Evaluating TSN Preemption for Cyclic EtherNet/IP™ Communication

Rick Blair
Senior Principal Network System Architect
Schneider Electric

Presented at the ODVA
2022 Industry Conference & 21st Annual Meeting
March 9, 2022
San Diego, California, USA

Abstract

Industrial Control systems use Ethernet-based protocols such as EtherNet/IP and Modbus TCP for cyclic communication to exchange process data, such as inputs, outputs, and motion commands and feedback. One limitation of Ethernet is its lack of determinism. For certain applications, like motion control, this has led to the development of protocols like SERCOS and EtherCAT, which rely on specialized hardware solutions.

Recently, Ethernet specifications have evolved to include Time Sensitive Networking (TSN) which defines features like a Time Aware Shaper (TAS) to achieve predictable transport of Ethernet packets. However, not all cyclic communication used in industrial control systems need the determinism offered by TAS, but do require a bounded packet latency. TSN defines a feature called preemption that can address this use case.

This paper will provide an overview of TSN preemption and will analyze its use in Ethernet-based industrial control communication. A simulator, written in the Python programming language, is described, allowing for analysis of varying topologies and cyclic rates. Interfering traffic is introduced to demonstrate the effectiveness of preemption to limit the disturbances.

Keywords

Time Sensitive Networking (TSN), preemption, latency, simulation, Python, Quality of Service (QoS), interference, store-and-forward switching, cut-through switching.

Introduction

This paper introduces an Ethernet switch simulator written in the Python programming language. A brief overview of how Ethernet bridges (switches) operate is presented first so the reader can better understand the simulator description that follows. The high-level information of the simulator includes many Python class descriptions for simulating the behaviors of devices like switches, endpoints, and cables. Monitors are also described, allowing capture of egressing packets and analyzing bandwidth utilization.

A sample system is described (which uses store-and-forward switching, 100 MB/sec network speed, and two express traffic queues) and programmed into the simulator. Two types of devices (with differing payload sizes, packet priorities, and cyclic update rates) compose this system, sending their packets to a controller device. Simulator output is presented, both with and without interfering traffic and bandwidth analysis.

Finally, some observations are presented.

Ethernet Switch Behavior

This section provides a brief summary of how today's modern Ethernet bridges (switches) operate.

Overview

Ethernet bridging was initially defined in the 802.1D IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges. Ethernet bridges (switches) relay Ethernet frames (packets) between devices connected to switches. This is accomplished by copying Ethernet packets from one switch port to another, based on the MAC addresses in the packets.
A switch looks up the destination address contained in a packet in its forwarding, or switching, table to determine the outgoing port and egresses the packet out that port toward its intended destination.

QoS – Strict Priority

Strict priority is a queuing mechanism for Ethernet switches. It is defined in 802.1Q-2018 Clause 8.6.8.1. In strict priority queueing, the queue with the highest number (class) has priority over the remaining queues. When multiple Ethernet frames are queued on an interface for transmission, the queue with the highest priority having an Ethernet frame ready for transmission will transmit. Ethernet frames in lower priority queues are held until the priority of their queue becomes the highest queue with a ready Ethernet frame.
TSN offers transmission selection (shaping) algorithms that can be bound to higher priority queues which determine if such a queue is served before a lower priority queue.

Store-and-Forward Switching

Store-and-forward switching ensures a high level of error-free network traffic. This is because erroneous frames are not forwarded across the network. With store-and-forward switching, the switch copies each complete frame into the switch memory buffers prior to forwarding. The CRC portion of the frame is used to verify the integrity of the received frame. If an error is detected (bad CRC, too short or too long), the frame is discarded. If the frame is error free, the switch queues the frame for forwarding out the appropriate interface port.

Because an Ethernet switch, when using store-and-forward switching, must store an entire frame prior to forwarding, frame latency is dependent upon frame size. This is exacerbated when using large frames and daisy-chained architectures typically found in industrial control system designs.

Cut-through Switching

Cut-through switching is where a switch starts forwarding a frame before the whole frame has been received. Compared to store-and-forward switching, cut-through switching can offer lower latency but, because the frame check sequence appears at the end of a frame, the switch is not able to verify frame integrity before forwarding it. Cut-through switching will forward corrupted frames, whereas store-and-forward switching will discard them.
Pure cut-through switching is only possible when the speed of the outgoing interface is equal to or lower than the incoming interface speed.
A switch may buffer (acting in a store-and-forward manner) a frame instead of using cut-through under certain conditions:

- Speed: When the outgoing port is faster than the incoming port, the switch must buffer the entire frame received from the lower-speed port before the switch can start transmitting that frame out the high-speed port, to prevent underrun. (When the outgoing port is slower than the incoming

port, the switch can perform cut-through switching and start transmitting that frame before it is entirely received, although it must still buffer some of the frame).

- Congestion: When a cut-through switch decides that a frame from one incoming port needs to go out through an outgoing port, but that outgoing port is already busy sending a frame from a second incoming port, the switch must buffer some or all of the frame from the first incoming port.

Frame Preemption and Interspersing Express Traffic

The purpose of preemption is to provide reduced latency transmission for time-critical frames in an Ethernet switch. Evaluation of preemption is based on the definitions in 802.1Q-2018 Clause 6.7.2 and 802.3-2018 Section 7 Clause 99 with “express” and “preemptible” configurable in the switches for all queues on a per queue basis.
Frame Preemption specifies procedures, managed objects, and protocol extensions that define a class of service for time-critical frames that requests the transmitter in a switched Local Area Network to suspend the transmission of a non-time-critical frame and allows for one or more time-critical frames to be transmitted. When the time-critical frames have been transmitted, the transmission of the preempted frame is resumed. A non-time-critical frame could be preempted multiple times.

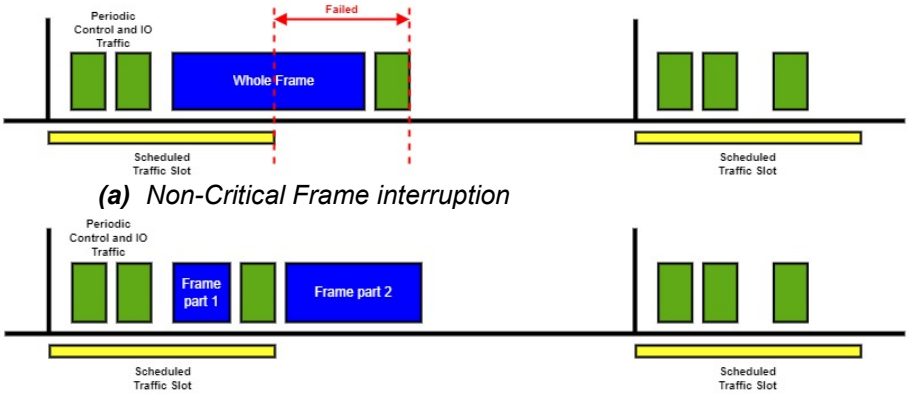

Figure 1: Frame Preemption and Interspersing Express Traffic

Figure 1-(a) shows that a large, non-time-critical frame (in blue) may start transmission ahead of the desired transmission time of time-critical frame (in green). This condition leads to excessive latency for the time-critical frame. Transmission preemption preempts the non-time-critical frame to allow the time-critical frames to be transmitted as shown in **Figure 1-(b)**. This provides the capabilities of an application that uses scheduled frame transmission to implement a real-time control network.

