NETWORK TOPOLOGIES FOR LONG ARMATURE LINEAR MOTOR MULTI-CAR ELEVATOR SYSTEMS

by

GÖKALP ÇETİN

Submitted to
The Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

SABANCI UNIVERSITY

December 2020

NETWORK TOPOLOGIES FOR LONG ARMATURE LINEAR MOTOR MULTI-CAR ELEVATOR SYSTEMS

APPROVED BY

DATE OF APPROVAL .....

ACKNOWLEDGEMNETS

First and foremost, I would like to express my appreciation to my supervisor Dr. Ahmet Onat for his constant leadership, support and knowledge. I feel fortunate to have had the opportunity to work with a considerate and easygoing supervisor.

Secondly, I would like to thank my friends in Sabanci University for their support and fellowship. I feel so lucky to have such a great group of friends that is almost like a family for me.

Finally, I would like to thank my family for their patience and sacrifices. They have been standing behind me in every situation. I will forever be grateful for their encouragement.

Abstract

Long armature linear motor is an example of a distributed control system with a large number of motor drivers spread in a linear fashion. Having multiple movers on the same motor increases the level of complication while adding a new challenge to the control and network aspects of it.

In such a system, there is no room for error on the communication between nodes, as the motor drivers must be accurate despite delay and packet loss. Although delay and packet loss in communication networks are almost unavoidable, reducing these disadvantages by choosing a suitable network topology with accompanying communication protocol is possible.

In this thesis, Ethernet, CANBUS and EtherCAT protocols have been tested for their suitability for a reliable real-time communication protocol to be used on control systems while being implemented on three different network topologies taking three different approaches to the same problem. These topologies are introduced for the communication of motor drivers and sensor nodes of long armature linear motor with multiple elevator cars.

Topology G with its global approach, is a system containing motor drivers spread linearly, a main computer to plan and coordinate the motion of the linear motor movers and a gateway computer in between. The structure of this topology, with its simplicity introduces a bandwidth problem having over a certain number of motor drivers. Due to this, some of the sensor nodes messages collide and packets drop while running multiple elevator cars simultaneously.

Unlike Topology G, Topology L is a hierarchical system with a more localized approach, consisting of motor drivers, gateway computers that group motor drivers into small networks, and a main computer connecting to every gateway computer with an outer network. Thanks to the structure of this topology, increasing the number of motor driver nodes is not increasing each individual network load.

Finally, Topology R with an unorthodox method uses a completely different network structure yet, successfully utilizes gateways to group motor drivers and other nodes, similar to Topology L does, so that increasing the number of motor drivers does not affect the overall load on the network.

The proposed topologies are simulated on MATLAB Simulink using TrueTime, a toolbox for simulation of distributed real-time control systems. As creating such a system with multiple networks and many nodes connected to those networks is a re-iterative task, instead of using Simulink as its graphical user interface (GUI) with drag and drop method, we used MATLAB scripts to create and modify the Simulink models which also allowed us to easily change parameters and test various cases recording and analyzing the responses of the system.

The advantages and disadvantages of all topologies are explained in detail with examples of some test cases, comparing the performance of them based on reliability and delay amount. The results indicate that when compared the three, Topology L with a local approach and Topology R with a ring approach have a better performance with higher reliability and nominal delay than Topology G. Therefore, either Topology L or Topology R is suggested for the communication of motor drivers of long armature linear motor with multiple elevator cars.

Table of Contents

List of Figures

1 - Introduction

Since late 1800s, when the first skyscraper was introduced, architects and engineers are creating more useful spaces within the limited area of land by designing taller buildings allowing more people to live and work in. The ten-story high Home Insurance Building in Chicago, which was built in 1884-85 is considered as the first skyscraper today, which initiated the building of taller buildings movement. This led to another problem of reaching from one floor to the other with less effort and time which was challenging especially as these buildings got taller and taller. It was around same time Elisha Otis introduced the safety elevator that allowed easy and safe movement from one floor to the other for passengers and other heavy weight goods.

In very tall skyscrapers, one of the bigger challenges is the average amount of time it takes for an individual to get from one floor to the other such that the height of the building does not become illogical [1], [2], [3]. To achieve this, the number of elevators can be increased, but this means every elevator added will be stealing precious space from the building on every floor which could be utilized for other things. One way to tackle these problems is to use multiple elevator cars on the same elevator shaft. Although implementing this idea on traditional cable driven elevators are mechanically complicated and inefficient, with the introduction of linear motors, along the whole elevator shaft, driving each separate elevator car, there will be no physical connections such as cables reaching to the top of the building, eliminating the mechanical complications [2], [1], [3], [4], [5]. This cableless elevator car introduces some other challenges of its own; electrical connections such as position sensors, power supply and call buttons all needing to be implemented externally [6], [7], [8]. With these limitations considered, the best design option is to use long armature linear motors where the stator contains the coils and the movers contain the permanent magnets.

Having the coils on the stator side allows easier electrical connections, equipped with special motor drivers they can also allow smooth braking and even a low precision position sensing which is very useful to track the elevator cars' position and speed while moving in higher speeds. However, this does not apply for lower speed, high precision measurements, for that purpose, optical linear encoders are used and they are connected to the same

network as motor drivers so that the position data can be transferred in a very short amount of time.

We approach this multi car elevator system with different network topologies to test which are always suitable and reliable. In large buildings with multiple cable-driven elevators, a network tries to minimize the waiting time for any passenger at any given time by arranging which elevator car to be sent to which floor and after that, the large electrical motor at the top or the bottom of the building use set of pulleys and counterweights to move the elevator car to where it needs to go. It is an electrically simple but mechanically more complicated system. When the cable restrain is removed the number of elevator shafts can be decreased without necessarily decreasing the number of elevator cars. A similar network handles the same task of deciding which elevator car to be sent to which floor to minimize the amount of time the passengers wait. But the main difference is, rather than having a single motor driving the car up or down, there are multiple sections that compose a long armature linear motor throughout the whole building. This, while allowing for almost no mechanical complication such as pulley sets and cables, introduces the challenge of driving consecutive motors in perfect harmony, one after the other for a smooth and steady ride.

In this thesis, we will concentrate on simulations of different network topology models on MATLAB Simulink based simulator, TrueTime. TrueTime allows us to model and simulate real-time systems and networks topologies which can use different communication protocols. By using parametric MATLAB scripts and functions, we can create network topologies with various options and variables that we can play to fine tune as we wish.

2 - Background

In this section, existing network topologies and communication protocols that are being used with those topologies will be given followed by, proposed topologies and protocols for our specific case used in this thesis as well as their advantages and disadvantages.

2.1 – Existing Network Topologies

Network topologies are classified into two basic categories, physical topologies and logical topologies [9]. A physical topology is the transmission medium layout to link devices in physical world such as cabling layout, location of nodes and the links in between those nodes and other devices. A logical topology, however, is the way that the data passes through the network from one device to the next regardless of the physical interconnection of the devices. A network does not necessarily have the same logical topology as its physical topology. In this thesis we will be mostly talking about the logical topology of the network therefore it will be mentioned as network topology as shortly.

There are many various topologies and standards already existing, which are created to solve earlier challenges or problems in communication systems through years. In this section, we checked the most common network topologies and that helped us to decide on which network topologies are most appropriate for our system.

2.1.1 – Point-to-Point (Telecommunication)

It is the most basic link between two endpoints. There can be different versions depending on how it is set up but we can divide into two types, permanent point-to-point; there is one and only one connection between two specific nodes at all times, and on-demand point-to-point; where there can be more than two nodes but any link only connects two nodes to each other and for one of these nodes to be able to connect to a different node, the old link needs to be removed. The best example for these kinds of networks are phone lines. There

used to be telephone centrals which connects the caller's line to the person who is being called and once the phone call ends the line is removed until a new call. Telephone centrals let their place to automation, but the main idea is still viable.

Figure 1: Point-to-point Topology

2.1.2 – Daisy Chain

In a daisy chained network all nodes are connected in a series way, and if a message is intended for a specific node, it is transmitted one by one from the source to that node bouncing off every node in between. A daisy chained network can be in two different forms: linear and ring. In a linear topology every node is connected only to the next one, and the nodes that connect to only one node are at each end of the network. By simply connecting those ends to each other in a linear topology, we get a ring topology. On a ring topology if a message is sent to a specific node, every node in that topology will eventually get the message. One great advantage of using ring topology instead of linear topology is the fault tolerance (only if implemented) that, in a situation where one of the nodes die on the linear topology, the network is cut into two pieces where in a ring topology if one of the nodes die, the ring topology becomes similar to a linear topology and can continue until fixed.

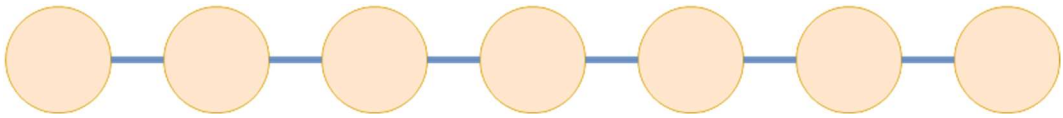

Figure 2: Daisy Chain (Linear) Topology

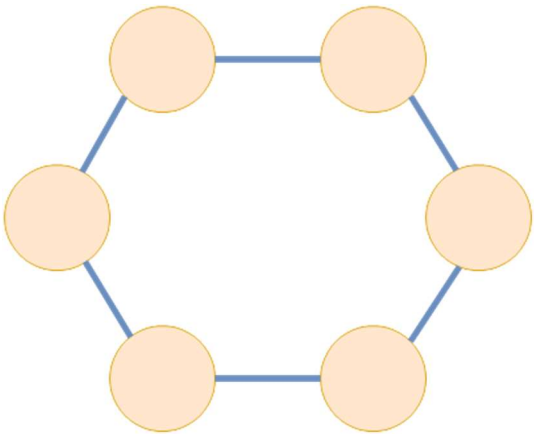

Figure 3: Ring Topology

2.1.3 – Bus

In a network if all the nodes are connected to a single central line and any two or more nodes can communicate through that central line it is called a bus topology. This central line acts as a communication medium between every single node on the network and it allows all nodes to receive messages simultaneously [10]. In bus topologies the message contains the intended recipient node's address as the message is sent to all nodes simultaneously, this way any node can check the address on the message with its own address and if it is not matching, data part of the message is ignored.

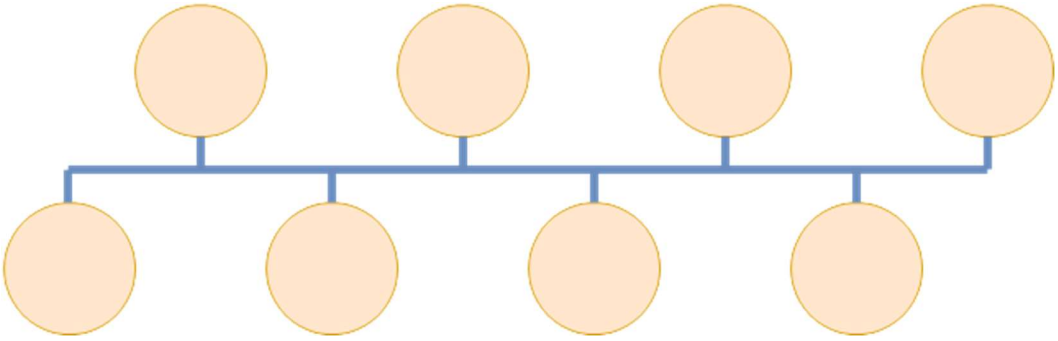

Figure 4: Bus Topology

2.1.4 – Star

In a star topology, every node in the network is connected to a central node also known as a hub. This hub acts as a server where all other nodes and peripherals act as clients. All message traffic within the network passes through the central hub, making the hub a repeater. Since every node is separately connected to the central hub, it is very easy to add or remove new nodes without affecting others. The downside of this is that, if the communication frequency of nodes is too high the central hub becomes a bottleneck on the maximum communication speed between nodes. Even though, every node is separately connected to the hub and if a link between a node and the hub fails, the rest of the network can work; if the hub fails the whole network fails hence creating a single point of failure.

