# Using Time Synchronization to Improve Determinism of Application Response Time in Industrial Control Systems

by Rick Blair
Sr. Principal System Architect
Schneider Electric

Presented at the ODVA
2020 Industry Conference & 20th Annual Meeting
March 4, 2020
Palm Harbor, Florida, USA

## Abstract

In a modern PLC system, one of the key performance factors is application response time (the time it takes a PLC system to set an output based upon a change in an input). The application response time can vary significantly due to the independent components and tasks present in a modern PLC system. Remote I/O systems are ubiquitous and synchronizing these data exchanges with an application can have a dramatic effect on performance. Examples using remote I/O protocols like Modbus scanners and EtherNet/IP implicit messaging will be shown. This paper investigates the use of time synchronization like CIP Sync and IEEE 802.1AS-Rev to synchronize the relevant asynchronous tasks to improve the predictability of application response times.

## Keywords

Application response time, Process image, Modbus®, EtherNet/IP™, CIP Sync™, scan, inputs, outputs, PLC, RPI, client/server, peer-to-peer

## Overview

Today's modern Programmable Logic Controller (PLC) systems have come a long way since they were first introduced in the early 1970s. Microprocessor speeds and memory capacity have dramatically increased, and low-speed fieldbuses have been replaced with high-speed networks based on Ethernet. This has led to dramatic increases in overall system performance. However, with the exception of motion control systems, synchronization between I/O systems and PLC logic execution is not present. This paper demonstrates the benefit of synchronizing these tasks.

## Time Synchronization

In order to synchronize tasks across multiple devices, they must all share the same time basis. This can be in the form of clock time (for example, date and time), sometimes called wall time, or simply an arbitrary counter value. This section describes some of the time synchronization methods available to synchronize time between network devices.

In order to allow devices to send data at the appropriate time, all participating devices must have the same notion of time. There are several Ethernet-based time synchronization protocols to choose from:

- Network Time Protocol (NTP)
- IEEE 1588™ Precision Time Protocol (PTP)
- IEEE 802.1AS-Rev
- CIP Sync

### Network Time Protocol

Network Time Protocol (NTP) is a networking protocol for clock synchronization between computer systems over packet-switched, variable-latency data networks.[1] In operation since before 1985, NTP is one of the oldest Internet protocols in current use.
NTP can usually maintain time to within tens of milliseconds over the public Internet and can achieve better than one millisecond accuracy in local area networks under ideal conditions.
The current protocol is version 4 (NTPv4), which is a proposed standard as documented in RFC5905. It is backward compatible with version 3, specified in RFC1305.

### IEEE 1588 Precision Time Protocol

Precision Time Protocol (PTP) synchronizes clocks throughout a computer network.[2] On a local area network, it achieves clock accuracy in the sub-microsecond range, making it suitable for measurement and control systems.

PTP was originally defined in the IEEE 1588-2002 standard, officially entitled "Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems" and published in 2002. In 2008, IEEE 1588-2008 was released as a revised standard; also known as PTP Version 2. It improves accuracy, precision, and robustness, but is not backward compatible with the original 2002 version.

### IEEE 802.1AS-Rev

IEEE 802.1AS-Rev specifies the protocol and procedures used to ensure that the synchronization requirements are met for time-sensitive applications, such as audio, video, and time-sensitive control, across networks (for example, IEEE 802 and similar media).[4] It specifies the use of IEEE Std 1588 specifications where applicable in the context of IEEE Std 802.1Q. Synchronization to an externally provided timing signal is not part of this standard but is not precluded.

This standard enables stations attached to bridged LANs to meet the respective jitter, wander, and time-synchronization requirements for time-sensitive applications. This includes applications that involve multiple streams delivered to multiple endpoints. This standard leverages the work of the IEEE 1588 Working Group by developing the additional specifications needed to address these requirements.

### CIP Sync

CIP Sync provides increased control coordination needed for control applications where absolute time synchronization is vital to achieve real-time synchronization between distributed intelligent devices and systems.[3] CIP Sync is compliant with the IEEE-1588 standard and allows synchronization accuracy between two devices of fewer than 100 nanoseconds. Real-time synchronization can be achieved over conventional 100Mbps, Ethernet systems with a switch-based architecture.