Preemption Examples

One or more queues related to an interface port in an Ethernet switch can be marked as express. This allows packets from these queues to preempt packets from non-express queues that are currently being egressed. However, this is limited to queues that are both lower priority and are not also marked as express.
Current Ethernet standards only allow one preemption on a port to be active at a time. So, this is unlike preemption in task scheduling of operating systems, where it is standard to allow multi-level preemption by higher priority tasks. If a packet preempts a non-express packet, the new packet cannot be preempted. Following is an example with multiple express queues:

- QueueA with priority p4, express
- QueueB with priority p3, express
- QueueC with priority p2, non-express
- QueueD with priority p1, non-express

Higher values of “px” are higher priority.
QueueA and queueB are marked as express. This would allow packets of priority p4 or p3 to preempt packets with priority p2 or p1. However, packets with priority p4 cannot preempt packets with priority p3. in case both queueA and queueB are ready for transmission at the same time, the packet in queueA would be selected based on priority. If the transmission of a packet from queueB has already started, packets from queueA need to wait for that transmission to complete.
As a result, the latency for packets from queueA is increased.

Preemption and Packet sizes

During Preemption, a switch splits an egressing packet into two fragments. the preemption algorithm adds a four octet Cyclic Redundancy Check (CRC) to the currently egressing fragment and adjusts the CRC in the remaining fragment to account for the reduced packet content. Preemption does not provide a padding of fragments to meet minimum Ethernet packet sizes (64 octets). Hence, a packet can be fragmented after 60 octets as long as there are 64 octets remaining. Thus, a packet must be at least 124 octets in length in order to be preemptable. Any packet of 123 octets or less cannot be preempted.

Preemption and Cut-through Switching

When using store-and-forward switching, the preemption algorithm knows the size of an egressing packet and can determine if sufficient octets remain in order to make a preemption decision. However, when using cut-through switching, the switch is egressing a packet as it is ingressing and the switch has no knowledge of the number of remaining octets. One workaround is to not cut-through a packet until at least 64 octets have been received. This diminishes the benefit of using cut-through switching. Another solution adopted by many switch suppliers is to only allow cut-through switching for express packets and use store-and-forward switching for non-express packets. This second approach is what is currently implemented in the Ethernet Switch Simulator.

Physical Interface

Each external port of an Ethernet switch contains a physical layer interface (PHY) consisting of electronic circuit transmission technologies. It can be implemented using various hardware technologies with widely varying characteristics. It performs encoding, transmission, reception, decoding and provides galvanic isolation.
A PHY translates logical communications into hardware-specific operations to cause transmission or reception of electronic (or other) signals. This conversion increases packet latencies and may be different for a transmitter compared to a receiver. Since both transmission and reception are necessary, the Ethernet Switch Simulator uses an average value for this delay.

Interference

This section considers the effects of packets that interfere with another packet as it travels to its intended destination. Only store-and-forward switching scenarios are considered.
The effects of store-and-forward switching are shown in **Figure 2**. A line topology is used for this demonstration. Each row in the figure represents an ingress or egress time of a packet to or from a particular device or switch in the line. A packet originator (or talker, in TSN terms) and three switches plus an end-device (which is the packet destination or target and listener in TSN terms) are shown.

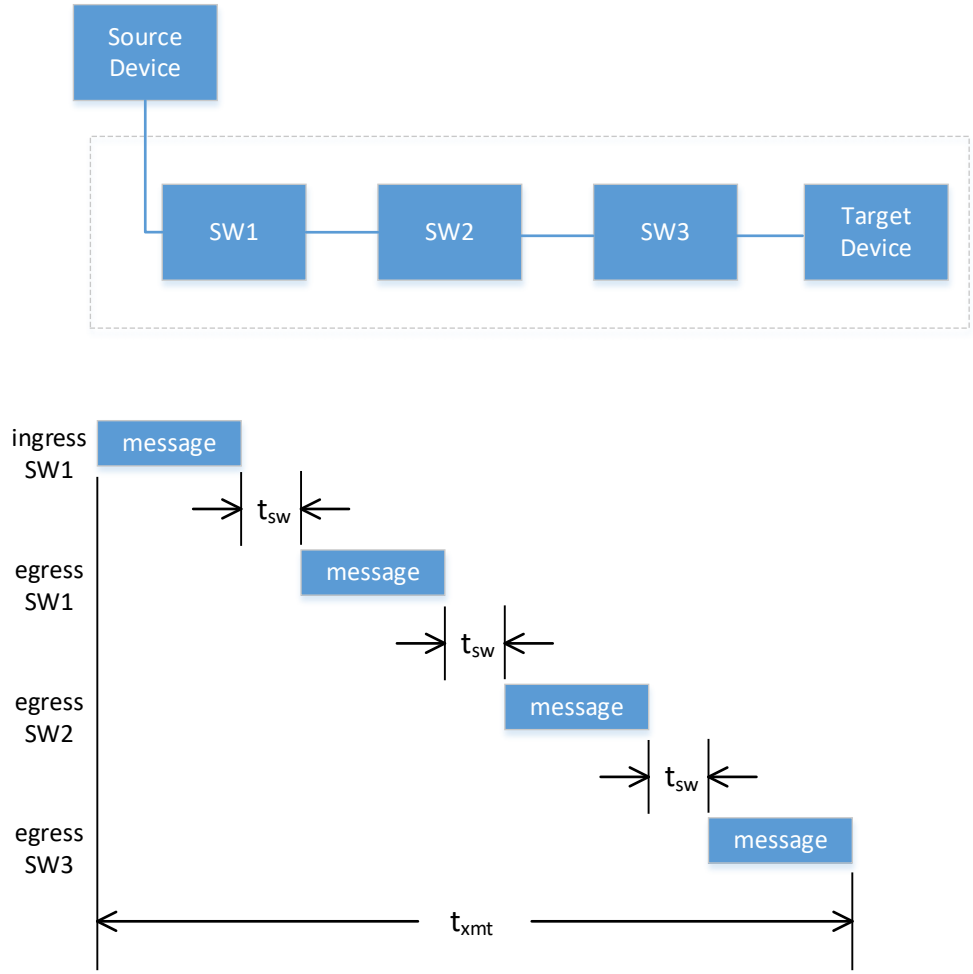

Figure 2: Store-and-Forward Packet Timing

Since the format of **Figure 2** is used throughout this paper to represent packet timings, here is a brief explanation. The top of the figure represents the topology of the network being analyzed. On the left, labels describe whether the packets are ingress to or egress from a particular port on a specific switch. Rectangles, with packet labels inside, represent packet times, with time flowing from left to right. Packet labels like HP_0, HP_1, and so on, or 'high priority' represent high priority packets while terms like int_0, int_1, and so on, or 'low priority' are used to represent lower priority packets. A dimension line shows the total latency (t_{xmt}) for the packet of interest. When two switches' ports are directly connected (for example, P2 of SW1 connected to P1 of SW2) egress from one port is equivalent to the ingress of the other. Wire speed times are not considered in any diagrams. Switches take time to calculate which packet should be forwarded next, represented by t_{sw}. The timing of t_{sw} is from the completion of a packet ingress to the start of a packet egress or, in the case of cut-through, after the requisite data has been received. To demonstrate maximum interference, all interfering packets in this document's figures are depicted at the latest possible arrival time to still cause interference.