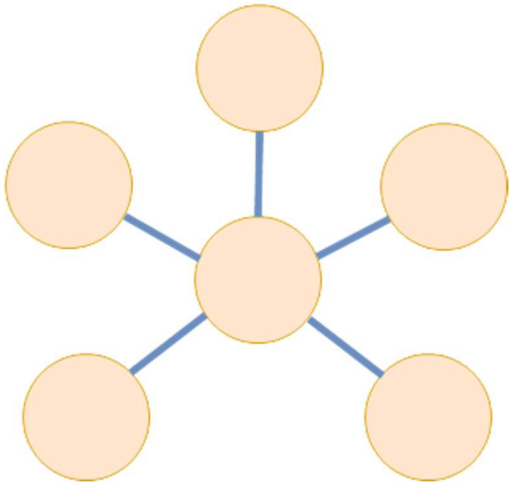

Figure 5: Star Topology

2.1.5 – Ring

A ring topology, as we introduced in daisy-chained networks, is a linear topology with the two ends connected to each other. In ring topologies data travels in one direction on the network and each node passes the data to the next until a message is sent to the intended recipient. There is no server and client, all the nodes are peers. This allows better

performance on increased loads on network, but it can also create problematic situations as the whole network bandwidth is bottlenecked by the weakest link between any two nodes. (see Figure 3)

2.1.6 – Mesh

A mesh network consists nodes that connect to more than two other nodes allowing some or all nodes to have more accessibility within the network. We can classify mesh networks as partially connected mesh networks and fully connected mesh networks. A partially connected mesh network have some of the nodes connecting to multiple other nodes acting as a relay for other nodes, where a fully connected mesh network has all nodes connected to all other nodes within the network. In mesh networks in general, as the number of the nodes increase, the amount of links necessary increase a lot faster; especially in fully connected mesh networks so it becomes impractical to use mesh networks in systems that consist a large number of nodes.

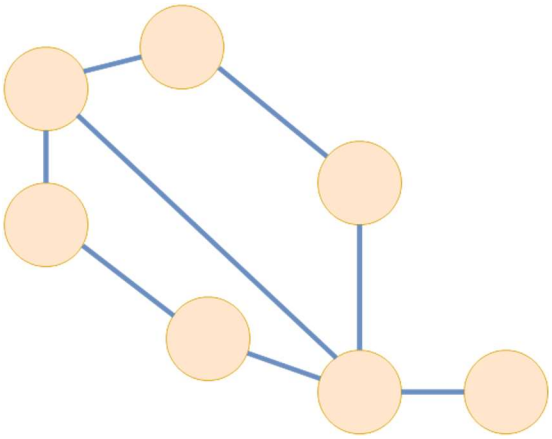

Figure 6: Mesh Network Topology

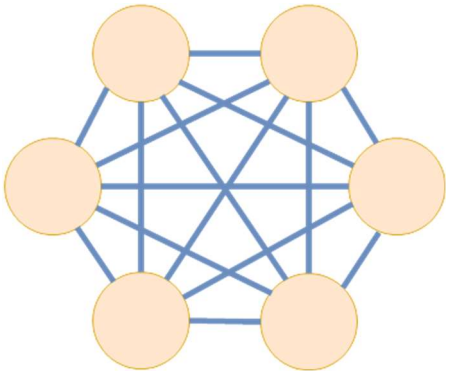

Figure 7: Fully Connected Network Topology

2.1.7 – Hybrid

A hybrid topology also known as a hybrid network combines two or more types of topologies such that the resulting topology cannot be expressed by any of the standard topologies [11]. One example to a hybrid topology is the tree network which interconnects star networks to each other via bus networks. Some systems have complicated requirements and specifications such that none of the simple network topology types can manage to satisfy, so in these cases a hybrid topology is tailored specifically for these purposes.

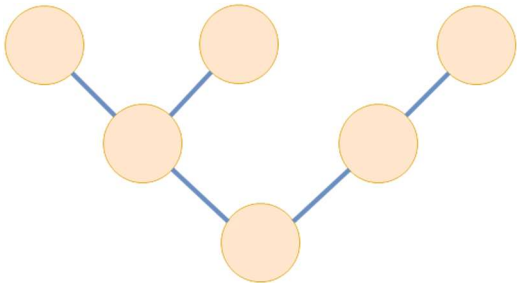

Figure 8: Tree Network Topology

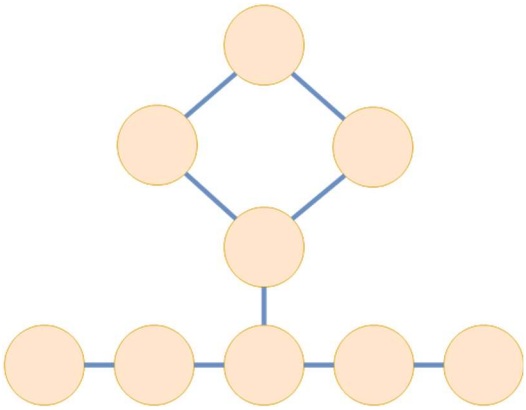

Figure 9: Hybrid Topology (Ring + Linear)

2.2 – Existing Communication Protocols

As we can think of a network topology as the structure of a network, we can think of the communication protocol as the set of rules that allow nodes to communicate in a topology. This includes the rules, syntax, synchronization of communication and sometimes possible error recoveries [12]. When a specific message or data is sent from a node to another through the network, there is an exact intended meaning and possibly a pre-determined response for any given situation. [13] To make sure that, there needs to be some sort of standard that communication protocols follow depending on technical requirements

including the network topology that protocol is going to be used on. [14] Some of these standards are Internet Engineering Task Force (IETF), Institute of Electrical and Electronics Engineers (IEEE), International Organization for Standardization (ISO).

There are so many different communication protocols in existence, to go over all of them to choose which one to use in the system is very difficult, but there is a way that can make it easier. We already have some requirements for the system to meet, and if we also decided on the network topology to use, the number of communication protocols that can be compatible with the network topology we chose are limited. This is because some communication protocols are designed on purpose to be used with one or a few specific network topologies. We checked the following communication protocols which can be implemented with the network protocols from section 2.1.

Testing different topologies accompanied by different communication protocols to see which one can handle this kind of a task, indicated the advantages and disadvantages of each method and let us choose the most reliable and preferable option.

2.2.1 – Contention-based protocol (CBP)

In a contention-based protocol there is no pre-coordination, and it is mostly used in wireless telecommunication equipment such as radio, where multiple users (nodes) can join on the same channel and the operating procedure “listen before talk” in IEEE 802.11 is applied here [15]. The main advantage of contention-based protocols is the ease of implementation, under light load they can perform efficiently but in higher loads performance drops [15].

2.2.1.1 – Carrier-Sense Multiple Access (CSMA)

Carrier-Sense Multiple Access (CSMA) is a communication protocol where every node needs to verify the absence of any traffic in the network before starting to transmit on it. This tries to minimize the collision of two messages transmitted by two different nodes within the same network. A node that is going to start transmitting, first attempts to determine whether any other transmission is in progress by using a carrier-sense mechanism. If there is such transmission, the node simply waits for that transmission to end before initiating its own transmission. There are a few different variations of CSMA some of which include collision avoidance, collision detection and collision resolution techniques.

2.2.1.1.1 – Carrier-Sense Multiple Access with Collision Detection (CSMA/CD)

In Carrier Sense Multiple Access protocol, a node can use carrier-sensing to determine if there are any ongoing transmissions and if not, that node can start transmitting. But this does not mean there will be no collisions; if two nodes need to transmit at the same time, they will check the communication line if there are any ongoing transmissions, and both of them will see that there are no traffic in the communication line resulting in both of them trying to transmit at the same time and ending up in a collision.

Carrier Sense Multiple Access with Collision Detection protocol allows a node in the network to detect a collision while it is transmitting alongside being able to use carrier-sensing. As soon as a collision is detected, both nodes terminate their transmission, shortening the amount of time required before a re-transmission can be attempted.

CSMA/CD was the protocol used in old Ethernet variants which used repeater hubs. Modern Ethernet networks that we use today is built with network switches and full-duplex connections, so CSMA/CD is no longer needed. Each ethernet segment or collision domain is now isolated. CSMA/CD on Ethernet is still supported for backwards compatibility and

for half-duplex connections. The IEEE 802.3 standard defines all Ethernet variants which until 802.3-2008 are under the name of “Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications”, where after 802.3-2008 switched to the new name shortly as “IEEE Standard for Ethernet”.

2.2.1.1.2 – Virtual Time Carrier Sense Multiple Access (VTCSMA)

Virtual Time Carrier Sense Multiple Access (VTCSMA) protocol is used mostly in hard real time communication systems meaning messages in the system have explicit hard deadlines. These messages are critical to reach before their deadlines or there may be data loss or other consequences to the system. In networks that use this protocol every node in the topology has two clocks; a real time clock and a virtual time clock which runs in a higher rate than real time, and it is reset (set equal to real time point) whenever the node finds the channel to be idle [25]. This extra clock allows scheduling messages to be sent in such a way that collisions between messages transmitted simultaneously by different nodes is avoided as much as possible.

2.2.2 – Token-based protocol / Token Passing

On a token-based protocol there exists a special signal message called “token” which is passed between nodes to authorize the node holding the token to transmit meaning that there is no pre-determined master or slave nodes. By implementation this prevents any two or more nodes to transmit at the same time as there is always only one token allowed. The network checks if there is only one token on the network periodically and sometimes the token might get corrupted or lost, in those situations the communication protocol realizes the token is lost and generates a new token. The advantage of this type of protocols is the nodes can utilize the full bandwidth of the network without idle time with heavy loads, but the disadvantage of this type of protocol is even on light load if a node wishes to transmit, it has to wait for the token, increasing latency.

2.2.2.1 – Token Ring

Token ring is a protocol used to create local area networks (LAN) which is defined in IEEE 802.5 standard. As other token passing protocols, token ring protocol also has a special signal message called a token which passed between nodes on the network in a ring form. Every node can only receive from its neighbor and usually there is one common direction for transmission, so possibility of collision is eliminated. This protocol was introduced in 1984 by IBM as IBM Token Ring LAN and in 1989 it is standardized under IEEE 802.5. Unlike contention-based communication protocols, token passing protocols are deterministic, which means we can calculate the maximum amount of time needed before a specific node can start transmitting. This property makes networks with token passing protocol to be ideal for predictable delay applications and more robust networks [16].

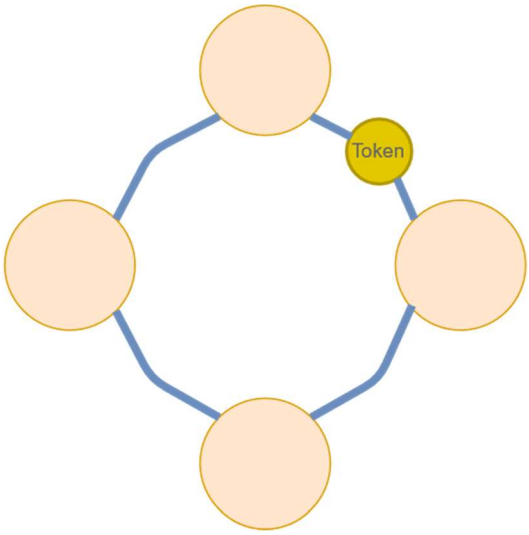

Figure 10: Token Ring Protocol

2.2.2.2 – ARCNET

Attached Resource Computer Network (ARCNET) is a communication protocol for local area networks just like token ring protocol. ARCNET was the beginning of a widely available network between microcomputers such as Amiga 500. It was especially popular

for office automation tasks in 1980s and later it was also applied to embedded systems. It used a star network topology, and this brought so much ease to set up, expand and maintain, leading it to become more popular than its competitor, the linear bus Ethernet of the time. One disadvantage of ARCNET was the requirement of active or passive hubs between nodes if there were more than two nodes when compared to Ethernet but the passive hubs were easy to access and cheap, so it was not a big disadvantage. Later on, as Ethernet switched to an easier to work with physical topology with higher quality wiring, network throughput and reliability increased massively and ARCNET lost the popularity race to Ethernet.

2.2.3 – Polled Bus / Polling

In computer science, polling is the terminology used for, when a client device or program actively sample the status of an external device or task as a synchronous activity. In embedded systems polling is used for checking the state of a line in the network. In a network topology that polled protocols are used, there is a bus-busy line that all nodes have access to and can check if there is any transmission ongoing on the system as well as switching the state of it if they started transmitting a message. In these networks usually by design all the tasks or messages are prioritized and when there are more than one node that wants to transmit, they transmit a poll number on the bus proportional to the priority of the message. After the ongoing transmission ends and the bus-busy line resets, the node that has the highest priority message has the right to transmit, and all other nodes wait until they are the node with the highest priority message waiting to transmit. This protocol works with time slots for better scheduling and every slot duration is equal to the end-to-end propagation time of the bus.