## PLC Operation

This section describes the basic operation of a modern PLC system. The PLC used for the analysis is a rack-based device consisting of a CPU, a remote I/O communications module, and some local rack-based I/O. The system in Figure 1 is used for discussion. It is understood that many more peripherals exist in most installations, but this simple system should be sufficient to demonstrate the intended principles. Both rack-based and stand-alone I/O peripherals are shown.

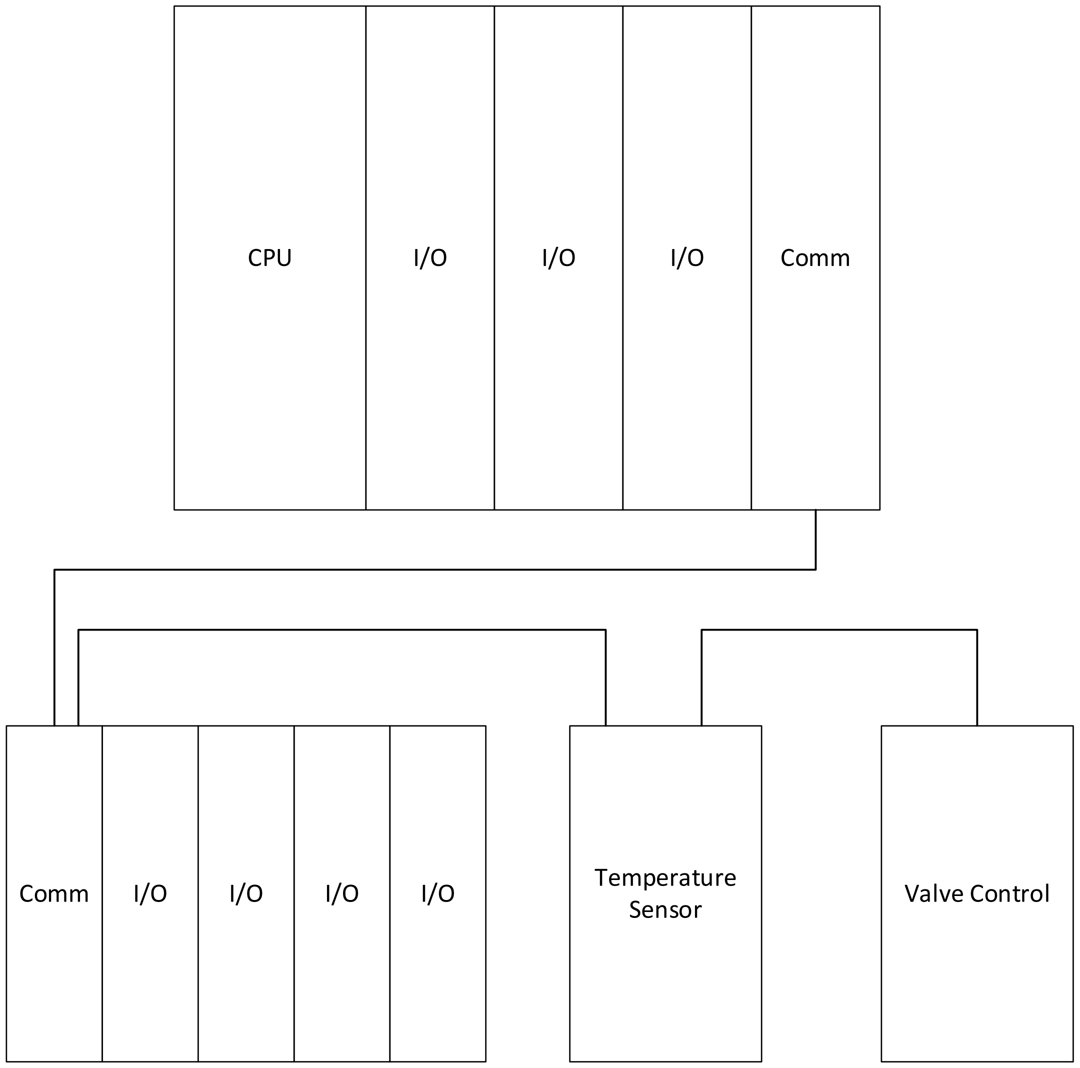

*Figure 1: Sample PLC System*

PLCs perform many tasks in order to control a process. These include reading inputs and writing outputs from a network or local bus, executing user logic, updating I/O process images, and so on. Many of these tasks are performed cyclically but asynchronously, leading to wide variations in application response times.