Interfering traffic can occur when a switch is already transmitting a packet out a particular port and another packet, destined for the same port, arrives. This newer packet must wait for the completion of the already in-progress packet before it can be sent, as shown in **Figure 3**. This form of interference can happen regardless of packet priorities.

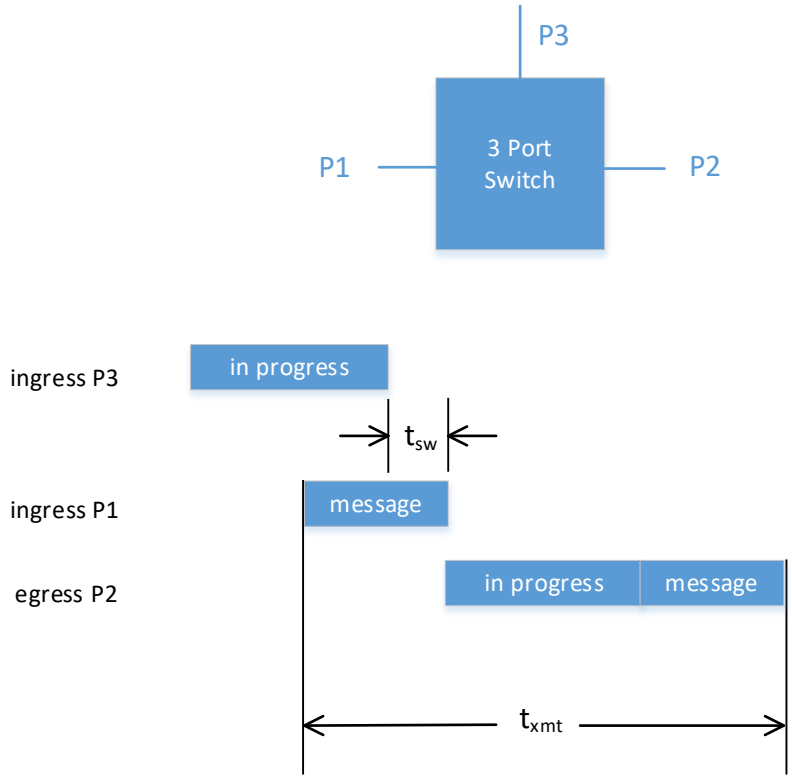

Figure 3: Interference by Packet Already In Progress

Another form of interference is when a switch is currently egressing a packet and it receives a high priority packet, which it queues, and a lower priority packet, which it also queues. Upon completion of egressing the initial packet, the switch chooses to send the higher priority packet first, thus delaying the lower priority packet, as shown in **Figure 4**.

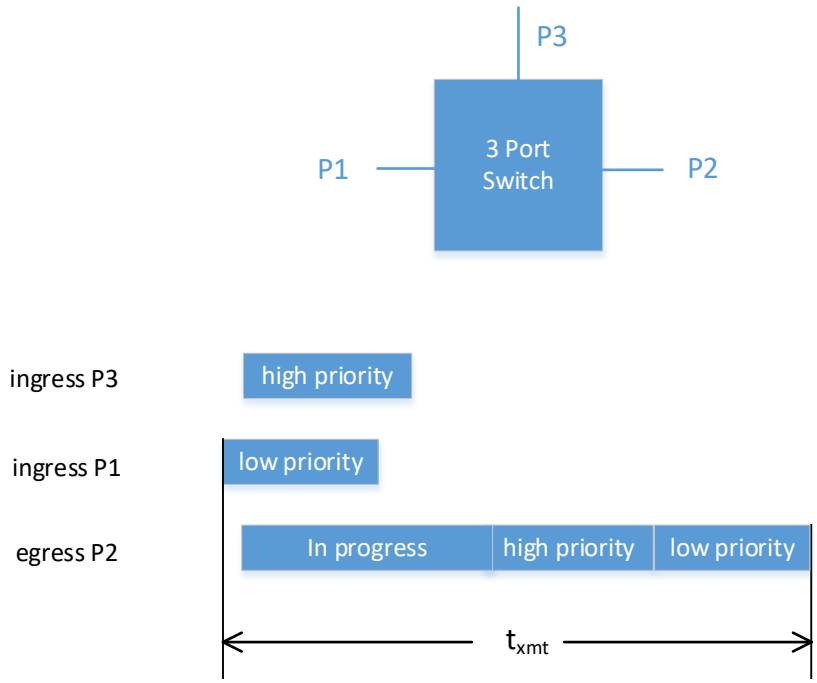

Figure 4: Interference by Higher Priority Packet

The above examples demonstrate how a packet travelling through a switch can get delayed due to interference. In order to determine maximum packet latencies, the time when packets arrive (ingress) and packet priorities must be taken into account.
In-progress interference occurs when a packet destined for a particular port arrives after the switch has already decided to send another packet out that same port. Packet priority has no effect in this case, because the decision was made prior to the arrival of a potentially higher priority packet. Assume a high-priority packet is arriving on port 1 destined for port 2 and a lower priority interfering packet is arriving from port 0 or has already been queued or is currently being transmitted out of P2.
The worst-case scenario occurs when the interfering packet is queued just prior to the switch deciding which packet to send next and the high priority packet is queued just after the switch makes this decision. In this case, the lower priority packet will be sent prior to the high priority packet. This will cause a delay of the high priority packet equivalent to the amount of time remaining to send a packet already in progress, which could be the time for the entire packet in the case of starting transmission just before the arrival (queuing) of the high-priority packet.
Without preemption, the full packet transmission time of the lower priority packet will impact the transmission of the high-priority packet. However, if the high-priority packet is express traffic and preemption criteria is met for the non-express lower-priority packet, preemption can occur and only the initial fragment of the non-express packet will impact the transmission of the express packet. The worst-case delay occurs with the maximum non-preemptable size packet, which is 123 octets.
The latency introduced by a 123-octet packet that a switch just started to transmit, or committed to transmit, is the maximum time an output port can be blocked. A 124-octet packet could be split into two fragments after 60 octets. Any larger packet could also be split into two fragments after 60 octets.

Ethernet Switch Simulator Behavior

There are numerous Ethernet switch simulation environments available on the market, but, for various reasons (not described in this paper), the author decided to develop one from scratch using the Python programming language. The initial goal was to focus on predicting maximum latencies for various architectures and types of cyclic traffic when preemption is used. Express queues can be configured to support multiple cyclic traffic scenarios (such as motion and I/O).
The Ethernet Switch Simulator (ESS) attempts to simulate rudimentary Ethernet switch, end device, and cable behavior in a network of switches linked together.