2.2.4 – Controller Area Network BUS (CANBUS)

A Controller Area Network Bus (CANBUS) was created to have a robust communication protocol connecting various subsystems within a vehicle allowing each of the nodes on the bus to communicate with the rest. Later, as it performs reliable enough that it has been used in other applications mostly industrial. CANBUS is a multimaster broadcast serial bus protocol that connects electrical control units (ECUs) and as there are no general master nodes in this protocol, every one of these ECUs can transmit and receive messages, but not simultaneously. When a node transmits, all other nodes receive the message. A message is composed of two things; an ID which sets the priority of the message and the 8 byte sized data. In CANBUS protocol all the nodes are connected to each other via bus lines so there can be collisions but by design this is solved. When the communication bus is idle, it is represented by recessive level (TTL=5v) and any node can start to transmit. If multiple nodes begin transmission simultaneously, As every message start with the ID (priority) part, in the case of more than one node start transmitting at the same time, a priority based bus arbitration steps in and bit by bit the IDs are compared and the node with the recessive ID bit stops transmission until only the most dominant ID is left on the bus and the node with the most dominant ID keeps on transmitting until the end of its message.

2.2.5 – Fieldbus

Fieldbus is the name of a family of industrial computer networks used for real-time distributed control systems. Fieldbus profiles are standardized by the International Electrotechnical Commission (IEC) as IEC 61784/61158.

In complex automated industrial systems, there is the need to use structures in hierarchical levels as distributed control systems (DCS). In this hierarchy the upper levels for production managements are linked to the direct control level of Programmable Logic Controllers (PLC) via a non-time-critical communication system such as Ethernet. The fieldbus links the PLCs of the direct control level to the components in the plant of the field such as sensors, actuators, motors, lights etc. This way of connection allows us to replace direct connections via current loops or digital Input-Output (I/O) signals.

2.2.5.1 – Ethernet for Control Automation Technology (EtherCAT)

EtherCAT is an Ethernet-based fieldbus system, invented by the German company Beckhoff Automation. The protocol is suitable for both soft and hard real-time computing requirements and it is standardized in IEC 61158. The goal during development of EtherCAT was to apply Ethernet for automation applications that require short data update times with low communication jitter and reduced hardware costs.

This protocol is used in topologies similar to Daisy-Chain (Linear) Topology where each node is only connected to two adjacent nodes, with a slight difference that; instead of leaving the two ends of this line at the start and end, we create a loop that is similar to Ring Topology but not exactly the same. As we discussed previously, the Ring Topology (fig 7) connects the two ends of a Daisy-Chain Topology directly to each other whereas EtherCAT (fig 8) loops back on itself by having every connection between the nodes as bi-directional. So when a message reaches to the last node, it is being sent back to the node before itself instead of the first node, going node by node back to the first one (there is no direct connection between the two end nodes). This full cycle takes a very short amount of time and more importantly, when implemented correct it can make sure that any given message will take a constant amount of time to arrive at its recipient.

When it was being developed, the aim of EtherCAT was to apply Ethernet for automation applications which require very low cycle times (=<100µs) with low communication jitter and low hardware costs. The standard Ethernet packet or frame is not sent, received or interpreted as process data at every node but instead nodes read and modify the part of the packet containing the data addressed to them while the whole message frame (a.k.a. datagram) passes through the device, allowing to process data “on the fly”.

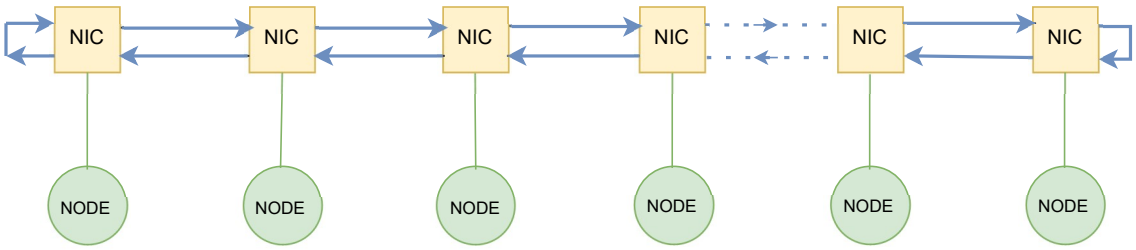

Figure 11: EtherCAT Topology

3 – Problem Definition

In this thesis, we will implement a simulation of a long armature linear motor elevator system with multiple cars using three proposed network topologies with realistic simulation of the real-time computer nodes and communication protocols in the network to observe the system can handle the given task or not.

3.1 – Long Armature Linear Motor Elevator System

The starting point of this work was to implement a network topology for a linear motor elevator with multiple elevator cars to create an alternative to the elevator system we currently use. Since there will be multiple elevator cars on the same shaft there can be no electrical connection reaching to the elevator cars. This means we need a system configuration where the electrical coils are on the stator side, and the best solution for this, is to use long armature linear motors.

This experiment aims to observe if the proposed topologies can handle multiple elevator cars on the same network. We need to ensure the message transmission between sensors and motor drivers are within requirements to move each elevator car at a certain speed. This will prove if the proposed topologies and protocols can be later applied to a real skyscraper with long armature linear motor elevator with multiple cars. The long armature linear motor elevator system consists heaps of linear motor actuators and motor drivers controlling them. As explained in 2.3, hard real-time systems are very sensitive against delay and missed deadlines, so controlling many motor drivers within a very small margin of error is critical. We need to use a protocol that allows high number of nodes as well as low latency and high response rate which is emphasized in 2.2.

MATLAB Simulink's TrueTime library allows us to use any communication protocol we need on any custom network topology we can create on Simulink and with changing some parameters on creating the network we can also simulate multiple elevator cars running on the same network, with a few exceptions that are going to be explained following sections.

3.2 – Custom Network Topologies

The first part of this experiment has been considering various network topologies and communication protocols to decide which topology-protocol couples can make sense and can be used to meet the requirements. For example, we can use a simple bus topology with Ethernet protocol; adding every motor driver and position sensor as a node on the network then also add the main computer and we have a network that can work but we have other requirements, such as easily maintainable and adjustable. To achieve these requirements as well we can add one more network and a gateway in between these two networks and this is how we get Topology G (Global). The second network allows us to off-load the main computer and if we ever want to change settings or when we need to maintain we can connect to that network to handle. This topology being very simple comes with disadvantages such as the massive number of nodes creating problems such as bandwidth and increased response time (latency) when main computer sends a message to a node. To solve this, we need to approach the problem in a different way; we need to localize the motor drivers into small groups with gateway computers. We will be calling these groups including a certain number of drivers and a gateway computer on a separate network, a "Motor Section".

By applying this we end up with a network topology like Topology L (Local). This will be dividing the number of nodes -the main computer sends a message- to the number of motor drivers per motor section. So instead of directly sending a message to each and every motor driver one by one, it can send every gateway computer that can relay the message to five motor drivers hence the number of nodes the main computer needs to transmit is divided by five. To overcome each problem we had, we made small changes on the basic network we had and with trying to keep the simplicity we created this custom network topology we needed. A similar approach has been examined previously in [23].

This second network topology is created by solving the problems occurring in the first topology but there is still a chance of messages colliding while multiple nodes start transmitting at the same time. There is, of course, a solution implemented in Topology L to deal with this problem which is going to be explained in detail under section 3.4.2, but

there is a better solution to get rid of the collision problem by design. Instead of using a bus type topology that involves multiple nodes on the same network, where each node can transmit on the network that can lead to collision, we can have a ring type topology where there is only one message as a message frame (datagram) which allows each node to read/write messages from/onto it and pass to the next node so at any time during the system is active, there is only one message being transmitted. The third and final topology we cover on this thesis, Topology R (Ring), is using this method and will be explained in detail in the following sections.

In all of these topologies we implemented a special sensor node that is allowing us to measure the position of the elevator car so that the system can precisely control the right motor driver at the right time allowing the elevator to move smooth and safely. In topology G we can see this sensor node every 5 motor driver and on Topology L and R every gateway computer have a sensor node connected to it. We designed the implementation in such a way that there is a linear encoder (transducer) as the sensor and there is a scale (strip) which the encoder can move in front of and encode the position data. As the elevator is cable free, the sensor is positioned on the wall with motor drivers and the scale is attached to the elevator mover. By keeping the scale longer than the height of the elevator on both sides, we can start measuring the position of the elevator before the elevator car enters the level of the specific motor drivers of that level which guarantees that either side of the level the elevator car approaches from, the sensor can measure the position. This also allows us to separate the motor drivers into groups as Topology L and Topology R does, simplifying the communication as the gateway computers can but do not have to communicate with each other for position transfer.

3.3 – Proposed Topologies and Protocols

In real-time systems, communication delay is an important concern, specifically the amount of time from a certain event to system response. Within specified time constraints which are called “deadlines”, real-time systems must guarantee response. There are three real-time systems based on how important it is for the system not to miss its deadlines: soft real-time, firm real-time and hard real-time.

Hard real-time systems -by definition- must not miss any deadlines, or it is a system failure. This scheduling is used in highly critical systems where a system failure results in a loss of life or property. We can think of this as the value of a task is 100% until the deadline and if deadline is missed the value of the task completion is minus infinity (-∞).

An example for a hard real-time system in real life can be a missile trajectory controller; if the controller misses even one correction movement or responds too late, the missile can end up kilometers away from its original target, putting human lives or general valuable properties in danger. There is no way to take it back or undo.

Firm real-time systems allow missing deadlines infrequently. In this scheduling method, the system can survive missed deadlines as long as they are adequately spaced, this means the task that missed its deadline has no value anymore, but the system can still operate. We can think of this as the value of a task is 100% until the deadline and if the deadline is missed the value of the task completion is zero.

An example for firm real-time system in real life can be a manufacturing robot in production line; if the controller misses one of the deadlines in movement ending up mis-assembling one product in the line, that specific product is going to be noticed in quality control and get recycled while the system can still operate. The value of that particular part suddenly got to zero, but the production continues. This is valid if the controller does not miss deadlines too often, and ruined parts are not that many.

Finally, soft real-time systems can frequently miss deadlines, but it is better if they do not. In soft real-time scheduling, the system can miss the deadline but if the task is then completed within a short notice, there is still value for that task. We can think of this as the value of a task is 100% until the deadline and then if the deadline is missed, the value of the task completion decreases with time until reaches zero (for most cases).

An example for soft real-time system in real life can be Bluetooth speakers to play music from; when we start the music to play on our phone, if the speaker misses the deadline to start playing the music, the task is not discarded and it is not of zero value, it is simply late and it can still execute and start playing. This does not affect the rest of the song or the

experience, it responded late so the quality of service diminished but is not completely disappeared. Like this behavior especially noticed when the speaker is running on low power as the range of the Bluetooth decreases and the responses start to take longer and some of which miss their deadlines.

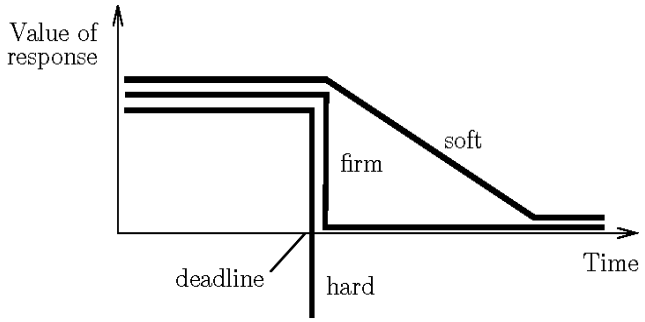

Figure 12: Real-Time Scheduling Deadline Vs Value [24]

Our linear motor system can be thought as a hard real-time system, there is no room for delay and if any task misses its deadline, since it would jeopardize the transition of the elevator car from one motor driver to the next, the task instantly drops, and the system sends an emergency signal to the main computer that halts the movement and brakes the elevator. This of course is to eliminate any chances of putting people or valuable equipment transported by the elevator as well as the elevator itself in danger.

The following proposed topologies aim to decrease the delay and have a higher response rate to satisfy the requirements of a hard real-time system. We will be discussing their advantages and disadvantages over the others in this section.

3.3.1 – TOPOLOGY G

This topology is a simple bus topology which contains a main computer, a gateway computer, sensor nodes and motor drivers. The main computer is responsible to create and send the reference position to the gateway computer which is then distributed to all motor drivers, sensor nodes get the real position data and it is sent to the relevant motor drivers as well, then motor drivers combine these two information and calculate the motion and then apply the necessary current to the required section of the motor at the required time which moves the elevator car. In Figure 13 the structure of the topology can be seen. There are two different networks, one connecting the main computer to the gateway computer which is called “outer network”, and the other connecting the sensor nodes and motor drivers to the gateway computer which is called “driver network”. In the figure main computer is labelled as “MC”, gateway computer is labelled as “GWC”, sensor nodes are labelled as “SN” and finally the motor drivers are labelled as “MD”.

From a real-time system requirements point of view; main computer’s response does not have to be hard real-time as, the reference does not change too quickly and milliseconds of delay to the user’s input is not critical. But the sensor nodes’ and motor drivers’ responses have to be hard real-time as sensors need to send the measurement to the motor drivers which also affects the motor driver performance, and motor drivers need to respond within the acceptable amount of delay so that the elevator ride is smooth and stable.