Within a PLC, I/O data is typically contained in a block of memory called a Process Image (PI). This allows the user program to execute on a constant set of inputs for a cycle. As inputs are read, their contents are placed in the Input PI. The content of the Output PI is used to set output values. Copies of these tables are used during PLC program execution. This allows asynchronous I/O updates to occur during program execution without affecting I/O values operated upon during program execution.

## PLC System Program Execution Cycle

PLCs perform control by executing user-programmed logic on inputs and producing outputs. The typical operation is performed as described:

1. Input data acquisition;
2. Transfer inputs;

3. Update PLC Input Process Image;
4. Execute PLC application;
5. Update output Process Image;
6. Transfer outputs;
7. Apply outputs.

The following sections describe the steps in more detail.

**Input Data Acquisition**
Typical input devices consist of physical inputs and some mechanism for transferring data that represents these inputs to a PLC via some communication mechanism (for example, network, fieldbus, backplane, and so on). In the case of an input device with multiple inputs, a task cyclically gathers the data from the physical inputs and places them into a local process image (PI), and then the communication task cyclically transfers this image over the communication mechanism.

In a stand-alone input module, the communication interface communicates the input process image directly with the PLC. In remote I/O racks, the input data can be transferred to a communications module (called a “head”), which gathers all the inputs from all input modules in the rack and provides an aggregated process image to be transferred to the PLC.

For future calculations, the time to acquire a device’s inputs and place them into an input process image will be represented by the variable $t_{in}$.

**Transfer Inputs**
This operation considers the time needed to transfer the input data from an input process image over a communication medium and to receive it at the PLC’s communications interface. Generally, this is the time the data is on the communication medium. However, for high-speed networks or low-speed processors, the processing time may also need to be considered. Transferring inputs can use a pull model, such as a read operation issued by the communications interface over a backplane or via a protocol like Modbus over a network. A push model can also be used, such as implicit messaging of EtherNet/IP.

For future calculations, the time to transfer a device’s inputs to a PLC’s communication interface will be represented by the variable $t_{inXfr}$.

**Update PLC Input Process Image**
PLCs containing local I/O or a separate remote I/O communications module will need to transfer input data from these devices into the PLC. This cyclic task represents the time needed for a PLC to create a static representation of all the inputs that can be used by the user’s program.

For future calculations, the time to update the PLC’s input process image will be represented by the variable $t_{inPI}$.

**Execute PLC Application**
This task represents the execution of the user’s program. This is a cyclic activity, where each cycle is sometimes referred to as a scan. Each scan begins by making a copy of the input process built in the previous task for its inputs during the entire scan. It also maintains its own image of the outputs, which it copies to the output process image upon completion of each program scan.

While some PLCs provide multi-cycle capability, this paper will focus on a PLC using a single thread of execution for user logic.

User programs typically contain conditional branches (for example, if an input is on, do X; if not, do Y or maybe nothing). Hence, the Execute Logic scan time can vary from cycle to cycle. Some PLC programming tools allow the user to set a constant scan time that is slightly larger than the time needed to execute the user's logic. This will be necessary if synchronization of I/O updates to user logic scan is desired.

For future calculations, the time to execute one scan of the user's program will be represented by the variable $t_{scan}$.

**Update Output Process Image**
This task represents the time needed for a PLC to transfer its output process image to the PLC's communication interface and / or local I/O modules.

For future calculations, the time to update the output process image will be represented by the variable $t_{outPI}$.

**Transfer Outputs**
This operation considers the time needed to transfer the output data from the PLC's communication interface and to receive it at the output device. Generally, this is the time the data is on the communication medium. However, for high-speed networks or low-speed processors, the processing time may also need to be considered. Transferring outputs from the PLC's communication interface typically uses a push model, such as a write operation issued by the PLC over a backplane or via a protocol like Modbus or EtherNet/IP over a network.

For future calculations, the time to transfer a PLC's outputs to an output device will be represented by the variable $t_{outXfr}$.

**Apply Outputs**
Typical output devices consist of physical outputs and some communication mechanism for transferring data that represents these outputs from a PLC via some communication mechanism (for example, network, fieldbus, backplane, and so on). In the case of an output device with multiple outputs, the communication task cyclically receives outputs and places them into a local output process image. Another task takes data from the local output process image and writes it to the physical outputs.