Current Feature Level

Currently, the ESS implements:

- Store-and-forward and cut-through switching
- Programmable port speeds
- Programmable number of ports per switch
- Eight priority queues
- Preemption
- Multiple express traffic queues
- Unicast endpoint addressing
- Bandwidth utilization measurements

The current version of the ESS does not support:

- Redundant connections of any kind (implemented but not yet verified)
- Multicast addressing
- Non perfect internal connections of a switch (assumption is that a switch can process wire speed on all ports simultaneously)
- Time aware shaping
- Time variances between EthernetEndpoints

The Ethernet Switch Simulator allows modeling applications that consist of endpoints, switches, and cables connected together to form a network structure. Traffic patterns are preloaded into endpoints prior to running the simulation. The main architecture consists of several Python classes, as briefly described in **Table 1**. More detailed information is provided in later sections.

Table 1: Ethernet Switch Simulator Classes

Class	Short Description
EthernetSwitch	The EthernetSwitch class simulates basic switch behavior, including MAC, queues, and switch firmware execution. It does not consider external circuitry like physical interfaces (PHYs).
EthernetEndpoint	The EthernetEndpoint class is the source and destination of all packets. It can be used to represent devices like controllers, drives, I/O devices, and so on. Packets are queued (For time-based transmission) in EthernetEndpoints prior to simulation.
EthernetCable	The EthernetCable class emulates the behavior of PHYs and cables, adding a packet delay based on PHY parameters and cable length. While transmit and receive PHY delays are not the same in the real world, the EthernetCable class assumes a PHY on each end and uses a single variable that is the average of the transmit and receive PHY delays.
EthernetPort	The EthernetPort object is built into the EthernetEndpoint and EthernetSwitch classes, and eventually, the EthernetCable class. It is used to manage ingress and egress of packets, along with truncating packets when preempted. While a cable in the real world does not contain an actual port, the EthernetPort class is used to separate the ingress and egress of a packet, because they occur at different times, based on the programmed delay.
EgressMonitor	(For time-based transmission) The EgressMonitor class allows choosing which ports in the system should be used to monitor egress traffic during the simulation, similar to a WireShark capture, which can be used to analyze the results upon completion of the simulation.

Instances of the EthernetEndpoint, EthernetSwitch, and EthernetCable classes can be connected together in the main program to form a network topology. The main program also provides the capability to define packets and packet patterns, choosing data to monitor, and calculating bandwidth utilization for selected ports.

Simulation Setup

The main program is where the system and the network traffic to simulate is defined. The current version supports multiple device types (like controller, drive, I/O rack, and so on). In addition, safety traffic can be simulated as well. The cyclic rate to be simulated can be defined per traffic type and per device (for example, 1 msec for motion traffic and 20 msec for safety traffic). Additional traffic patterns could be added to a device type to support even more cyclic exchanges for a single device type. By having multiple device types, situations comprising multiple cyclic rates can be analyzed. A Python dictionary is used to store the desired packet information and the number of devices of each type in the system. A single value can be used for the size of packets sent and received by a particular device type or different sizes can be chosen for each device of a particular type. An offset of when a packet should be sent within a cycle is also available. This value can be static or influenced using a random number generator to simulate initial packet egress variances (jitter) in an application.
Next is to connect the various objects together to represent a system consisting of end devices, switches, and cables. In today's industrial control environment, devices with two external switched ports (enabling easy daisy-chaining of devices) are quite common. These devices, referred to as switched endpoints, are

constructed using a three-port switch, two that are exposed to outside connections while the third is typically connected to the Central Processing Unit (CPU) of the device. **Figure 5** demonstrates this architecture and will be used in a simulation example.

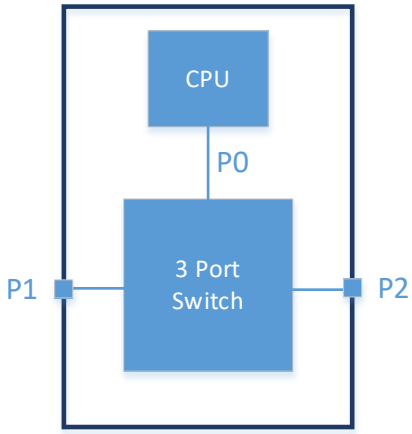

Figure 5: Three Port Switch Diagram

To create a switched endpoint, The ESS allows connecting an EthernetEndpoint directly to an EthernetSwitch without an intervening EthernetCable. This is one of the reasons the PHY delay is simulated in the EthernetCable class and not the EthernetSwitch class. While the current version of the ESS only simulates a standard Ethernet interface for this type of connection, enhancements could be made to support other EthernetEndpoint to EthernetSwitch interfaces. To connect one EthernetSwitch to another, an EthernetCable object is used.
The ESS provides several types of EgressMonitors (for example, data capture and bandwidth analysis). The next step is to decide what egress data should be captured and assign an EgressMonitor to the desired ports on an EthernetSwitch or EthernetEndpoint.
The last step before running the simulation is to preload the EthernetEndpoints with the desired packet patterns defined earlier.

Simulation

The Ethernet switch simulator works on the principle of time slicing. The variable simulationTime allows choosing the simulation duration (in nanoseconds). The variable simulationStep is used to choose the increment of time (default is 20 nanoseconds). At each time increment, the main program calls the process method in each of the aforementioned objects. This method, which is unique to each object type, analyzes packet information properties to decide if an action is required, and if so, applies that particular action. For example, in the EthernetCable object, a packet must not start to egress until the 2 PHY delays and the cable delay have been reached.

Ethernet Packet Classes

The main purpose of the ESS is to simulate the movement of Ethernet packets from sources to destinations, tracking salient information (like latency). Output from multiple runs of the simulator (with varying interfering packets and with preemption on or off) can be compared to evaluate the influence of interference.
The ESS uses instances of the EthernetPacketState class to track the movement of Ethernet packets from their source EthernetEndpoint to their destination EthernetEndpoint. There are two components to an Ethernet packet (dynamic and static). The dynamic portion is stored in an instance of the EthernetPacketstate class (described later) and contains information that can vary (such as state, packetTime, source and destination ports, to name a few) as an Ethernet packet moves within and between instances of the EthernetSwitch, EthernetCable, and EthernetEndpoint classes. The static

portion is stored in an instance of the EthernetPacket class (described later) and contains information that remains constant (such as size, priority, source, and destination addresses, to name a few).

EthernetPacket Class

The EthernetPacket class contains information that remains static (except for preemption) as a packet moves through the simulated network. In case a packet is preempted, the existing EthernetPacket instance is changed to reflect the new size of the initial fragment and a new instance is created to contain the remaining fragment. **Table 2** defines various properties of the EthernetPacket class.