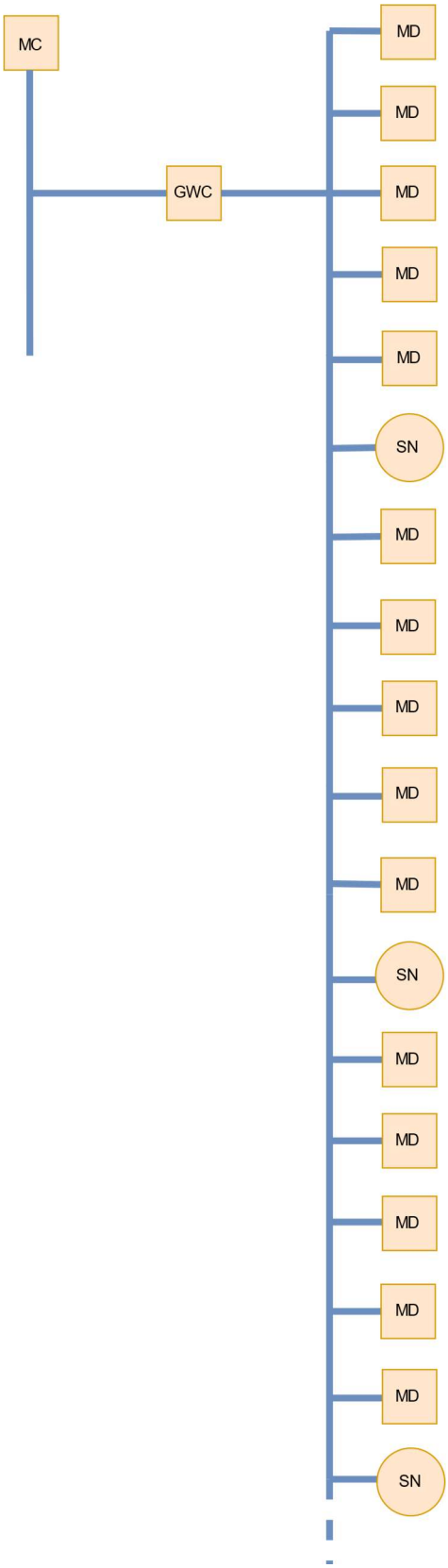

Figure 13: Topology G (Network Model)

3.3.1.1 – Suitable Communication Protocols for Topology G

Since Topology G has hundreds of motor drivers and sensor nodes, we thought the most stable communication protocol to handle that many nodes is Ethernet. Using Ethernet allows to keep the network topology simple like a bus topology as well as allowing easy add and remove node operations if necessary. We have a point-to-point connection between main computer and gateway computer, so it does not have a major effect on the rest of the network which protocol we choose unlike driver network. Many different protocols are suitable for the outer network.

3.3.1.2 – Advantages of the Topology G

Motor driver nodes can be implemented with simple computers, they are not expected to be loaded heavily with tasks; there is a simple position reference to real sensor position difference calculation and using the end result of this calculation to send the signal to drive the motor. Having specific nodes just for sensors, while increasing the number of nodes on the network, off-load other components such as motor driver nodes or gateway computers.

3.3.1.3 – Disadvantages of the Topology G

There are two disadvantages of this topology that majorly affect the performance; the first one is that the number of nodes on the network is high and as the number gets higher, the constant bandwidth that gets shared between the nodes gets lesser and lesser. Every time the main computer transmits a message, the message is sent to the gateway and gateway spreads it to every motor driver node one by one, as there are more nodes on the network, one cycle of spreading the message to every motor driver takes longer time. The second disadvantage is that when multiple elevator cars move simultaneously, bandwidth needed may become less than the amount Ethernet can provide. This limits the number of elevator cars that can run at the same time.

3.3.2 – TOPOLOGY L

This topology is a hierarchical system containing a main computer, gateway computers and motor drivers. The main computer's job is the coordination of the motion in general, it is connected to the motor driver groups collected under gateway computers. Gateway computers have a few tasks; one of which is to act as a bridge between the main computer and motor drivers, it also is connected to the position sensors so it gets the mover position and use this data in calculations, which lead to driving the right motor drivers. Gateway computers also pass position information between each other when it is required. Finally, there are motor drivers which are connected to the linear motor coils and they generate the signal that motor drivers require to move the elevator car. All these nodes are placed in different networks in different ways. There are three different types of networks, each one is composed of different nodes. First one is the Outer Network; it is the network that connects the main computer to the gateway computers. Second one is the Gateway Network; which connects the gateway computers to each other, but every connection is between only two gateway computers, so it creates a daisy chained network between gateways computers. Finally, there is Driver Network which allows communication between a gateway computer and motor drivers.

To understand better we can analyze each of the node types from a real-time requirement point of view with the tasks they do:

Main computer sets the position reference for the mover to go then create the plan of that motion and finally transmit this as a message to the related motor drivers through gateway computers. It sends the necessary messages to the gateway computer on the outer network and after processing this the gateway computer transmits to the right driver.

Gateway computers get the position reference data from the main computer and actual position data from the sensor that is connected to it and calculate which motor drivers need to work to move the elevator car and then send the message via driver network to the corresponding driver in the right time for a seamless operation.

Motor drivers receive the messages from gateway computer and start or stop driving the part of the motor they are connected to so that they move the elevator car. Motor drivers also calculate the amount of time passed from the task call until the execution of the task which lets us calculate the delay introduced to the system by the communication protocol.

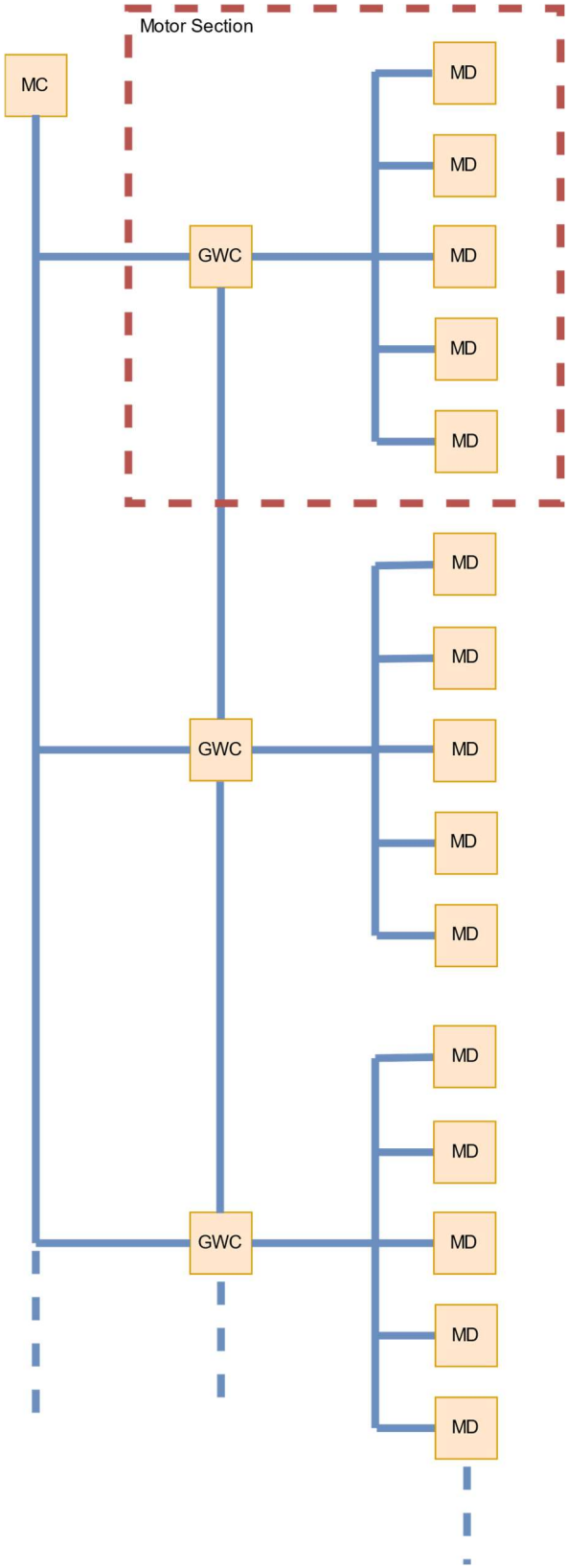

Figure 14: Topology L (Network Model)

3.3.2.1 – Suitable Communication Protocols for Topology L

As this network topology is going to be capable of serving for a very tall building, the linear motor will have so many motor drivers and connected to them will be so many gateway computers. To be able to guarantee that everything is going to work smoothly, there are some requirements that the system needs to meet. There will be limited number of drivers connected to each gateway computer with very low latency demands, so we chose the most suitable communication protocol for the driver network as CANBUS protocol. For the outer network however, we do not have hard real-time network requirements, but we have hundreds of gateway computer nodes that need to be connected and communicated on the same network, that's why we chose Ethernet protocol for outer network.

3.3.2.2 – Advantages of the Topology L

The main advantage of this topology when compared to Topology G is separation into smaller networks with gateway computers. Since there are not many drivers on each of the motor sections, we can trade in more node support to low latency. As the number of nodes that main computer needs to communicate significantly dropped compared to Topology G, (since once the message is sent to gateway computer, main computer can send to the next gateway computer while the first gateway computer can distribute the message to motor drivers) the amount of load created on outer network decreased significantly. Moreover, since now there are gateway computers connected to motor drivers grouping them, outer network is not affected by the message loads that are sent to motor drivers. These messages are sent on the driver network from gateway computers to motor drivers.

3.3.2.3 – Disadvantages of the Topology L

Since there are driver networks with gateway computers connected to each one, no need to attach the position sensor to another computer as a separate node but it can simply be connected to the gateway computer, decreasing the number of nodes on the network, with the price of adding some more load on the gateway computer. Having gateway computers as an extra layer between motor drivers and main computer adds a slight delay but it is unavoidable to have some delay on the system and as long as it does not make the system unstable or uncontrollable, it is acceptable.

3.3.3 – TOPOLOGY R

Similar to the Topologies G and L, Topology R consists of the same nodes with an addition. These nodes are;

- main computer; which manages the motor drivers by sending messages periodically to every gateway computer,
- gateway computers; which relay the messages coming from the main computer to the motor drivers and being positioned in-between motor drivers and the main computer, they group motor drivers into more manageable numbers,
- sensor nodes; which obtain the position data of the elevator car and passes this information to the motor drivers,
- motor drivers; which drive the actual elevator car with respect to the information coming from the main computer messages and the sensor input,
- and last but not least the network interface controllers; which take charge in the communication of the afore mentioned nodes.

As explained in section 2.2.5.1 Topology R is a 2-level hierarchical system with the lower level being a Ring Topology. In the lower level there are Network Interface Controllers (NIC) each connected to a critical piece of the overall system, the first connects to the gateway computer (GWC), second connects to the sensor node (SN) and the rest of the NICs (5 of them) each connect to a motor driver (MD). Moving to the higher level part of the hierarchy we have an outer network which allows us to connect the main computer,

which makes sure all the systems work in coordination, with the gateway computers that connect to NICs to relay the messages from main computer to motor drivers.

