RTI. As the tests have shown, it is generally the underlying network and federate processing rather than RTI processing which is the true bottleneck in performance. Real simulations run in less than optimal conditions and will see varied results, but the MÄK RTI allows simulation designers to focus on the performance of their federates and have confidence that the RTI will support them.

Of course, performance is just one of many criteria federation developers use to decide on a choice of RTIs. Usability, extensibility, portability, robustness, and a vendor's customer support are all often equally important features to evaluate. The MÄK RTI compares favorably in all of these categories - just ask our users.