Table 2: Major properties of EthernetPacketClass

Property	Description
Name	Unique name of packet. Useful for analyzing results.
Size	Size of packet, in octets. Includes Ethernet header, IP header, payload, and CRC. Does not include preamble, Start of Header, or Interframe gap.
Priority	Packet priority, 0 being lowest and 7 being highest.
SrcAddr	String containing address of source EthernetEndpoint instance.
DestAddr	String containing address of destination EthernetEndpoint instance. Used by EthernetSwitch instance to decide egress port of packet.
CreationTime	Emulates time at which an endpoint would queue a packet to the Ethernet fabric. Used to calculate latencies.
StartingOffset	Allows setting an offset from the Creation Time for when a packet is queued. Can be a fixed value to simulate sending a packet at some time other than the beginning of a cycle or can be a random number to simulate jitter in packet sending from an application.
Preemptable	Boolean indicating whether the packet is preemptable or not. Based on packet size and express queues and priority.
PreemptID	Random number generated when a packet is preempted. ID is same for all fragments (used to consolidate fragments).
PreemptFragNum	Integer containing fragment number of a preempted packet. 1=first fragment, 2=second fragment, and so on.

EthernetPacketState Class

The EthernetPacketState class contains information that may change as a packet moves within a EthernetSwitch, EthernetEndpoint, or EthernetCable or as it moves from one entity to another. The ESS creates new copies and destroys expired packets as a packet moves through the simulated network. Multiple copies of an EthernetPacketState class containing the same EthernetPacket may exist simultaneously. For example, an instance of an EthernetPacketState could exist for an egressing packet from an EthernetSwitch, ingressing and egressing instances within an EthernetCable, and an ingressing packet on another EthernetSwitch, all containing an exact copy of the same EthernetPacket instance. **Table 3** defines various properties of the EthernetPacketstate class.

Table 3: Major properties of EthernetPacketState class

Property	Description
State	Enum used to denote the state of an instance of an EthernetPacketState class.
StartTime	Denotes the time at which a packet entered the state.
EndTime	Denotes the time at which a packet is expected to exit the state.
gapTime	Captured for egressing only. Denotes any gap between this packet and the previous packet. Does not include interframe gap time.
pktTime	Time needed to transmit or receive a packet based on the current port speed. Does not include preamble, start of header, or interframe gap.

Property	Description
inPort	Ingressing port of the packet.
Outport	Egressing port of the packet.
EthernetPkt	Instance of an EthernetPacket class containing the static information for a packet.

Packet States

As an Ethernet packet moves through the simulated network, it may exist in various states during its journey. In fact, the same packet may exist in multiple states simultaneously (for example, egressing in one device while ingressing in another). **Table 4** describes the various states of a packet as it is processed by the ESS.

Table 4: Ethernet Switch Simulator Packet States

State Enum	Description
EGRESS_SCHEDULED	This is the initial state when an EthernetPacketState instance is created by the main application in an EthernetEndpoint.
INGRESSING_PREAMBLE	This state indicates that the preamble and start of header portion of an Ethernet transmission is occurring at the ingress of a switch, cable, or endpoint.
INGRESSING_IFGAP	This state indicates that the interframe gap portion of an Ethernet transmission is occurring at the ingress of a switch, cable, or endpoint.
INGRESSING_PACKET	This state represents the time needed to ingress a packet.
CABLE_QUEUED	If the transmission time of a packet is shorter than the delay configured for an instance of an EthernetCable class, the packet is stored in the instance and is set to this state.
QUEUE_PENDING	A packet enters this state upon completion of the INGRESSING_PACKET state. This state represents the time for a switch to execute items like CRC processing, address lookup, and queuing the packet.
QUEUED	This state simulates the time needed by a switch to evaluate which packet should be egressed next.
QUEUED_FRAGMENT	If preemption occurs, the remaining fragment is stored here for later egress and set to this state.
EGRESSING	This state represents the time needed to egress a packet.
PREEMPTED_EGRESSING	This state is used to alert receiving devices that the packet being ingressed has been shortened due to preemption.
RECEIVED_FRAGMENT	Received fragments are stored here until the last fragment is received, at which time the packet is reassembled and queued.
RECEIVED_IN_ENDPOINT	This is the terminal state of a packet, when it arrives at its final endpoint destination.

Ethernet Switch Simulator Parameters

Table 5 describes many of the variables used to control the desired simulation scenario. Each of these variables can be changed in the application defining the scenario to be simulated. Various time units are provided in the output of the Ethernet Switch Simulator, but the simulator uses nanoseconds (nsec) for all internal calculations.

Table 5: Common Simulation Parameters

Variable Name	Description
QueueingTime	Value used to simulate the time (after the last octet of a packet is received) a switch needs to complete the reception of a packet and place it into a queue.

Variable Name	Description
processingTime	Value used to simulate the time it takes a switch to evaluate its egress queues contents to decide which packet to egress.
Preempt	Tuple containing preemptable priority queue numbers. Empty for no preemption.
preemptionTime	Value used to simulate the time it takes a switch to evaluate its express queues contents to decide which packet to egress.
cutThrough	Boolean (True = use cut-through, False = use store-and-forward.
cutThroughTime	Value used to simulate time it takes a switch to make a cut-through decision after “cutThroughOctets” are received.
cutThroughOctets	Value representing the number of octets that must be received by a switch before it can decide whether the packet can be cut-throughed.
phyDelay	Value representing the average delay time of the Ethernet physical interface.
simulationTime	Value representing the time period to simulate.
simulationStep	Value used to increment the simulation time for each step of the simulation.
portSpeed	Default port speed for all Ethernet ports in the simulation.
cableLength	Value (in meters) used for simulating delays introduced by Ethernet cables.

Addressing

To simulate packet addressing, the Ethernet Switch Simulator assigns a property of type string to instances of EthernetEndpoint classes during their creation. It uses a recursive algorithm to preassign address lookup tables in instances of EthernetSwitch classes as the network is constructed. Upon receiving a packet, the EthernetSwitch can use this information to determine the correct destination port and that no external address lookup is required.

EthernetEndpoint Class

The EthernetEndpoint is a single port device. When connected directly to an EthernetSwitch, a switched endpoint can be simulated. If connected to an EthernetCable, it can represent a device with a single Ethernet port. Prior to executing the simulation, packets are queued in EthernetEndpoints representing the Ethernet traffic to simulate. Currently, only a single queue is implemented, so support for multi-cycle patterns, where a slower cycle pattern may need to be distributed over several fast cycles, is not supported. Note, this scenario may exist when a controller is sending packets to multiple devices at different cyclic rates.
The current queuing of packets in the ESS is designed for device to controller communication. Packets are queued cyclically, starting with data packets, followed by safety packets, if applicable, and then interfering packets (if selected). The number of interfering packets (and their size) can be configured. Interfering packets are queued in non-express queues and are used to simulate best-effort traffic that could affect cyclic traffic.

Creating Packets

The schedulePacket method is used to create packets in the EthernetEndpoint instances prior to simulation. The schedulePacket method takes the following arguments:

- pktSize: Size of packet (not including preamble or IF Gap) in octets.
- destAddr: Final destination address for packet.
- priority: Queue number for QoS, 7 being the highest, 0 the lowest.
- startTime: Desired start of egress, in nanoseconds.
- Offset: Offset from startTime for packet egress. Can be used to add random jitter or to represent a cyclic offset.
- pktName: name to identify packet as it traverses through the network. All packets in the simulation are uniquely named to assist in analyzing output.