In a stand-alone output module, the communication interface receives the output data directly from the PLC. In remote I/O racks, the output data can be transferred to a communications module (called a "head") that gathers all outputs from the PLC and transfers the appropriate outputs to the respective output modules in the rack.

For future calculations, the time to receive a PLC's outputs and write them to the physical outputs will be represented by the variable $t_{out}$.

## I/O Cycle
PLC I/O can be local or remote. Local I/O is present on the same bus or backplane as the PLC processor. Remote I/O is connected to a PLC using a network, such as a fieldbus or Ethernet protocol, like EtherNet/IP or Modbus TCP/IP. Local I/O typically has shorter update latencies, but as network speeds continue to improve, the differences in latencies continue to shrink.

Input and output values located on a network can be exchanged between a PLC and remote I/O devices using two methods.

- Client / Server (CS), like Modbus TCP/IP
- Publish / Subscribe (PubSub), like implicit EtherNet/IP messaging

In the CS model, the PLC initiates a request for data and the input device responds with values. Synchronization of this task to PLC program execution is realizable by the PLC. In the PubSub model, the PLC cyclically sends outputs (which can be synchronized with program execution) while the input device continuously sends input data to the PLC at a predetermined interval (or upon change). In this case, the input data is sent asynchronous to a PLC scan cycle, causing variations in application response time.

### Remote I/O Cycle

A remote I/O device is responsible for receiving output values and applying them to outputs and taking input data and sending it to a remote device. As stated earlier, it can receive a request for inputs and subsequently send a response or it can periodically send its input values.

Whether the remote device is a single I/O device or a head device on a rack of I/O devices, it will take received output data and apply it to its outputs. It will also gather all its inputs and place them in input messages. Typically, a Remote I/O device will contain a PI and have asynchronous tasks processing input and output messages and a separate set of asynchronous tasks reading inputs and writing outputs. However, this implementation can add variances to application response time and can be improved using time synchronization.

## Application Response Time

As stated earlier, application response time is defined as the time from when a physical input changes to when a physical output change occurs as a result of that input change.

With the exception of motion control applications, many PLC systems perform the tasks described in the previous sections asynchronously to each other and the application response time can vary greatly during execution of a user’s program. The previously defined variables represented the times needed to complete a certain task. These tasks are run cyclically, so some new variables are introduced to define the cycle times at which these tasks are executed. In most modern PLCs, these tasks are run at some repetitive rate. For example, on PLCs using the EtherNet/IP protocol, the transfer inputs and transfer output tasks are run at a Requested Packet Interval (RPI) that is loosely coordinated with the application scan time. Table 1shows these new variables.

*Table 1: Variables Use to Calculate Application Response Time*

| Task | Execution Time | Cyclic Time |
|---|---|---|
| Input data acquisition | $t_{in}$ | $t_{inCyc}$ |
| Transfer inputs | $t_{inXfr}$ | $t_{inXfrCyc}$ |
| Update PLC Input Process Image | $t_{inPI}$ | $t_{inPICyc}$ |
| Execute PLC application | $t_{scan}$ | $t_{scan}$ |
| Update output Process Image | $t_{outPI}$ | $t_{outPICyc}$ |
| Transfer output Process Image | $t_{outXfr}$ | $t_{outXfrCyc}$ |
| Apply Outputs | $t_{out}$ | $t_{outCyc}$ |

The worst-case application response time occurs when a task is preparing data for a dependent task but does not complete before the dependent task begins. Cascading this effect among all seven tasks creates the worst-case application response time. Calculating this worst-case time when the various tasks are unsynchronized is quite complicated. There are dependencies relative to the cyclic times of the various tasks. This paper makes the following assumptions to demonstrate the principle:

- Application scan time is longer than any other task’s execution or cycle time.
- Task execution times are less than their respective cycle times.
- The amount of misalignment (overlap) that is needed for a subsequent task to miss the data provided by a dependent task is based upon task design. To demonstrate worst-case behavior, it is assumed that this value is negligible and is ignored.

Figure 2 demonstrates worst-case timing for asynchronous cyclic task execution. Narrow rectangles represent cycle times while thicker rectangles represent task execution times. Shaded tasks show where the data that represents the changed input or output first appears in the task’s data.

