App timestamping and MiFID 2: Myths and mysteries

Blog: www.STACresearch.com/aleblog
By Peter Lankford, Founder & Director, STAC
September 10, 2017

The STAC-TS Working Group has poured a lot of effort into developing standards and software for demonstrating timestamp accuracy to the satisfaction of regulators. Much of that effort has concerned software-application timestamps. Application timestamping is crucial since it's often the least expensive way to comply with the reporting requirements of MiFID 2. That's because most apps already timestamp and capture reportable events. It's also because there are certain events that the regulations explicitly require to be timestamped within applications.

There are two sources of error in application timestamps:

- *Host clock error*. This is the difference between UTC and the time of the clock in the server or desktop where the app is running.

- *Application-level error*. This is primarily the time it takes the system to assign a timestamp that has been requested—i.e., timestamp-assignment delay. Timestamp-assignment delay is caused in small part by the instructions necessary to obtain the timestamp, in larger part by things like cache misses, and in largest part by gross disruptions such as scheduling jitter in the operating system or garbage collection in a virtual machine. There are secondary components to application-level error such as resolution of the timestamp mechanism, which can be fairly coarse in virtualized environments; but timestamp-assignment delay still causes the biggest outliers.

I'll discuss host-clock error in a subsequent blog series. The focus of the present blog series is **application-level error**. The reason for focusing on this first is that it is often one of the biggest potential compliance issues with RTS 25, for reasons I will explain.

It is also one of the most misunderstood issues. When I discuss application-level error with people outside the STAC-TS Working Group, or come across vendor points of view, I often encounter certain myths that can be risky or expensive if they are the basis for a compliance strategy. So I thought it was worth examining them in a blog series. Along the way, I thought I'd also point out a couple mysteries around how MiFID 2 will be interpreted.

To support some of my points throughout the series, I've inserted data that summarize the application-level error characteristics of different system configurations. These measurements (STAC-TS.ALE benchmarks) are automatically created and analyzed by the STAC-TS toolset when run on a given system. The tools conduct tests under a variety of load conditions representing challenging cases in production and apply a conservative analysis to the results.

MYTH #1
Application-level error doesn't matter because
it does not affect the sequence of reportable events.

Some people have argued as follows:

> *"The point of the timestamp-accuracy requirements of RTS 25 is to establish a clearer sequence of events. As long as nothing happens between an event and its timestamp, there's no practical importance to delays in timestamping because they do not change the sequence of events. If my business logic is blocked from timestamping, it is also blocked from doing anything else, so there is no significance to the timestamping delay. For example, suppose my app makes a decision to*

deal, then sends out a corresponding order. And suppose that due to some OS scheduling issue, the timestamp on the decision to deal is assigned two seconds later than the actual decision (to use an extreme number). As long as the order does not go out until after the timestamp on the decision to deal has been assigned, there is no importance to the fact that the timestamp was delayed. The timestamp on the order-send event will still be after the decision to deal."

This argument is indeed effective in certain situations. The problem is that is not effective in others. In general, plenty can happen between the event and the timestamp *outside the context of the logic thread that is handling the event*. The crucial point is that RTS 25 is meant to establish a clearer sequence of events not only within each thread but also **across threads, across applications, and across institutions**.

For example, suppose that in a given app, receipt of market data updates is timestamped by a thread that reads from a market data feed, while decisions to deal are timestamped by another thread that executes a trading algorithm. (As we've discussed elsewhere, MiFID 2 does not explicitly require you to treat market data updates as reportable events, but if you want to use market data to justify your trading decisions, you'd better record them with RTS 25-compliant timestamps.) Now suppose that the algo decides to deal based on market data update A but suffers a large timestamp-assignment delay when recording that decision to deal. By the time the timestamp is assigned, the market data thread has timestamped updates B, C, and D. To the observer looking at the event log, the state of the market at the time of the decision to deal appears to be D, when in fact it was A.

For another example, suppose an application timestamps the sending of an order. To ensure that the timestamp is not assigned before the message has been sent, the application uses a blocking send call and requests the timestamp after the call returns. But suppose there is a large delay in assigning that timestamp. The event log will say that the event was sent long after it actually was. (In practice, low-latency apps don't usually use blocking sends. But non-blocking sends have a similar issue. The send can be delayed until long after the timestamp.)

The need to consider the broader goal of event reconstruction is the same reason that "stop the world" garbage collection in Java is not some sort of exemption from RTS 25. The only world that stops is the world within the particular JVM instance. The rest of the world keeps on moving.

MYTH #2
Application-level error isn't significant enough to matter.
Look at how small my mean and standard deviation are.