EthernetCable Class

The EthernetCable class is used to simulate the delay effects of Ethernet physical interfaces and Ethernet cables. It is a two-port device supporting bidirectional packet flow. The current implementation uses 5 nsec delay for each meter of cable, representing the typical delay of twisted-pair medium. Because there is an Ethernet physical device at each end of a EthernetCable, the delay from ingress to egress is two times the physical interface delay plus 5 times the cable length.
For large packets, where the transmission time exceeds the delay calculated by the EthernetCable instance, a packet can be both ingressing and egressing at the same time. If the transmission time of the packet is shorter than the EthernetCable delay, the packet is queued in the EthernetCable instance.

EthernetSwitch Class

The EthernetSwitch class is used to simulate the behavior of an Ethernet switch (sans a physical interface, which is simulated in the EthernetCable class). Fundamentally, the EthernetSwitch is responsible for receiving an ingressing packet and, at some later time, egressing that packet out the appropriate interface port. The number of ports and their speed can be chosen when the class in instantiated. Several options (such as switching method and express queues) can also be chosen. The following sections describe the EthernetSwitch behavior based on these options.

Store-and-Forward Switching Behavior

In store-and-forward switching (without preemption), an Ethernet switch must receive the last octet of a packet before it can begin any processing of that packet. This consists of validation of its contents via a cyclic redundancy check (CRC), determining the correct egress port, and placing it in the appropriate QoS queue of that port. The queuingTime is used to simulate the time a switch needs to perform those actions, represented by the QUEUE_PENDING state. The EthernetSwitch uses the address lookup table (created during the network construction) to decide which port should be used to later egress the packet. Upon completing of the queuingTime, the packet is placed in the appropriate queue and its state is set to QUEUED, where it remains until the time represented by the processingTime variable has elapsed. This simulates the time an Ethernet switch needs to evaluate port queue contents to determine which packet to next egress out of a port. Switches are constantly performing this activity, even when packets are currently egressing thus enabling contiguous egress (back-to-back) of packets separated by only an Interframe gap. The ESS simply uses this value as the minimum time a packet must be queued before it will be available for egress. When a port is not egressing a packet, the EthernetSwitch examines the queued packets in order of priority. If a packet is ready for egress (the processingTime has expired), the packet is moved to the EGRESSING state and egress begins. Therefore, if multiple packets are simultaneously available for egress, the packet with the highest priority will be chosen. The packet will remain in the EGRESSING state until the last octet of the interframe gap time has been reached.

Cut-through Behavior

Cut-through switching can occur after a fixed-length portion of a packet (as specified in the cutThroughOctets variable) has elapsed. After this time has elapsed and if the packet's destination queue supports cut-through, it is copied and its state is set to QUEUED, using the time provided by the CutThroughTime variable. If the egress port is free after this time has elapsed, the EthernetSwitch begins egressing the packet while continuing to ingress the remainder of the packet. If the output port is in use (blocked) and preemption is disabled (that is, no express queues are present), the packet will be treated using store-and-forward. The ESS does not support immediate egress upon completion of the packet that blocked cut-through.

Preemption Behavior

This section describes the EthernetSwitch behavior when preemption is enabled. Two cases need to be considered; with and without cut-through switching.
When cut-through switching is enabled, only express packets utilize cut-through. Non-express packets always use store-and-forward switching. This is because remaining octets are unavailable when using cut-through switching (unless the number of octets that must be received to make a cut-through decision exceed the minimum Ethernet packet length). Hence, non-express packets are treated as described in the previous section regarding store-and-forward switching.
Processing of packets with preemption enabled remains the same as described in the store-and-forward and cut-through switching sections through the QUEUE_PENDING state. Once the time expires for a packet in the QUEUE_PENDING state, it moves to the QUEUED state. Express packets use the preemptionTime variable to decide when they can exit the QUEUED state where non-express packets use the processingTime variable as previously described.
With preemption enabled, the EthernetSwitch no longer waits until an egressing packet completes to choose which packet is egressed next, but will examine the express queues if the egressing packet is a non-express preemptable packet and is at a point where it can be successfully fragmented. If preemption can occur, the currently egressing packet is truncated and an additional 4 octets will be added to simulate the required CRC octets. The state of the fragmented packet is set to PREEMPTED_EGRESSING, its size and endTime adjusted, and the properties related to fragments in the EthernetPacket instance will be adjusted. In addition, the remaining fragment will result in the creation of a new EthernetPacketState instance with the size of the remaining octets. The fragment properties of this instance are adjusted to indicate that this is the next fragment, the state is set to QUEUED_FRAGMENT, and it is queued for next egress (provided there are no express packets to egress). Finally, the preempting packet will egress upon completion of egressing the initial fragment.

EthernetPort Class

The EthernetPort class is instantiated for every port in an EthernetSwitch, EthernetEndpoint, and, in the future, EthernetCable class (device) instance. It is responsible for egressing and ingressing of packets, as well as understanding any connection between the devices. Understanding the connections is paramount. If a packet is egressing on a port connected to another port, then it must also be ingressing on the connected port. If preemption occurs, both ports must be adjusted to reflect the change of the preempted packet. This is one of the responsibilities of the EthernetPort class.

Simulation Example

This section will describe the use of the ESS to analyze a simple system (shown in **Figure 6**). This simple system consists of a controller, 6 servo drives (ServoDrive 1-6), and 2 rack I/O devices (BlockI/O 1-2). This system is programmed in the ESS to analyze Ethernet packet latencies and the influence of interfering traffic. EtherNet/IP implicit cyclic messaging is used, with the servo drives communicating at a cyclic rate of 1 msec and the rack I/O communicating at a cyclic rate of 4 msec. Only device to controller traffic is considered. A baseline is first established to determine minimum packet latencies. The focus of the analysis will be on packets from ServoDrive1 and BlockI/O1, because these are furthest from the controller and will have the greatest latencies.