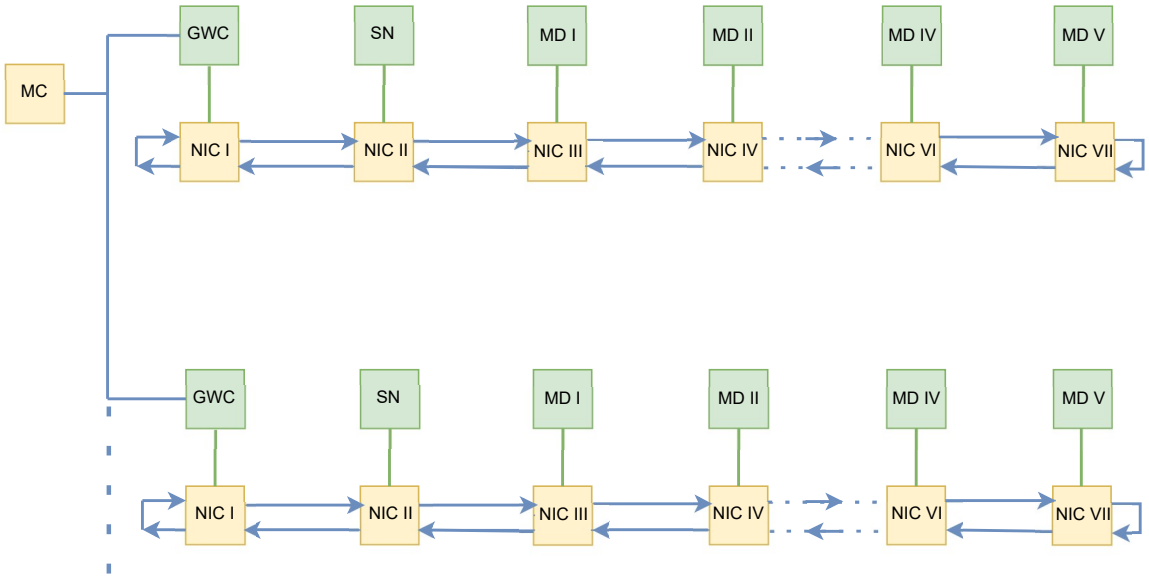

Figure 15: Topology R (Network Model)

As an example to how it works, we can follow the steps as it progresses assuming the elevator car is reaching in front of the 4th motor driver of the first gateway;

- in the beginning of the cycle, the gateway computer checks if there are any messages coming from the main computer to be passed to the motor drivers, if there is, it is transferred to the NIC of node, connected to the GWC (if not the NIC prepares an all empty message frame (datagram)).
- Then this node's NIC sends the whole message frame to the next node's NIC which is connected to the sensor node. The sensor node measures the elevator position and then calculates which motor driver is responsible for that position to actuate. It is the 4th motor driver, so the sensor node puts this position data measurement on the message frame's respective empty space.

- Then this node's NIC is done and the frame is sent to the next node's NIC. From this node forward, all other nodes are motor drivers, so they are most likely not going to add/edit any data on the frame but rather read from it. So this NIC of node checks its respective position in the message frame (but this node is the 1st motor driver) and there is no data to read so the NIC of node passes the message frame to the next,
- the 2nd motor driver checks and there is no message for it either,
- the 3rd goes the same way as well and finally the NIC connected to 4th motor driver checks its respective position and gets the position measurement data coming from SN and drives the motor accordingly.
- The message frame goes to the next and final node's NIC and that motor driver does not get any data and the communication to all nodes are done but still the message frame needs to come back,
- so the last (7th) NIC sends it back to the previous, and 6th to its previous and so on until it reaches to the beginning for a new cycle.

Additionally, if there is a message coming from the main computer there is an extra space for gateway computer to add on the message frame and every motor driver reads this message independent from if they had a message from the sensor node or not. This whole process including the message frame to come back to the first node's NIC to start a new cycle takes around 115 – 120 µs.

3.3.3.1 – Suitable Communication Protocols for Topology R

The higher level can use Ethernet for connection as it has a very similar structure to the Topology L, while the lower level simulates an EtherCAT communication network which also uses Ethernet as foundation, but some requirements and specifications are different from Ethernet. Using EtherCAT for the lower level communication for connecting sensors, actuators and other computers is very common in industry as it can guarantee hard real-time distributed control systems.

One of the biggest differences between Topology R and L is that in Topology L, the nodes are communicating with each other on the network where in Topology R there are controller hardware with its own processing unit specifically used for the network communication side of the system so that each node is left with more free processing time for other calculations and other tasks.

3.3.3.2 – Advantages of the Topology R

When listing the advantages of Topology R, the most important one is that it is built to have constant delay as the messages on the network are periodically transmitted from one node to the next; and all sensors, actuators and gateway computers are connected to network interface controllers, allowing more processing time left for computations instead of using some of it for communications since there are specific hardware (NIC) for it. This assures that the network is always going to be on a certain schedule.

Another advantage of this topology is, since it is Ethernet-based the NIC modules that handles the communication are cheaper compared to other industry standards and it is a lot simpler to track fault compared to a bus topology communication system.

3.3.3.3 – Disadvantages of the Topology R

One of the disadvantages of this topology is that it requires very low cycle times (period) for the message frame to start from a certain node, travel all nodes and come back to the same node. This is easily achievable on professional hardware such as the industrial systems' use but on MATLAB it is more challenging given that the simulations have an upper limit on how quick the nodes process and how much bandwidth can be simulated. It

can be handled via decreasing the number of nodes on the network so that the message frame needs to travel less nodes to complete a full cycle.

One other disadvantage compared to bus topologies is that it is easier to spot an error or an issue but it is much harder to fix it and it can directly cause problems to rest of the network as this is a daisy-chained communication system if a network interface controller stops working (malfunctions) the rest of the NICs and therefore the rest of the nodes would be cut away from the network.

4 – Simulation Models and Environment

Networked control system (NCS) is a closed loop control system where the loops are complete through the network [26]. It is composed of a mixture of continuous time-driven dynamics and discrete event-driven dynamics. Both of those dynamics can be affected by delays in the system. To be able to measure the amount of delay and other critical timing operations in the system we need a software tool where we can create a tool to simulate and analyze this [17]. We need to be able to measure timing of tasks within the nodes as well as the timing of communication between nodes to observe how it affects the control performance [18].

4.1 – TrueTime

TrueTime is a MATLAB/Simulink based library allowing networked control systems and embedded systems to be studied and simulated. It is developed at Lund University in 1999. TrueTime creates great opportunity for the user by enabling real-world continuous dynamics work alongside computer architecture such as task execution and network connections co-simulated to observe the interaction between the two.

TrueTime consists a library of various blocks each can be added to a Simulink system and configured in detail. These blocks are; TrueTime Kernel Block, TrueTime Network Block, TrueTime Send Block, TrueTime Receive Block, TrueTime Battery Block, TrueTime Wireless Network Block and TrueTime Ultrasound Network Block as we can see the whole library in Figure 25. Any TrueTime block can be connected to any ordinary Simulink block to also be able to co-simulate a real-time control system.

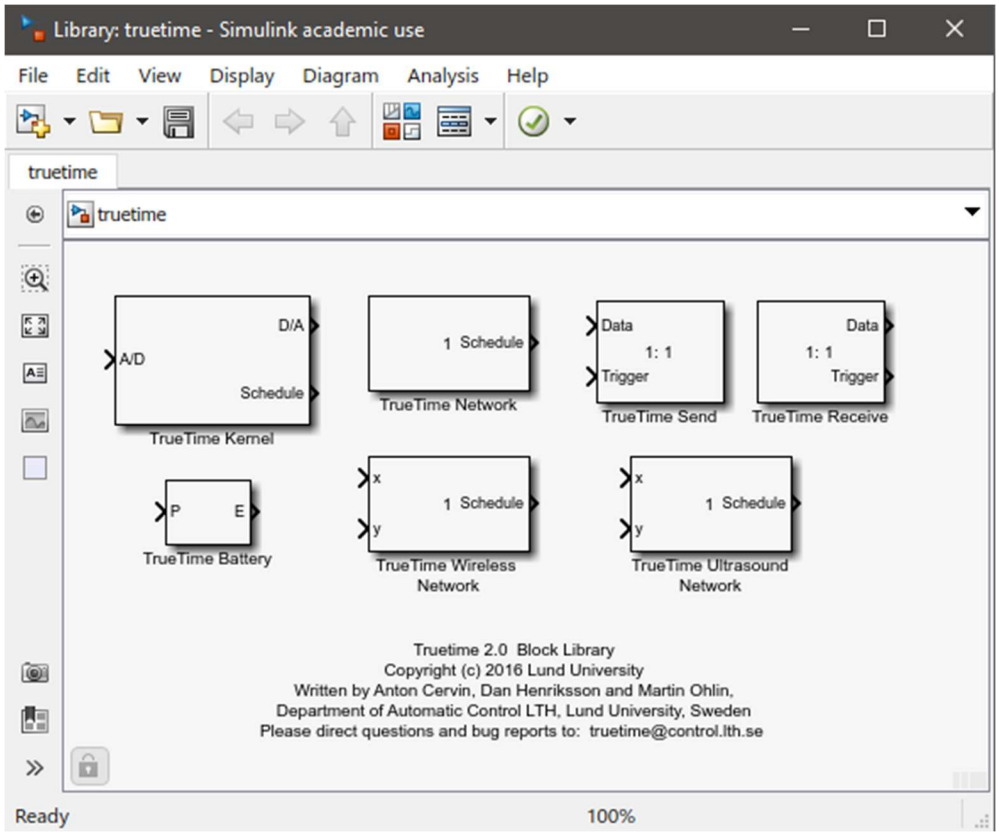

Figure 16: TrueTime Library

4.1.1 – TrueTime Kernel Blocks

A TrueTime kernel block is essentially a computer node, which simulates a generic real-time kernel as well as network interfaces to connect to and communicate through a network. For a kernel block to work after adding it to our Simulink simulation we need to configure it. To configure we have an initialization script which allows us to create tasks, timers, interrupt handlers, etc. These objects are continuously called by the kernel. For our initialization scripts, we can use either C++ or MATLAB m-files. In a kernel block we can use a variety of scheduling algorithms such as, fixed-priority scheduling, earliest deadline first scheduling, deadline monotonic scheduling or a custom scheduling algorithm [19].

As we can configure a kernel block while we are creating it via a script, we can also use the block subsystem mask dialogue box after we have created the model on Simulink. As we can see on Figure 26, we can edit; initialization (init) function name, initialization (init) function argument, number of analog inputs and outputs the kernel has, number of external triggers, network and node numbers, local clock offset and drift. The important ones that we are going to be using are; init function name, init function argument, analog input and output ports and most importantly network and node numbers. (see Appendix A)

4.1.2 – TrueTime Network Blocks

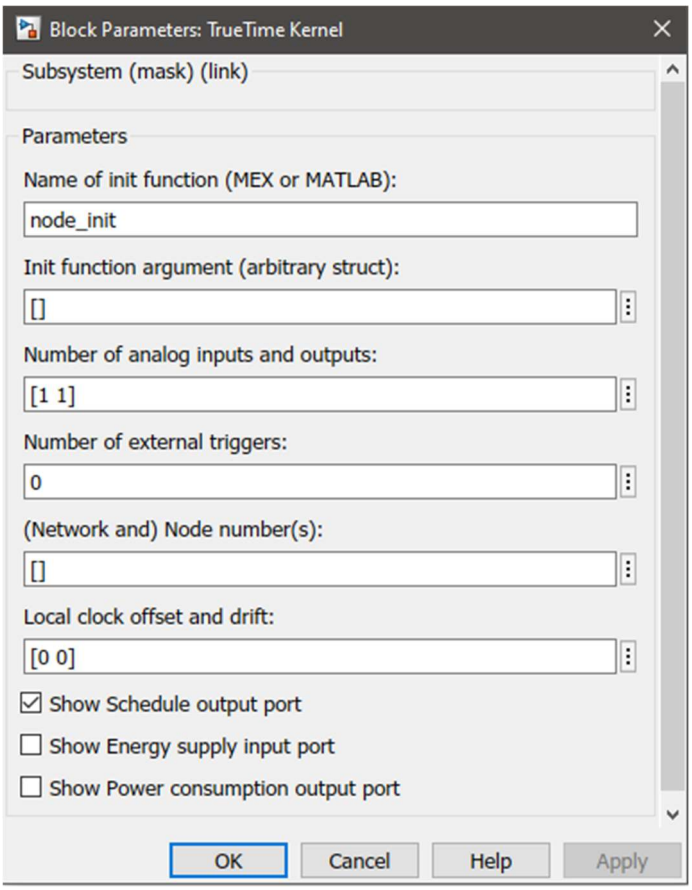

Figure 17: TrueTime Kernel Block Configuration

TrueTime network block simulates medium access control and packet transmission in a local area network (LAN). Every time a node starts transmitting, a trigger signal is sent to the network block, and when a node finishes transmitting, a new trigger signal from the network block to the receiving node is sent. This transmitted message is saved in a buffer at the receiving computer node. [20]

Similar to a TrueTime kernel block, we also need to configure a TrueTime network block. The configuration process is very similar, we can either use a script while creating it, or we can use the block subsystem mask dialogue box that we can open after we create. There are some parameters that are common for all networks such as; network type, network number, number of nodes, data rate and minimum frame size, as well as some parameters that are specific on the type of network such as; transmit power, receiver signal threshold in wireless networks as we can see in Figure 27. There can be more than one network block in a model that is why we use network numbers to be able to identify them. Every node connected to the network has a specific node number within that network. (see Appendix B)