We can establish upper bounds on application-level error by wrapping calls to the subject timestamp method in other timestamps or by calling the method repeatedly without wrapping it. These observations provide an upper bound on timestamp-assignment delay.

The mean of these observations is typically quite small—on the order of tens of nanoseconds. That is over 1000 times smaller than the error allowed by even the most stringent RTS 25 requirement of +/- 100 microseconds. Similarly, the standard deviation, a common way to measure the variability in a statistical distribution, is also the same small order of magnitude.

But many people unjustifiably use the sample mean and standard deviation to claim (for example) that 99% of the application-level errors on a given platform will be within two standard deviations of the mean. This is a misuse and misinterpretation of confidence intervals. Confidence intervals do not predict the probability that a future observation will be a given value; they indicate the strength of a hypothesis that the true mean of the distribution that is being sampled is a given value. And even that knowledge isn't very useful, because mean timestamp error is irrelevant to RTS 25. RTS 25 is all about outliers, not averages. In fact, its wording is specifically about the *maximum* outlier. (How those words will get enforced is one of the mysteries that I discuss later in this blog series.) So what we need is the probability that a given timestamp will be beyond some threshold value. That is, we're concerned with prediction.

In making those predictions, we can't assume distributions are normal (Gaussian), either. Figure 1 is a typical example of such a distribution, displayed in a histogram with vertical and horizontal log scales to enable better scrutiny. This is from one of four test sequences that the STAC-TS.ALE tools run on a system. The system in this case is C++ using the clock_gettime() call on a default installation of Red Hat 6.9 on a server that is about four years old. This kind of system and timestamping method are not uncommon in the wild today.

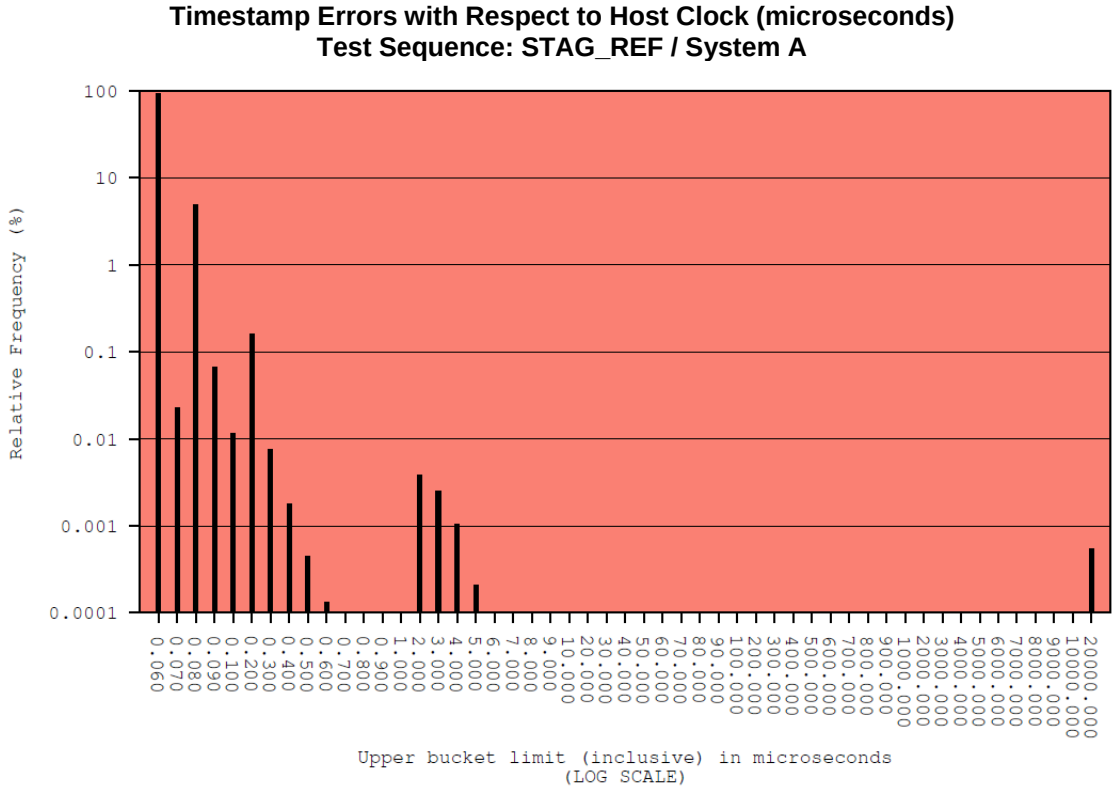

Figure 1

A quick glance makes it obvious that our observed distribution is not normal. It is bounded on the left by zero. Nearly all the observations are in the first bucket, between 50 and 60 nanoseconds (0.050 and 0.060 microseconds). The bar for that bucket stretches nearly to 100% (it is a couple percent shy but looks closer to 100% due to the log scale. And the distribution has a very long, positive tail. In other words, application-level error is characterized by a lot of positive outliers.