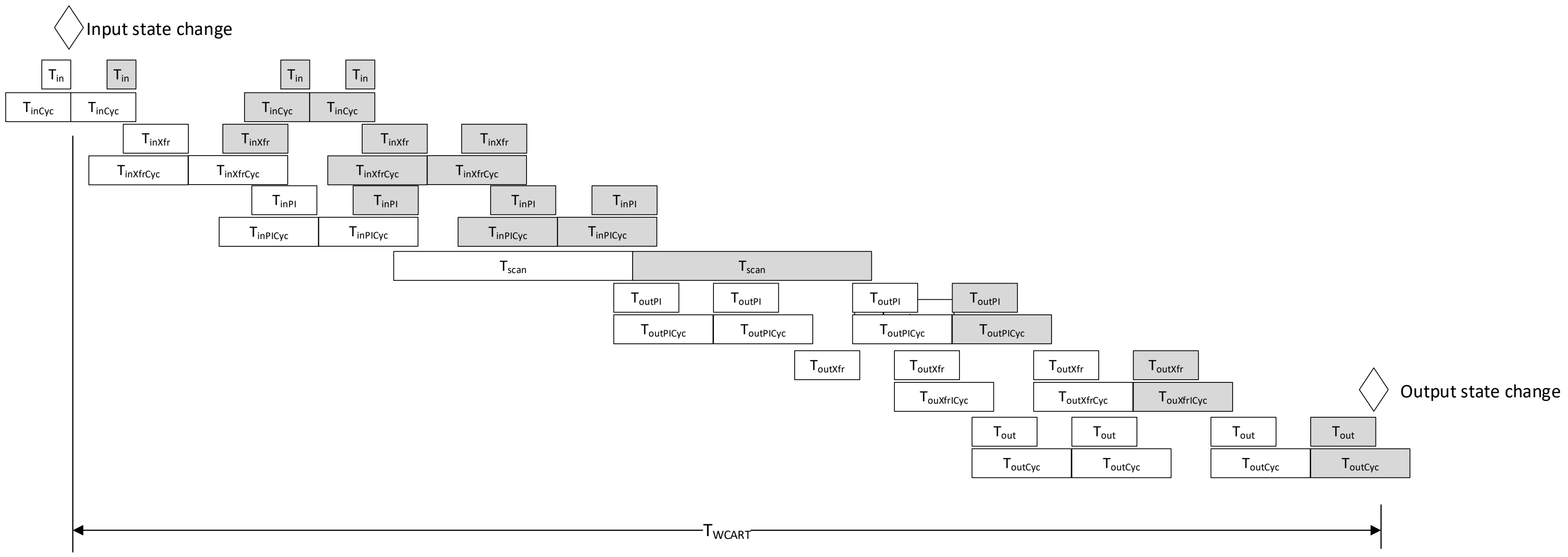

*Figure 2: Worst-case Application Response Time ($T_{WCART}$)*

The following formula represents the worst-case application response time based on the above assumptions:

$t_{WCART} = t_{inCyc} + t_{inXfr} + t_{inXfrCyc} + t_{inPI} + t_{inPICyc} + 2 * t_{Scan} + t_{outPICyc} + t_{outPI} + t_{outXfrCyc} + t_{outXfr} + t_{outCyc} + t_{out}$

Now let’s look at optimum conditions when all these tasks are synchronized to each other. Because the state change of a physical input cannot be synchronized with any of these tasks, two scenarios arise. First, when an input arrives just prior to the execution of the input data acquisition task and second, when the input arrives just after completion of the input data acquisition task. This leads to the following two formulas for application response time:

$T_{minART} = t_{in} + t_{inXfr} + t_{inPI} + t_{scan} + t_{outPI} + t_{outXfr} + t_{out}$

$T_{maxART} = t_{inXfr} + t_{inPI} + 2 * t_{scan} + t_{outPI} + t_{outXfr} + t_{out}$