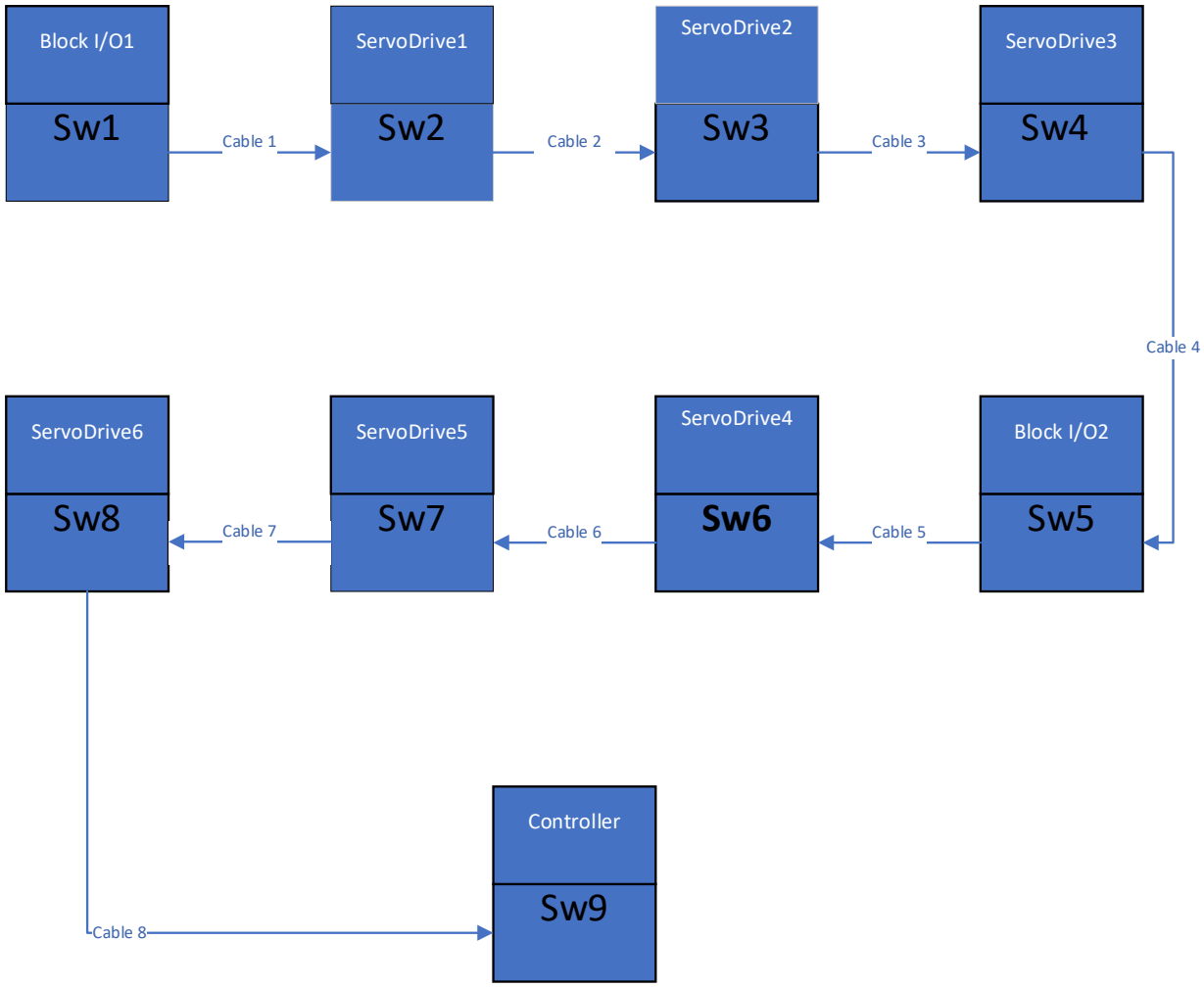

Figure 6: Sample Simulated System

Table 6 contains the parameters used for the simulation. The content of the table is direct output from the ESS. For this simulation, all cable lengths were set to zero.

Table 6: Simulation Parameters

Sim Parameter	Value	Unit
Queueing Time	800	nsec
Processing Time	700	nsec
Preemption	6, 7	
Preemption Time	500	nsec
cut-through	FALSE	
Cut-through Time	400	nsec
Min CT Octets	14	octets
Cable Length	0	meters
PHY Time	500	nsec
Sim Time	30000	usec

Packet Size

To aid users of the simulator, only the payload of packets needs to be defined. The simulator will then add 74 octets (not including header, start of frame delimiter, or interframe gap) of overhead to account for Ethernet and CIP headers (see **Table 7**). UDP transport is assumed (50 octets) and implicit CIP I/O messaging (24 octets). If the resulting addition is less than the minimum required Ethernet packet size, 64 octets is used, because this is the minimum legal Ethernet packet size.

Table 7: Ethernet Packet Overhead

Ethernet Frame	Octets
Preamble	7
Start of frame delimiter	1
Inter-packet Gap	12
MAC Destination	6
MAC Source	6
802.1Q Tag (optional)	4
Ether Type	2
Frame Check Sequence	4
IP V4	20
UDP	8
CIP IO Overhead	6
EIP Overhead	18

Device Characteristics

Table 8 describes the device characteristics being simulated. The types of devices, cyclic packet rate, quantity, data payload, packet size, and packet priority are provided. This table is actual output from the ESS.
Two types of devices are shown (a drive and a block I/O device). From the table, it can be seen that there 6 drives and 2 block I/O devices in the simulation. For example, the drive sends 90 octet packets (16 octets of payload and 74 octets of overhead) with a priority of 7 every 1 millisecond.

Table 8: Simulated Device Characteristics

Device Type	Cycle time (us)	num dev	payload	Pkt Size	Priority
Servo Drive	1000	6	16	90	7
Block I/O	4000	2	251	325	6

Baseline

In order to evaluate the effects of interfering traffic on cyclic traffic, a baseline, without any ancillary traffic, must first be established. This section will examine, in detail, the simulator output for two communication cycles without any interfering traffic.

Table 9 shows output from the ESS for the first 2 milliseconds of simulation. All times are in nanoseconds. It contains information regarding packets egressing from sw9, which is connected to the controller. Note, the simulator can capture egress packets from any port. During the first millisecond, all devices send packets while during the second millisecond, only the servo drives send packets, which are expected to be received by the controller. This is because the block I/O devices only send packets every 4 milliseconds and the servo drives send packets every 1 millisecond.
The table also illustrates the packet naming convention used by the ESS. Every packet in the simulation is given a unique name. the first portion of the name is the name of the initiating device (for example, servoDrive2, BlockIO1) followed by a packet number, which starts with 1 and is incremented each time a

device sends a new packet (for example, pkt 1, pkt 2, and so on). Thus, a packet with the name servoDrive2-pkt7 is the seventh packet sent by the device named ServoDrive2.
It is interesting to note that during the first cycle, the packets from ServoDrive1 and ServoDrive2 are delayed due to the presence of the packet from the BlockI/O2 device. This is not the case for the second cycle. In a line topology, when all devices send the same size packet at the same time, a gap is created (equivalent to the switch delay plus the PHY and cable delays) provided this gap is greater than the interframe gap time. The gap time shown in the table represents an excess gap time, not including the interframe gap. Thus, a gap of zero means only the interframe gap is present.

Table 9: Baseline Simulation Results

start time	gap time	Latency	Pkt Name
19680	19680	27520	ServoDrive6-pkt1
30020	1540	37860	ServoDrive5-pkt1
40360	1540	48200	ServoDrive4-pkt1
61040	11880	68880	ServoDrive3-pkt1
145040	75200	171680	BlockI/O2-pkt1
172640	0	180480	ServoDrive2-pkt1
181440	0	189280	ServoDrive1-pkt1
261260	71020	287900	BlockI/O1-pkt1
1019680	730820	27520	ServoDrive6-pkt2
1030020	1540	37860	ServoDrive5-pkt2
1040360	1540	48200	ServoDrive4-pkt2
1061040	11880	68880	ServoDrive3-pkt2
1071380	1540	79220	ServoDrive2-pkt2
1081720	1540	89560	ServoDrive1-pkt2

Table 10 provides a summary of the minimum and maximum packet latencies (in nanoseconds) for the Servo Drive and Block I/O device furthest from the controller (ServoDrive1 and BlockI/O1). This table was generated by a script that uses a regular expression (shown in column 1 of the table) where the asterisk character will match all packet numbers from ServoDrive1 and Block I/O1 in the simulator output to extract minimum and maximum latencies.