Consider Figure 2, which summarizes the measurements of application-level error across all four test sequences run on the system from Figure 1 (System A):

Timestamp Errors with Respect to Host Clock (STAC-TS.ALE1)
Worst Case Error Percentiles (microseconds)
System A

Percentile	Error
Max	21,177.625
99.9999%	10,012.557
99.999%	32.836
99.99%	27.150
99.9%	21.833
99%	1.773
95%	1.640

Figure 2

This table shows several percentiles, all the way up to the maximum (the 100th percentile). Percentiles are a time-honored means of characterizing a distribution's outliers without needing to fit the distribution to some theoretical model that is described by a set of parameters (e.g., Gaussian, Poisson, gamma, etc.). That is to say, percentiles are *non-parametric* statistics. (In a subsequent blog, I'll explain why non-parametric statistics are generally superior to parametric statistics for a case like RTS 25.)

Note that the maximum of 21,177 microseconds is *way* beyond the MiFID 2 tolerance even for non-HFT electronic trading (1 millisecond). And remember that this is just the application-level error. It doesn't include host clock error. Even if the host clock were perfectly synchronized to UTC, some timestamps from this platform would violate RTS 25.

Faced with this kind of reality, some people carry another myth around with them….

MYTH #3
Application-level error is so significant that no application-timestamping solutions can meet MiFID 2 requirements.

Fortunately this isn't true either. Consider the system in Figure 3, below. With a max of 21.3 microseconds, timestamps from this platform will comply with even the most stringent MiFID 2 requirement (100 usecs) as long as the host clock never gets more than 78 microseconds away from UTC (which requires analysis of the time sync chain down to the host clock—again, a subject for another blog).

Timestamp Errors with Respect to Host Clock (STAC-TS.ALE1)
Worst Case Error Percentiles (microseconds)
System B

Percentile	Error
Max	21.297
99.9999%	2.764
99.999%	1.867
99.99%	1.232
99.9%	1.211
99%	0.221
95%	0.175

Figure 3

If all of a firm's systems that are subject to RTS 25 are like System B, then the firm may not have much trouble complying with the regulations. (Though it will still need to demonstrate to regulators that the systems are like System B, for which purpose it may want to use a standard like STAC-TS.)

But a lot of larger firms tell us they believe the event reporting and timestamping requirements of MiFID 2 apply to many systems with application-level error closer to that of System A than System B. (Exactly which systems are in scope depends on high-level MiFID 2 interpretation questions. A mystery I won't try to cover in this blog.) What is such a firm to do? Obviously, upgrading hardware is one option. But if the need is widespread, the hardware bill could be large. This fact can lead to despondency if combined with another myth….

MYTH #4
There's nothing to do about application-level error on a given server. It is what it is, and regulators will just have to accept that.

Once again, this is not true. While it is certainly true that there will be situations where the only options are to upgrade the server hardware or to rely on network capture (a topic for a subsequent blog), a regulated firm that finds that a platform's STAC-TS.ALE benchmark exceeds the firm's application-level error budget may be able to bring the platform into compliance through one or more types of remediation.

Tuning is the least disruptive sort of remediation. To illustrate its potential effectiveness, I will now reveal that System B in Figure 3 is actually the same hardware and software environment as System A but highly tuned by someone who knows what he's doing (in this case, Dom Pigott at Intelligent Compute). This tuning involves applying well-documented OS settings as well as pinning particular processes or threads to particular cores.

Another option is to change the software environment, such as upgrading the operating system. In the case of virtualized languages (Java, C#) or virtualized operating systems (VMware, Xen, etc.), upgrading other components in the software environment (virtual machines or hypervisors) may also help. (All the

examples in this blog are of completely non-virtualized environments, but STAC-TS also handles virtualized environments, another topic that deserves its own blog.)

Still other remediation options involve modifying the application to use a different timestamping method.

However, none of these options is free. Manually tuning systems takes time from experts. And tuning all the systems that are subject to MiFID 2 time compliance—if they are not already being tuned—may require new kinds of cooperation between application teams and corporate IT. Upgrading software environments is also a costly process. And modifying applications may be the most expensive and most risky option, so it is not usually the first recourse for remediation.