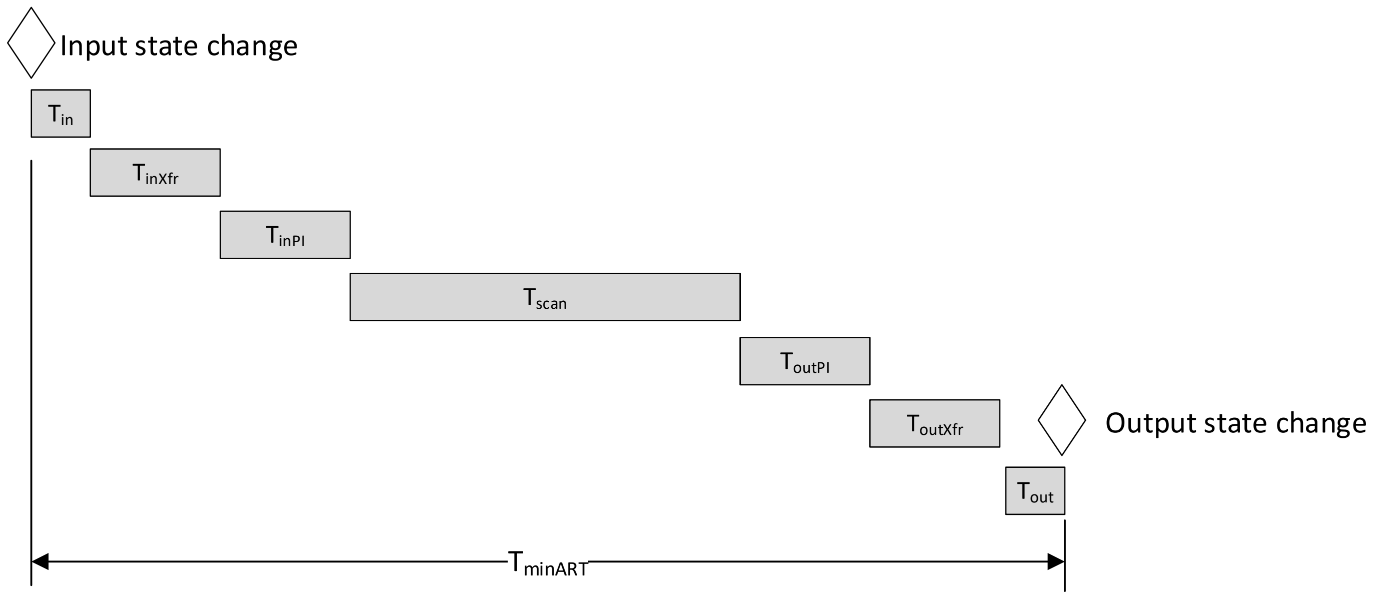

*Figure 3: Minimum Application Response Time ($T_{minART}$)*

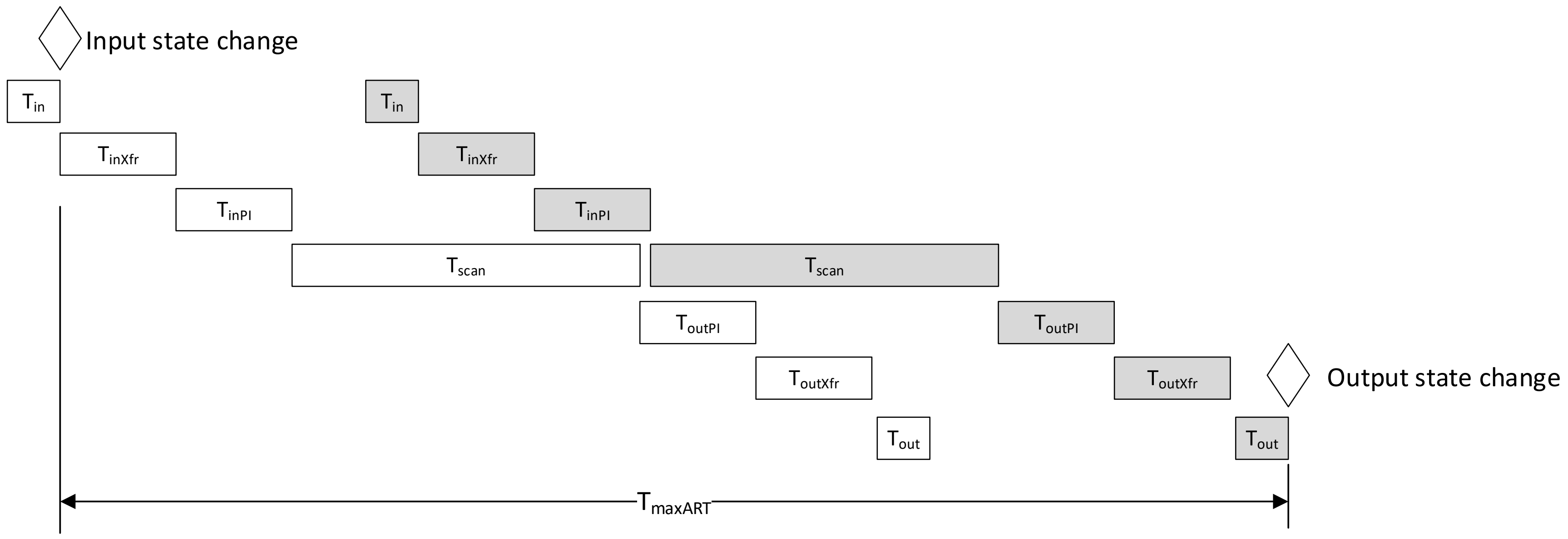

*Figure 4: Maximum Application Response Time ($T_{maxART}$).*

As shown by Figure 3 and Figure 4 and the equations for $T_{minART}$ and $T_{maxART}$, application response times can vary even when all tasks are ideally synchronized, because an input can change state at any given time.

Now, let’s look at the various application response times using some typical values. Table 2denotes some typical values for these variables and the calculation results follow. All times are in milliseconds.

*Table 2: Typical values for Application Response Variables*

| Task | Execution Time | Cyclic Time |
|---|---|---|
| Input data acquisition | 0.1 | 1 |
| Transfer inputs | 0.5 | 20 |
| Update PLC Input Process Image | 0.5 | 20 |
| Execute PLC application | 50 | 50 |
| Update output Process Image | 0.5 | 20 |
| Transfer output Process Image | 0.5 | 20 |
| Write outputs | 0.1 | 1 |

$T_{minART}$ = 52.4 ms
$T_{maxART}$ = 102.1 ms
$T_{WCART}$ = 184.1 ms

One can see that the cycle times of the transfer input and output tasks and updating the input and output PIs has a large influence on the worst-case application response time. One might be tempted to simply increase the frequency of these tasks. However, this results in more load on CPUs and network bandwidth utilization. Conversely, decreasing the frequency of these tasks will lower CPU usage and network bandwidth utilization at the cost of a larger worst-case application response time.

## Implementation Considerations

So far, this paper has shown how synchronizing I/O tasks with application scanning can improve application response times but has offered no guidance on implementation. This section offers some implementation guidance. However, internal PLC architectures vary from vendor to vendor, so its applicability may require some interpretation.