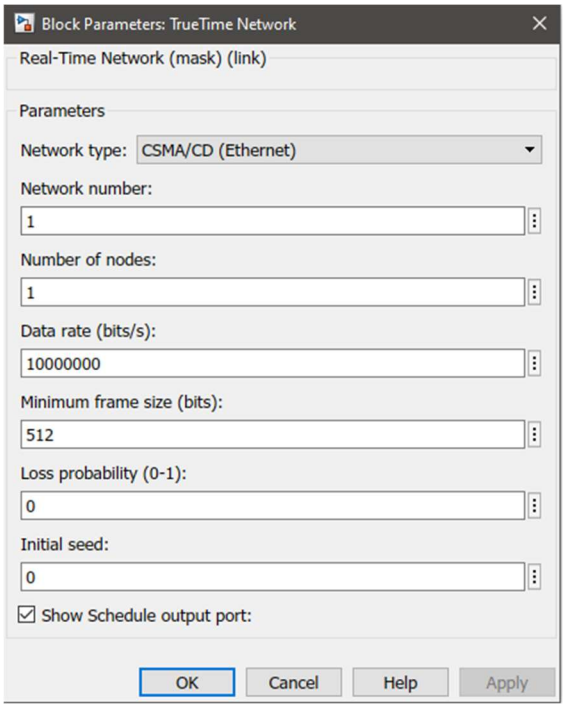

Figure 18: TrueTime Network Block Configuration

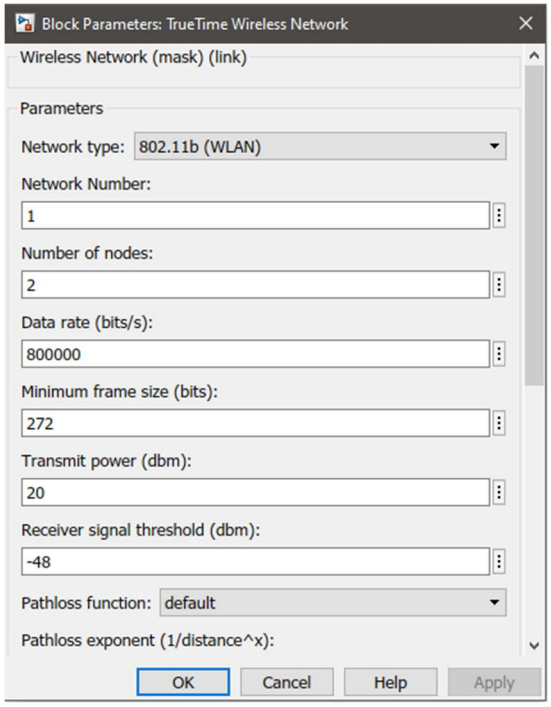

Figure 19: TrueTime Wireless Network Block Configuration

4.2 – Creating Simulink Network Models using MATLAB Scripts

While creating a linear motor for a very tall building, we need to use some nodes and modules in a re-iterative fashion. As we want to compare the performance of two different network topology, we need to create those topologies' network models, communication protocols and models. This thesis discusses the development of Simulink models that are created by using MATLAB script-based methods which makes it especially easier in repeating situations such as this one.

It allows us to create a parametric configuration file that keeps all the settings we want to set, before we want to create the topology and tune them as we wish. Since we need to meet some requirements, determining an optimum performance by tuning the parameters is crucial for the project. (see Appendix C)

4.3 – Creating EtherCAT Model using MATLAB Scripts

Even though the MATLAB Simulink library we use, TrueTime, does not have EtherCAT built in as a protocol, this does not mean we can not create this topology on MATLAB. In fact we can create our own EtherCAT protocol on a ring topology from kernel and network blocks that are built in the library but it requires a few simplifications. More detail can be found on Appendix D section at the end of the thesis.

The original EtherCAT is designed to be cheap, simple and easily maintainable (easy to fix in case of any problems), which is possible thanks to the industry level products as EtherCAT is used in industrial level works such as factories. In our simulation we assumed that there is no random packet losses or corrupted packets as this affects this topology specifically to the extent that it stops completely. A fair warning at this point: this does not mean that this topology is not compared by the same circumstances, and there can still be packet losses and missed deadlines if the system is not capable of keeping up with the messages or if there are collisions on messages. We only restrain the system in such way that there is no external disturbances as it would directly stop/kill our simulation implementation but not a professional application of EtherCAT.

This is easily handled via software in professional solutions with algorithms that realizes the system has stopped or even better, it does not even let it stop but since we would like to study and research the network side of this topic and that such algorithm would take so much time to implement on our simulation we assumed there is no random packet loss or random message corruption. There is only one key idea that this simulation keeps running that is one network interface controller sends message to the following one, so the next one is triggered and the handler gets a function call and gets activated (this is basically a short coming of the library we use, and our simpler implementation of EtherCAT). As described in section 3, all communications for motor drivers are handled by the network interface controllers (NIC), which by design allows no conflicting messages as there is a single message datagram carrying messages from and to nodes, step by step. As we have 7 NICs per gateway computer which requires 12 transmissions to complete a full cycle for a message and set this period for the NICs around 10µs, we get roughly 120µs as the period of the full cycle which we can see on figure 20.

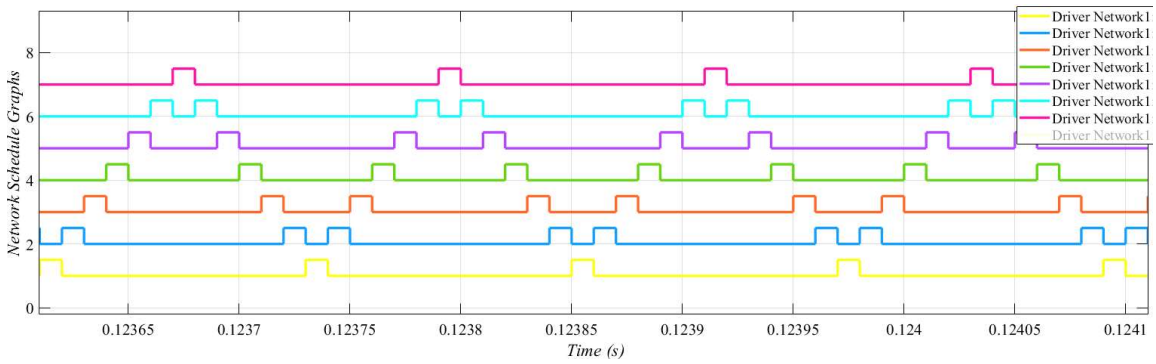

Figure 20: Topology R 4_20 Network Scheduling

As we can see, when datagram reaches to a NIC, even if there is no message contained for that node connected to that NIC, it still triggers that NIC to call the handler and pass the datagram to next NIC. This goes on until the last (7^{th} on this topology) NIC is reached, and then last NIC sends the message backwards to 6^{th} and 6^{th} to 5^{th} and so on which creates this "V pattern" until the datagram comes back to the first NIC and cycle completes.

5 – Simulation Results

Computer simulations usually try to be as accurate to real life as possible, but there are always limitations; sometimes this is software limitations and other times it can be hardware limitations. This is not a problem though; this is usually expected, and we need to create the simulation as close to real life as possible within the limitations. In our system we have some limiting factors such as huge number of nodes and other elements that will include hundreds of real-time computers and networks, and simulating that on MATLAB on a single computer(given that an average spec computer) is not possible, but with that said, we can still run a realistic simulation within our boundaries. Our real system that we are trying to create a model for is a skyscraper elevator system for a 200 meters tall building, as one motor driver is only 9cm (0.09m), we should have approximately 2220 motor drivers, 444 gateway computers and networks. These are huge numbers to simulate in MATLAB Simulink, so we create a smaller model that can still fully represent the real system and its problems. In our simulations we have tested many different size models and decided on two sizes that needs to be mentioned. We will go through each one of all three topologies and see how the size of the model affects the simulation and its results.

In our simulation, the control loop had a frequency of 10 kHz and a period of 100μs(0.000100s). We ran the simulation with various speeds for the elevator car and we will be looking at 5 m/s, 10 m/s and 20 m/s. These may sound like unusually high speeds but when working with a very tall building such as a skyscraper which is 200 meters tall, for the elevator to go from entrance floor to the top -200m-, with 5m/s moving speed the elevator takes 40 seconds to get to the top! With 10 m/s and 20 m/s, it is 20 seconds and 10 seconds respectively. We can compare that to a standard elevator speed that varies between 0.5 to 2 m/s. Considering that, the same 200m trip would take 100 seconds which is longer than a minute and a half. That is a great improvement as long as the elevator can manage to keep stability network wise and that is what we wanted to test.

As we went over it on section 3, we have three communication protocols that we use in our system: CSMA/CD (Ethernet), CSMA/AMP (CANBUS) and Fieldbus (EtherCAT). The first topology is based on Ethernet alone, where the other topology contains both Ethernet

and CANBUS, and the last one includes both Ethernet and EtherCAT. All three topologies have one common property that is, in all topologies the outer network that connects the main computer to the gateway(s) is Ethernet. The rest differs, as Topology G uses Ethernet for the whole system where Topology L uses CANBUS between motor drivers and gateway computers and Topology R uses EtherCAT with Network Interface Controllers connected to our actual nodes. On outer network side, the Ethernet protocol is used to convey messages carrying system information from main computer to gateway computer(s) with a period of 0.01s. In our MATLAB scripts, task execution takes 10 μs for gateway computers and motor drivers.

We assume that the messages are consisting;

Topology/Network	Outer Network	Driver Network
Topology G	125 bytes (Ethernet)	46 bytes (Ethernet)
Topology L	125 bytes (Ethernet)	8 bytes (CANBUS)
Topology R	125 bytes (Ethernet)	125 bytes (EtherCAT)

- 46 bytes of data which is composed of several information across the whole network including position and velocity data on Ethernet protocol for driver network and 125 bytes of data on Ethernet protocol for outer network on Topology G
- while 8 bytes of data which is composed of position (4 bytes) and velocity (4 bytes) on CANBUS protocol, 125 bytes of data on Ethernet protocol on Topology L
- and 125 bytes of data capable message frame which consists of position, velocity etc. including the messages coming from the main computer for the EtherCAT (which operates as Ethernet) and 125 bytes of data on Ethernet for outer network on Topology R.

	Header	Destination	Source	Length	Data	CRC	TOTAL
CANBUS	44 bits	-	-	-	64	-	108 bits
Ethernet	14 bytes	6 bytes	6 bytes	2 bytes	368 bits	4 bytes	64 bytes

- In our CANBUS protocol implementation on MATLAB, each packet has 108 bits length consisting 44 bits of header and 64 bits of data (2 float numbers each of which is 4 bytes),
- where in Ethernet protocol each packet has 512 bits length of packets that is composed of 14 bytes of header (6 bytes destination, 6 bytes source, 2 bytes length), 4 bytes of CRC (cyclic redundancy check) and 368 bits (that is the minimum payload size for Ethernet standard) of data.
- CANBUS is set to 1 Mbit/s that is the maximum data rate and
- Ethernet is set to 100 Mbit/sec that is the standard rate we use for Ethernet today.
- On Topology R the same values for Ethernet are used for EtherCAT as well since the infrastructure of it originates from normal Ethernet with the addition of EtherCAT being scalable and not bound to 100Mbit/sec,
- a future extension to Gigabit Ethernet makes it possible to set up a Gigabit EtherCAT as well but even in industrial usage it does not exist simply because 100Mbit/sec is highly sufficient.

5.1 – Topology G

As we discussed in section 3, in Topology G we have a main computer connected to a gateway computer through Ethernet protocol which we named the network as “outer network” and then the gateway computer is connected to motor drivers and sensor nodes via another Ethernet protocol which we call as the driver network. In simulation we can see two network traffic going on, one of which is the main computer sending high level operation information to be distributed to every motor driver one by one, and the other is the overall traffic created on driver network by the gateway distributing the message coming from main computer to every motor driver or by the sensor nodes to the motor drivers if the elevator car is in front of a sensor. Since in driver network there are two separate transmission tasks taking part, there is a high likeliness of some nodes trying to transmit a message at the same time, and when this collision happens one of the nodes need to wait until the other one is done. Of course this is acceptable only if the message deadline is not passed yet, as this system uses a hard real-time scheduling, if we see any node trying to transmit after its deadline or if we see that packets are being dropped, that means the system model is failed. For us to be able to tell if a task is running late or missing its deadline we need to be able to measure the amount of time it takes for a message to go from the sender node until it arrives at the receiver node. In MATLAB to be able to do that, we use timestamps. We mark the exact time on the clock when the task is started and then after it is sent we compare the first timestamp with the exact time at the clock as it is received on the other side; and this difference in the timestamps is our delay, allowing us to see how much time has passed. We know how much time can pass for any task at any given time so we can check if any delay is higher than what is required and call the system failed.

We have tested many situations and in order to check if the proposed topology is appropriate to be used in the real system, we came up with three cases to observe. These are: small size network; to see if the system is capable to handle the communication at all, then a larger size network; to see if there is any correlation between the size of the network and its performance, and finally a multi elevator car test on a mid-size network; to see if the topology can handle simultaneous runs by multiple elevator cars.