Therefore, while it's not true that application-level error is fixed for a given hardware platform, it may be expensive to get that error down to an acceptable level, particularly if the firm is dealing with many such platforms.

The other side of the equation

OK, so let's say you tune and upgrade your platform and even update your applications, yet your application-level error (STAC-TS.ALE) is still outside your error budget. All may not be lost. There's one more potential way to bring a platform into compliance with the rules: *change the rules*.

While this may sound like a Captain Kirk/Kobayashi Maru kind of ruse (like hacking into the ESMA website and changing the RTS document), that's not at all what I mean. What I'm talking about is changing how you interpret the rules. The fact is that ambiguities in the regulations force you and your compliance team to decide how to interpret the rules and to guess how regulators will enforce them. That is, you are faced with a couple of mysteries (actually, more than two, but I'll cover just two in this blog series).

MYSTERY #1
Will regulators be absolute in their focus on maximum divergence, or will they allow exceptions?

RTS 25 specifies a "Maximum divergence from UTC" for different situations, and ESMA has gone to some length to clarify that these are hard limits. Not only did they remove a 99th percentile tolerance from an early draft of the RTS, but they also explicitly rejected an industry request to reinstate a percentile tolerance (including a request from us).

Nevertheless, you will have noticed that the STAC-TS.ALE tables above don't just show the maximum of each sample distribution. They also show percentiles. Why do we bother?

The answer is that the question just won't go away. The tables above show that there is often a *huge* difference between percentiles and the max (the 100th percentile). Engineering for the max versus engineering for two, four, or even six nines can have significantly different costs. For example, suppose our budget for application-timestamp error is 40 microseconds (what's left after allowing, say, 10 microseconds for time distribution infrastructure and 50 microseconds for potential faults that throw my host into holdover for up to 20 minutes before being remedied). If our target is to have the 99.99th percentile below 40 microseconds, the system in Figure 2 is compliant. But if we engineer for the max, we need to take the remediation steps that result in Figure 3.

Furthermore, some regulators have told us that they understand it's not economically feasible for a firm to guarantee absolute compliance with the regulation at all times. They liken the regulation to a speed limit. Speed limits are expressed in absolute terms, but police choose which speeders to pursue and which ones to let off the hook (such as men with wives in labor).

The critical question is what the regulators will consider an exception. Is an exception a “black swan” technical problem that occurs only once after years of perfect compliance? Or is it a completely foreseeable issue that occurs with low frequency every day (such as pauses in OS scheduling)?

For example, if I engineer for compliance at the 99th percentile, will the regulator consider the 1% of my timestamps that violate RTS 25 to be exceptions? If an application is timestamping 100 events per second on average, that means that I will have something like 25,000 events each trading day whose timestamps violate RTS 25. Will regulators view these violations as wives-in-labor type situations? Some of the firms I speak with seem to think so. Others are more skeptical.

In any case, until enforcement becomes clearer (perhaps through the first RTS 25 cases that get enforced) the level of tolerance on which the firm bases its engineering is a choice that the firm cannot avoid. The STAC-TS Working Group does not stake a position on this question. Instead, it is offering the industry tools that provide the data to support a range of decisions.

Whatever tolerance you choose, don’t fall prey to another myth that is circulating….

MYTH #5
If 99.x% of my timestamps fall within RTS 25,
the odds that regulators will discover a timestamp that
is out of compliance is very small.

The myth in this statement is its premise: that regulators will only investigate your RTS 25 compliance if they come across an errant timestamp. That may happen on occasion, such as if a trading firm’s timestamps indicate a different sequence of events than an exchange’s timestamps. More often, however, regulators will assess compliance with RTS 25 by reviewing the processes by which you ensure accuracy. Whenever they ask you to supply records of reportable events to support an investigation, they will also ask for your evidence that the timestamps on those events comply with RTS 25. They will ask themselves: is the firm performing “relevant and proportionate monitoring” and “relevant and proportionate testing”? If so, what do the monitoring results and test results show?

As part of their assessment, regulators will review your application-level error test results. And, assuming they approve your test methodology, they will decide whether you were right to draw the line on compliance where you did.

It’s worth noting that this means regulators will be unable to escape the question of tolerances themselves. As soon as they face a situation in which the firm’s testing records show a platform whose max application-level error makes it possible for max total timestamp error to exceed RTS 25, they will be forced to take a position.

The final mystery of interpretation involves some philosophy and some arithmetic….

MYSTERY #2
Will regulators accept a skew-adjusted approach
to application-timestamp error?

For simplicity, let’s assume for the moment that we only care about maximum divergence. I’ll extend the discussion to percentiles in a moment.