Table 10: Summary, baseline Minimum and Maximum Latencies

Pattern	Min	max
ServoDrive1-pkt*	89560	189280
BlockI/O1-pkt*	287900	287900

Interference

The ESS can generate interfering packets to allow analyzing the effect on cyclic communication. The ESS allows specifying a single packet size, priority, and number of interfering packets per cycle. The ESS queues up interfering packets directly after each cyclic packet for each device. Hence, if 5 interfering packets are programmed, each device will send its cyclic packet followed by 5 interfering packets at the specified cyclic rate. Thus, each servo drive would send 5 interfering packets every 1 msec cycle and each rack I/O device would send 5 interfering packets every 4 msec cycle.
Table 11 shows the effect on packet latency for various quantity and sizes of interfering packets. Column 2 considers 5 interfering packets of size 123 octets, column 3 for 50 interfering packets of size 123 octets, column 4 for 1 750 octet interfering packet, and column 5 for 1 1500 octet interfering packet.

Table 11: Summary, Latencies with Preempted Interference

Pattern	5-123	50-123	1-750	1-1500
ServoDrive1-pkt*	199540	216180	191520	190360

Pattern	5-123	50-123	1-750	1-1500
BlockI/O1-pkt*	318660	343620	297940	291200

As expected, the largest latencies occur with 123 octet packets, because these cannot be preempted. For the larger packets, the maximum influence occurs during the first 60 or last 64 octets of transmission, because preemption cannot occur during these times. If preemption is requested during the first 60 octets of transmission, it will wait until that time before preempting.

Bandwidth Analysis

The ESS provides a capability to calculate the bandwidth utilization for individual output ports. **Table 12** shows the bandwidth utilization when no interfering packets are present and **Table 13** shows the bandwidth utilization when 50 interfering packets are present.
For each table, four time ranges (in msec) and bandwidth (in percent) for four switches are shown. The sw1 column represents the bandwidth on the output of sw1 connected to sw2, the sw2 column represents the bandwidth on the output of sw2 connected to sw3, the sw3 column represents the bandwidth on the output of sw3 connected to sw4, and the sw9 column represents the bandwidth on the output of sw9 connected to the controller.
For the case where no interfering packets are present, the bandwidth utilization is quite reasonable. As expected, during the first msec, the bandwidth used is higher than the 2nd, 3rd, and 4th msec due to the presence of packets from the rack I/O devices.

Table 12: Bandwidth Analysis, 0 Interfering Packets

Start Time	End Time	SW1	SW2	SW3	SW9
0	1	2.76	3.64	4.52	10.8
1	2	0	0.88	1.76	5.28
2	3	0	0.88	1.76	5.28
3	4	0	0.88	1.76	5.28

With 50 interfering packets per cycle, the bandwidth utilization is quite different. Because the current implementation of the ESS does not support multiple queues in an EthernetEndpoint, it is important to not schedule more than 100% bandwidth utilization at an EthernetEndpoint. Because the ESS can monitor any egress port, a separate simulation (not shown) was run to determine the bandwidth usage for each EthernetEndpoint, which concluded that the servo drives with 50 interfering packets per cycle only used 58.08% of the bandwidth on the connection between the servo drive and its switch.
The Ethernet switches starting with sw3 are overloaded with packets, as demonstrated by 100% bandwidth usage, but due to priorities, the servo drive and rack I/O packets get through, albeit with some additional latency, as demonstrated in **Table 11**. Overloading the switches along with longer simulation times can allow the simulator to approximate worst-case latencies.

Table 13: Bandwidth Analysis, 50 Interfering Packets

Start Time	End Time	SW1	SW2	SW3	SW9
0	1	59.96	98.802	98.912	95.998
1	2	0	77.318	100	100
2	3	0	58.08	100	100
3	4	0	58.08	100	100

Theoretical evaluations show that the worst-case latency impact that can occur is when a 123-octet non-express packet begins egressing just before an express packet is ready for egress. This is because a packet of 123 octets is the maximum non-preemptable packet size. At 100 MB/sec, a 123-octet packet

consumes approximately 11 µsec of the network (including preamble, start of header, and interframe gap). In the worst-case situation, this interference would occur at each switch (hop). In our simulation example, interfering packets are generated by the devices, so the I/O packet from BlockI/O1 has 7 switches where this could occur (sw2 – sw8), resulting in a theoretical latency impact of 77 µsec. looking at the simulation results when interference is present (see **Table 10** and **Table 11**) the maximum latency variance is for the packet from BlockI/O1 with 50 interfering packets (approximately = 56 µsec). Further experimentation (using longer simulation times and more interfering packets) results in approaching the theoretical maximum influence (not shown). This is understandable, because the theory requires an interfering packet to begin egress just before and express packet is ready for egress at every switch.

Conclusions

This paper presented simulation as one method for analyzing cyclic EtherNet/IP traffic in a network. Switch behaviors were described along with how interference can affect egressing packets. A simulator based on the Python programming language was described and its usage was demonstrated on a simple line topology network consisting of two types of traffic at different priorities and cyclic rates. Even this simple system offers some observations which may not have been intuitively obvious; that even without interference, the packets received at the controller are not in the order of the devices on the network (that is, the packet from BlockI/O2 does not occur after the packet from ServoDrive4, but rather after the packet from ServoDrive3). This is due to store-and-forward switching and the I/O packet size being significantly larger than those of the servo drives. Attempting to analyze even this simple system in an Excel workbook would be extremely complex, especially if one wanted to examine variance such as store-and-forward vs. cut-through, preemption vs. no preemption, packet size and priority variations, and so on.
The ESS described in this paper allows analyzing complex network topologies and cyclic data patterns by simply making a few changes to the main program file and re-running the simulation.

References

[1] https://www.oreilly.com/library/view/ethernet-switches/9781449367299/ch01.html, “Ethernet Switches,” by Joann Zimmerman, Charles E. Spurgeon, Retrieved 10-Nov-2021
[2]https://www.ciscopress.com/articles/article.asp?p=357103&seqNum=4#:~:text=Layer%202%20Switching%20Methods%201%20Store-and-Forward%20Switching.%20Store-and-forward,of%20cut-through%20and%20store-and-forward%20switching.%20More%20items...%20, “Layer 2 Switching Methods > How a LAN Switch Works | Cisco Press,” Retrieved 10-Nov-2021
[3] https://en.wikipedia.org/wiki/Cut-through_switching, “*Cut-through Switching,”* Retrieved 10-Nov-2021
[4] https://en.wikipedia.org/wiki/Physical_layer, “physical Layer,” Retrieved 10-Nov-2021