There are a few important notes that needs to be expressed in order to have a better understanding of the results and to be able to interpret the graphs easier; one of which is how to read the node transmission graphs, such as Figure 22. To be able to understand Figure 22 we need to know what a low, medium and high level signal means; a low level signal is, as one can guess, there is no activity on the channel, medium level signal means that the message is ready to transmit (message is prepared, but not transmitted at that moment), this usually happens when the communication medium is too crowded that there is no bandwidth to start transmitting so the node waits until the line calms down or the deadline is missed and packet is dropped, and finally high level signal which means the message from that node is being transmitted.

In our naming sequence, we used “topology name”, “number of gateways”, “number of nodes” followed by the type of the test. We will be following this sequence while naming the figures as well.

5.1.1 – Small Network Test

To see if the topology can be useful for our system, we create a very small network consisting of ten motor drivers, two sensor nodes, one gateway computer and one main computer. Then we simply run the system and our motor drivers print out the amount of delay for each transmission on a text file, so that we can keep the results, and we can also check our scopes and displays to see the messages, delays and scheduling time events to make sure system works without any problem.

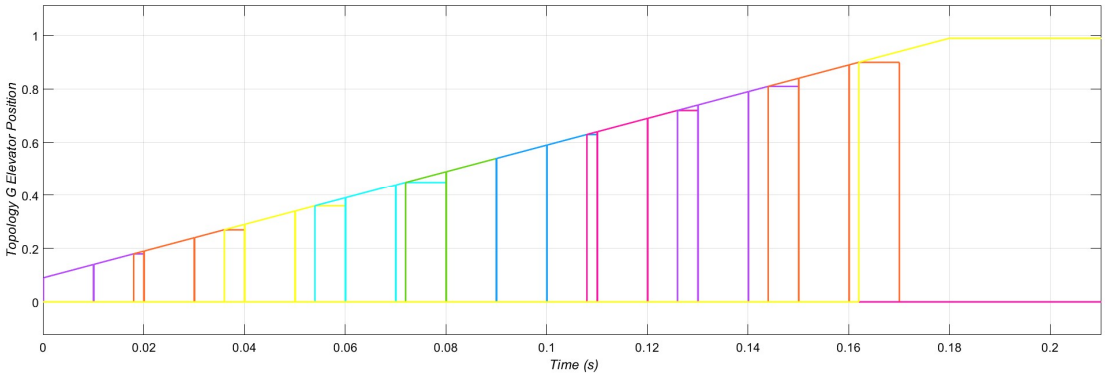

Figure 21: Topology G 1_10 Small Network Sensor Data

In Figure 21 we can see the sensor values delivered to the motor driver and printed on our network transmission sequence. As we analyze the plot, we can see that every motor driver prints in a different color and they all seem to be continuous. This is important as it means none or only negligible number of messages miss their deadline and the system works smoothly.

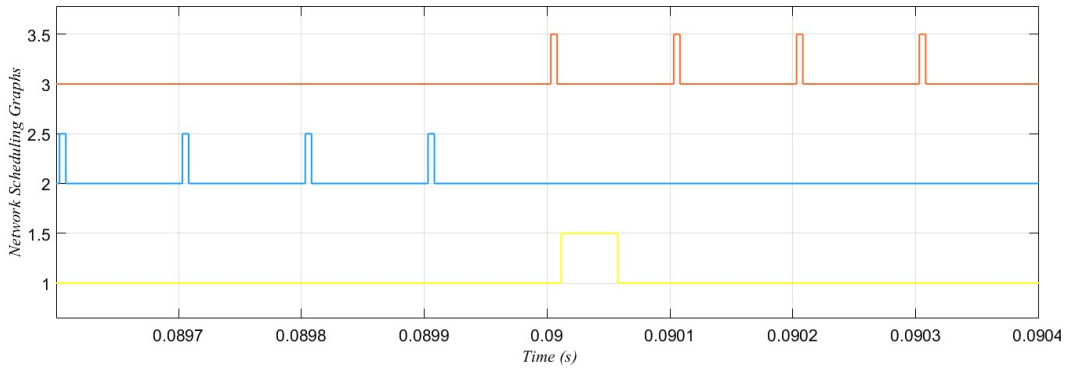

Figure 22: Topology G 1_10 Small Network Scheduling

As we check the scheduling plot of the system on Figure 22, we can see messages seem to be consistently sent and received, which means there are not any problems which can lead the system to fail. We can see the first line (yellow) is the gateway computer relaying messages onto the driver network, the second line (blue) is the first sensor node responsible for the drivers one through five, and the third line (orange) is the second sensor node responsible for the drivers six through ten. As we can see even on the transition moment, we are not missing any messages.

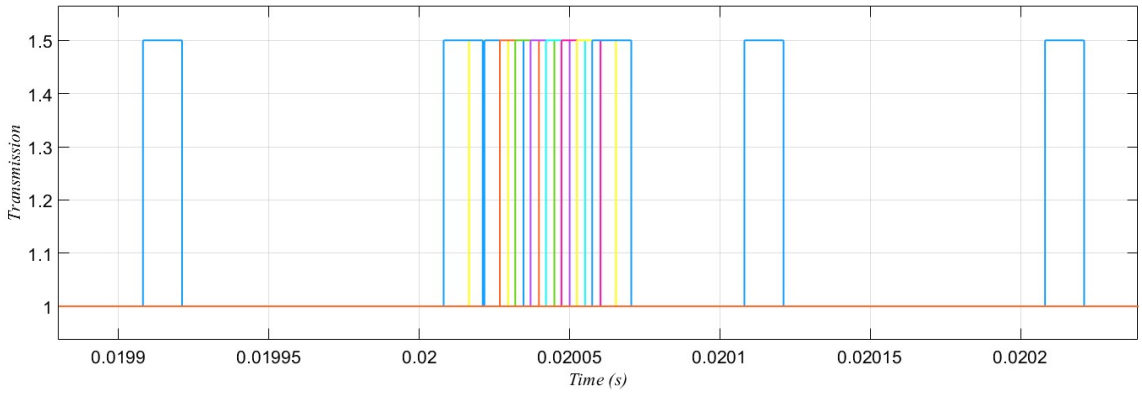

Figure 23: Topology G 1_10 Small Network Transmission

In Figure 23, we can see all transmissions through the network side of view. The blue periodic signals here are the messages from sensor node to the motor drivers. They have a period of 100 µs (0.0001 s) as we have set the task on our MATLAB script. The colorful ones between two sensor node messages is the message from main computer that is relayed by the gateway computer to motor drivers, each one of those colors represent a separate message from gateway computer to one of the motor drivers. We expect to see those signals equal to the number of motor drivers so that we will know gateway computer has no difficulty relaying messages to all motor drivers and we can clearly say that the gateway computer relaying messages from main computer is not causing any other message to miss its deadline. We can check in detail to see exactly how long this transmission section from gateway computer takes, which is approximately 55 µs (0.000055 s).

5.1.2 – Large Network Test

Now to have a simulation that is closer to the real system, let us try to increase the number of nodes. From the previous section, small network test, we have seen that the periodic messages coming from the main computer, relayed by the gateway computer to every motor driver one by one, creates a large period of continuous transmission block.

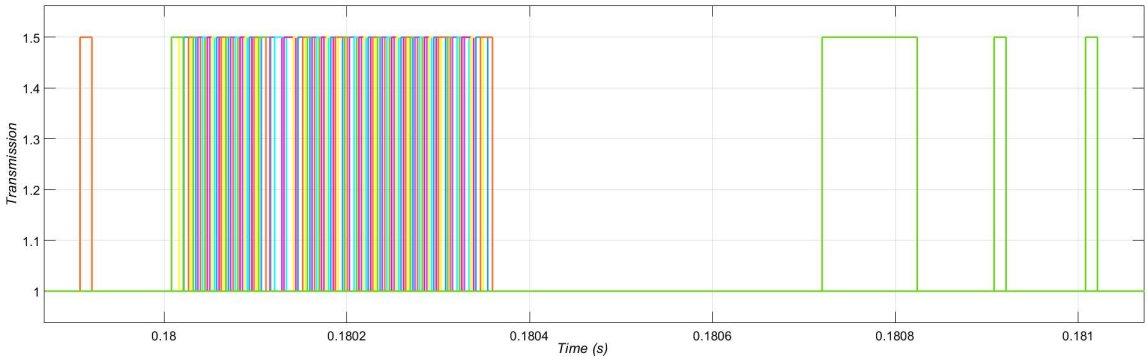

Figure 24: Topology G 1_60 Large Network Transmission

As we can see in Figure 24, with 60 motor drivers on a single ethernet protocol network connected to one gateway computer, the amount of time required to relay the message from main computer to motor drivers is increased with the number of motor drivers increasing. We can compare head to head with the previous topology with ten motor drivers on Figure 23. The amount of time from starting of the transmission to the end takes approximately 350 µs (0.00035 s) which is around 6 times more. As expected, the amount of delay caused by the gateway computer is proportional to the number of motor drivers. This proves that by increasing the number of motor drivers further on this topology, we will be decreasing the performance of the system until it can no longer operate.

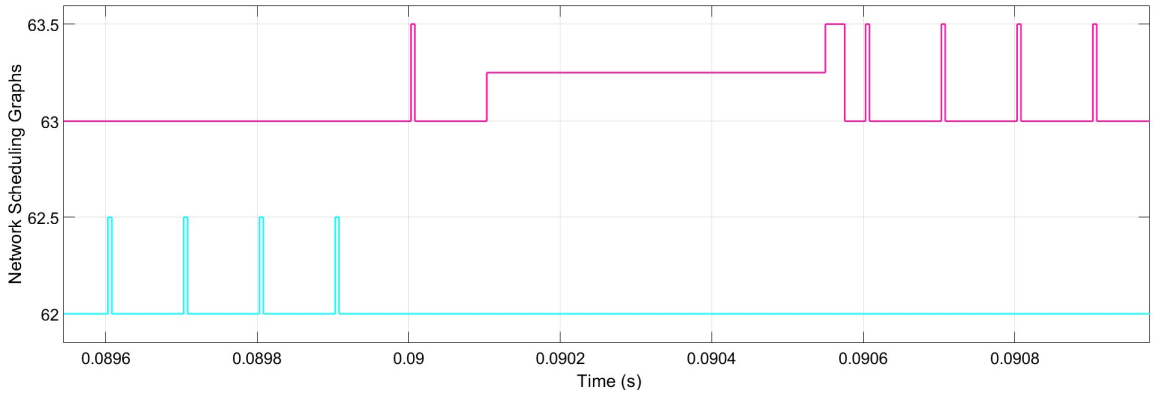

Figure 25: Topology G 1_60 Large Network Scheduling

When we look closer to the network scheduling of the topology, we can see that even though the messages are prepared they cannot be sent because the network is too crowded. We can see this behavior as in Figure 25 that the signal line does not go down but does not go up either, it stays in middle. This means the message is prepared but could not be sent, and here this happened because of the gateway computer transmitting on the network. So as one of the messages is being transmitted, the other messages from main computer to the rest of the motor drivers wait until it is over so the next one can start. This goes until the very last message for the last motor driver from the main computer is transmitted and finished. Only then the sensor node can transmit its messages to the motor driver again, and the gap in between can be critical, depending on the size of it.

The figure shows that when the network is larger with more motor driver nodes, the periodic messages from main computer to each motor driver takes a lot more time before letting sensor node to transmit again. This size of a network is not even close to the real system and causing these problems, so on a larger network these gaps get larger as the network gets more crowded. This is a problematic situation and one of the reasons why we had to design a second topology.

5.1.3 – Multi Elevator Car Test

Even though we already had some issues on the last section, we still wanted to observe how well can this topology handle the multi car test, so we created a medium sized topology not to create the problem mentioned in the last part but to be able to see if running simultaneous cars has a negative effect on the simulation results. With 40 motor drivers, the problem should not be too apparent so that we can analyze the multi elevator car test alone.