Consider System C, which has the application-level error profile of Figure 4, below. As before, suppose my budget for application-timestamp error is 40 microseconds. Figure 4 shows that the max for this platform is 61 microseconds, so clearly this platform exceeds my budget and requires remediation, right?

Timestamp Errors with Respect to Host Clock (STAC-TS.ALE1)
Worst Case Error Percentiles (microseconds)
System C

Percentile	Error
Max	60.565
99.9999%	30.128
99.999%	21.025
99.99%	6.623
99.9%	1.354
99%	1.050
95%	0.878

Figure 4

Not necessarily. Note that application-level error is always positive (the system can only delay access to the clock; it can't provide a timestamp before it's requested). But recall that the maximum divergence language of RTS 25 allows timestamps to be within some amount *either side* of UTC. (A maximum divergence of 100 microseconds means +/- 100 microseconds.)

Other tables (not shown) in the STAC Report for System C say that its minimum error is 0.047 microseconds—or if rounding to the nearest microsecond, 0 microseconds. So that means that I know my error will always range from a minimum of 0 to a maximum of 61 microseconds. Another way of expressing this is that the error is

30.5 +/- 30.5 microseconds

That means that for this platform, if we subtract 30.5 microseconds from every timestamp (that is, if we treat the range's midpoint, 30.5 microseconds, as timing skew), the resulting error of those timestamps will be

0 +/- 30.5 microseconds

Voila! By treating the midpoint as timing skew, this platform is now within our error budget.

Is this some sleight of hand? Not at all. In this approach, we make a simple tradeoff. By subtracting a fixed amount from future timestamps, we magnify the error of *most* of those future timestamps while ensuring that *all* of them stay within the RTS 25 bounds.

This is absolutely within the letter of RTS 25. Is it consistent with the spirit? I honestly don't know (which is why I consider it a mystery). While it is defensible in practical terms, ESMA may rule that deliberately making some timestamps less accurate than they could be is never acceptable, even if it means ensuring that all timestamps are sufficiently accurate.

I said above that the skew-adjustment approach can apply to percentiles as well as the max. Figure 5, taken from the STAC Report for System C, shows this. In place of percentiles, we use prediction intervals.

Prediction intervals are the equivalent of percentiles applied to ranges within a distribution. They are particularly useful for distributions that have outliers on both the left and right but can also be applied to cases like application-level error, which have positive outliers only. STAC-TS uses prediction intervals to analyze many other benchmarks. I'll explain how we derive them in a subsequent blog, but you can also read about their importance here.

Figure 5 shows the probability (first column) that the timestamp error will fall within the given range (second column) if we subtract the midpoint of the distribution (third column) from all timestamps.

Timestamp Errors with Respect to Host Clock (STAC-TS.ALE1)
Worst Case Error Prediction Intervals (microseconds)
System C

Probability	Application-Level Error (STAC-TS.ALE)	Skew adjustment required for all timestamps
100%	± 30.259	subtract 30.306
99.9999%	± 14.9915	subtract 15.1365
99.999%	± 10.566	subtract 10.711
99.99%	± 3.2465	subtract 3.3915
99.9%	± 0.587	subtract 0.775
99%	± 0.4185	subtract 0.6355
95%	± 0.3295	subtract 0.5515

Figure 5

Epilogue

The STAC-TS Working Group views testing of application-level error as a crucial part of demonstrating ongoing compliance. This is not only because testing is the way of seeing potential potholes in the road ahead (whereas monitoring is a rear-view mirror). It's also because it's not feasible to monitor application-level error. The application-level error in a given timestamp is unknown; otherwise you could correct the timestamp. Moreover, monitoring via sampling is intrusive and yields either too many false negatives or too many false positives. The STAC-TS philosophy is therefore to determine the statistical distribution of application-level error of a given platform via testing under conditions at least as bad as production, and to re-test any time the platform configuration changes in any way.

However, as I mentioned at the outset, application-level error is only part of the total error in application timestamps. Demonstrating that a given platform complies with RTS 25 requires combining STAC-TS.ALE benchmarks with STAC-TS.CE benchmarks that characterize the host clock error in steady state and holdover situations, or using a methodology such as STAC-TS.AVN (application versus network), which measures total application-timestamp error holistically, including both application-level error and host-clock error.

Nevertheless, STAC-TS.ALE can be quite useful on its own as a quick way to rule out particular platforms or identify those needing remediation, without requiring time sync software or any test equipment like capture devices or oscilloscopes.

While low application-level error is not sufficient to comply with MiFID 2, it is necessary.