### Time Synchronization

Time synchronization is the first step needed to synchronize tasks across multiple devices. There are various choices, as shown earlier. The main factor for choosing a specific time synchronization protocol is the needed accuracy.

Time synchronization provides each device on the network the same time reference so that they can trigger task executions at the appropriate times in order to complete their tasks just prior to a dependent task’s start of execution.

### Scheduling

Once all devices have the same time, tasks can be scheduled to complete just prior to a dependent task’s start of execution. However, many of today’s devices operating systems provide some sort of multi-tasking, so tasks can be interrupted. Scheduling needs to take this into account, either by padding start times or dynamically measuring and adjusting start time based upon data. Since execution times may vary by configuration, dynamic adjustment is preferred.

Similar situations exist on network traffic. I/O traffic could be interrupted by other traffic on the network. Again, padding start times to account for interruptions could be one solution. Another promising solution on the horizon is Time Sensitive Networking (TSN), which offers scheduling uninterruptable Ethernet traffic.

## Conclusions

This paper demonstrated the effects of synchronizing I/O tasks with application execution, resulting in improved application response times. Formulas were provided for calculating min and max times when fully synchronized as well as for a worst-case calculation when tasks are not synchronized. Implementation methods were also considered.

In an ever-changing world where vendors are striving to eke out every bit of performance from their PLC systems, this may be another method to add to the list of potential future upgrades.

## References

1. NTP
https://en.wikipedia.org/wiki/Network_Time_Protocol

2. PTP
https://en.wikipedia.org/wiki/Precision_Time_Protocol

3. CIP Sync
https://www.odva.org/Technology-Standards/Common-Industrial-Protocol-CIP/CIP-Sync

4. AS-Rev
https://1.ieee802.org/tsn/802-1as-rev/