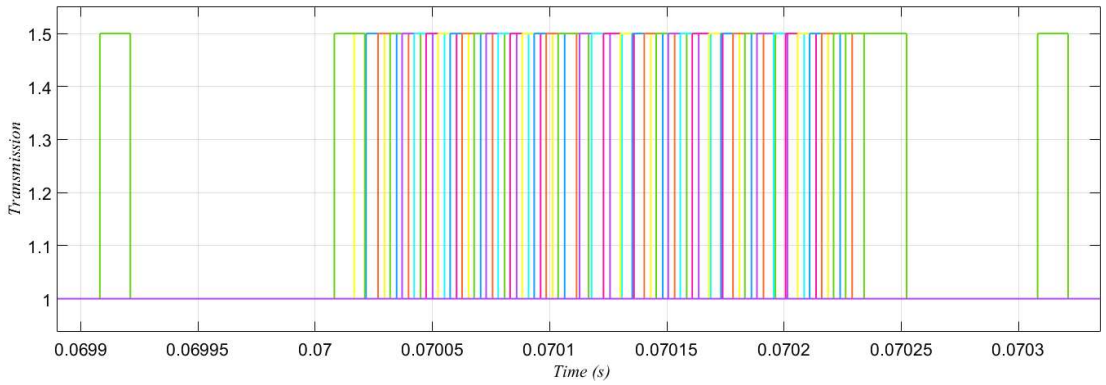

Figure 26: Topology G 1_40 Single-car Network Transmission

In Figure 26, we have the topology with 40 motor drivers on a single network and they are connected to one gateway computer. So again, when we are reading these figures we need to check if signals are either colliding with each other resulting in some of them to be transmitted later or some signals missing their deadlines completely so the packets are dropped.

If we look at the figure, we will be able to guess around 0.0701 second mark is the periodic green signal that we can see. That signal belongs to the sensor node sending position data to the motor drivers on the network. The colorful group of signals that is covering the sensor node signal is the messages from main computer to the motor drivers relayed by the gateway computer. Main computer sends a message to every motor driver node one by one and here we can see since there are 40 of them when they are added end to end, they can cover close to a period of 0.00025 s (250 μs). We can see that even though this period is long enough to block a message from the sensor node it is not critically harmful.

As we have this network as a control variable, we can now simulate multi-car elevator system and see if the we have similar results. We will be using 8 elevator cars to make the difference more visible.

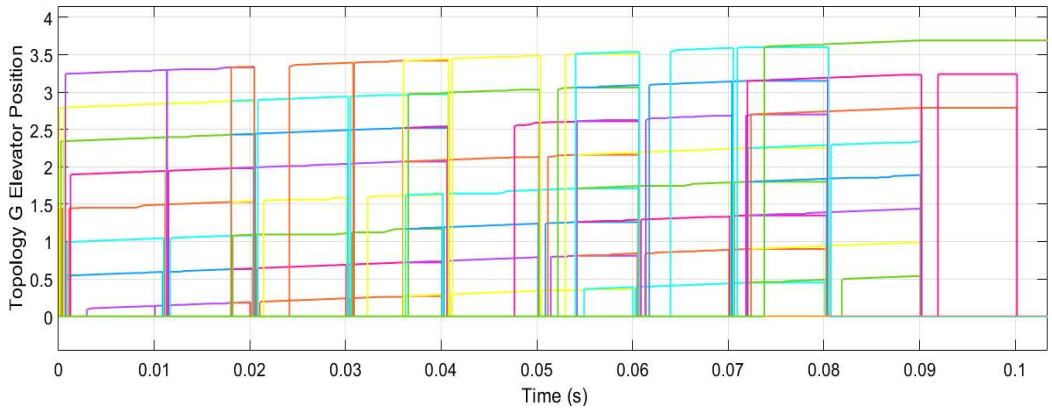

Figure 27: Topology G 1_40 Multi-car Network Sensor Data

On Figure 27, we can see a similar graph to what we can see on Figure 21, but there are some differences; first of all this is a sensor position data graph which means we should be seeing the position data of elevator cars, and since there are 8 movers, we can see 8 parallel lines slowly increasing their values, second and more importantly, in a successful simulation run we expect to see these lines to be uninterrupted, meaning they should be continuous in such a way that every time when one motor driver is finished receiving the next one should start so the switching should be seamless. Unlike the graph we see on Figure 21, this is not the case here; at multiple points in time, the graph is interrupted and

out for some period (see orange line at 0.02-0.023 s). This means while the elevator is being driven by a motor driver, the messages are cut out and the motor can no longer drive the mover as there is no new position signal coming. The reason for this is that the bandwidth of Ethernet protocol that we have implemented can withstand up to a certain amount of message load, and if in a situation that this certain amount is reached or passed, with messages overlapping, some of the messages will not be transmitted, they will wait for another node to stop transmitting but since they run together, that is not an option.

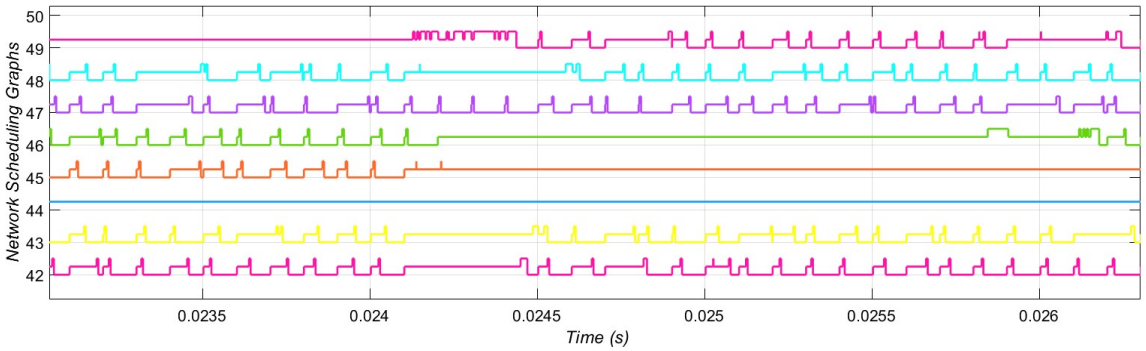

Figure 28: Topology G 1_40 Multi-car Network Scheduling

We can also check Figure 28 to see how much of the messages actually coincide, and as we stated in large network test, again we are facing a similar situation, where the nodes are ready to transmit as their messages are prepared but since the network is overloaded, some of the nodes cannot find space to transmit. As we observe with eight elevator cars running simultaneously, so many of the messages are either missed their deadlines being late as we can see on the pink signal on channel 49 on the network transmission sequence or dropped altogether as it happened with the green channel on 46.

Additional to the second test, this also proves that this topology is not sufficient to meet the requirements of the system. In many cases, Ethernet with its large bandwidth is considered to be capable to serve all communication requirements including real-time ones as here. However, it is apparent from this example that even high-speed communication networks designed for high average transmission rates is not capable of sustaining real-time communications on a modestly sized network. Next, we can observe how Topology L performs in the same conditions.

5.2 – Topology L

As we went over it on section 3, in Topology L we have an outer network consisting of a main computer and gateway computers, and every gateway computer is also connected to a driver network where motor drivers are grouped under. The biggest difference that we expect to become an advantage on Topology L compared to Topology G is that, as motor drivers are divided into small groups of networks, their cumulative load on one network is also divided into smaller pieces.

Just like Topology G on last part we have three tests again. We will be testing the small network, large network and multi elevator car test to see if this topology is capable of the requirements of the real system.

5.2.1 – Small Network Test

Once again, we start by creating a small network, to keep everything fair we tried to keep the motor driver node amount same on the same type of tests. So, we start by a network where there are ten motor drivers, two (instead of one) gateway computers, no sensor nodes as they are in gateway computers and one main computer.

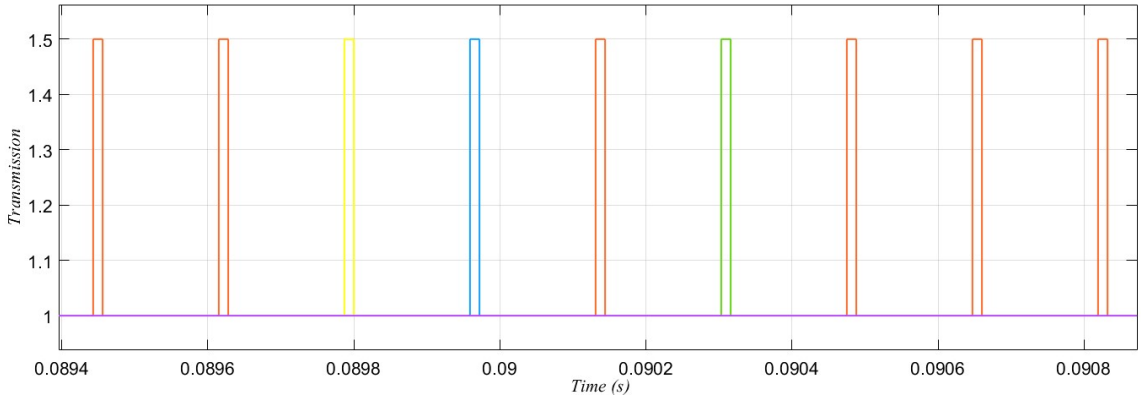

Figure 29: Topology L 2_10 Small Network Transmission

Seen on Figure 29 when the network is small it works as expected, gateway computers sending sensor read data to motor drivers and around every 8 µs gateway interrupts the sensor data transmission to relay the message from main computer to the motor drivers. Sensor data being sent by the gateway computers is shown as orange here and main computer messages relayed by gateways are shown in multiple colors one after the other.

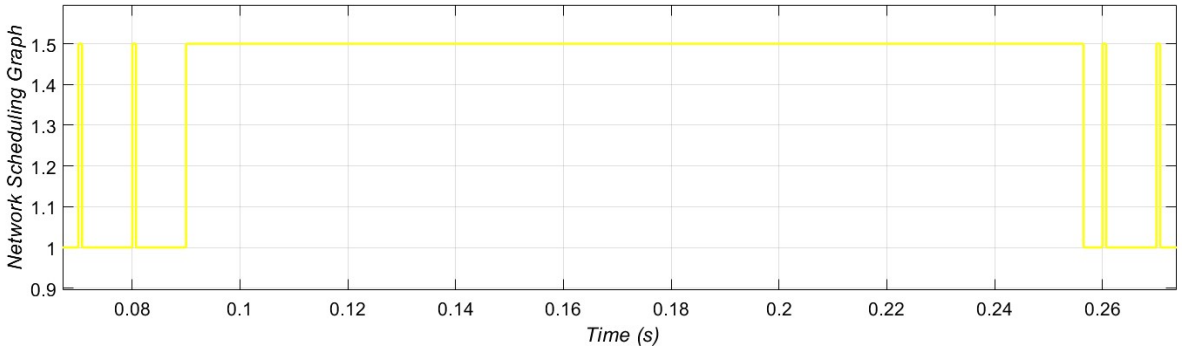

Figure 30: Topology L 2_10 Small Network Scheduling

On Figure 30, if we were to look at the network transmission sequence, we realize that the short and often transmission on the network is the messages coming from the main computer relayed by the gateway to the motor drivers. After around 0.09 s mark it is constantly occupied until 0.26 s mark meaning that the elevator car is passing in front of the sensors regarding this motor section. So, the motor drivers in this network are being used. And we could have seen if there are any interruptions or changes in the length pointing in a problematic situation but as we can see on Figure 30, none of these problems occurred.

Now we can check large network to see if the problem we had in Topology G occurs in Topology L as well.

5.2.2 – Large Network Test

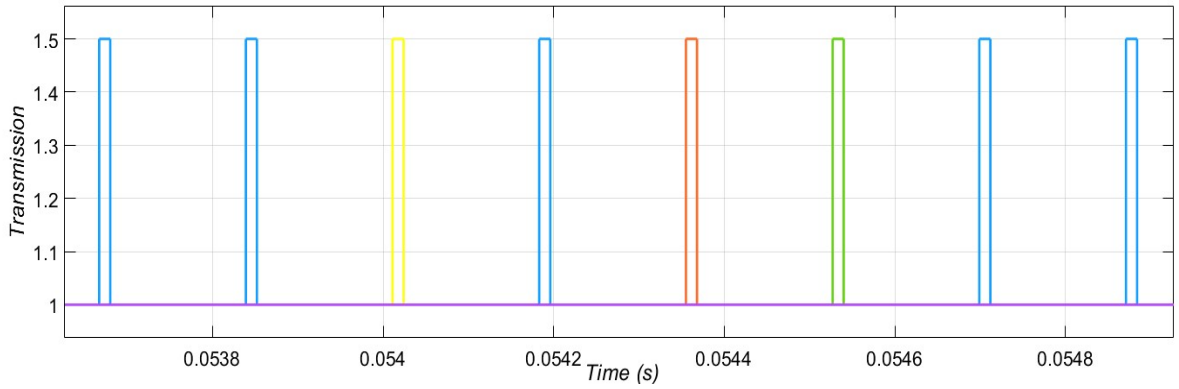

Figure 31: Topology L 12_60 Large Network Transmission

As we expected, when we divide the topology into smaller networks, the load is divided as well. We can see on Figure 31 that even a six-fold increase in number of motor drivers, did not put any affect into the plot when compared with Figure 29. This is because independent of how many gateway computer nodes there are, each gateway is connected to a separate network and only 5 motor driver nodes. This way the messages coming to a motor driver in another motor section does not affect the rest of the system at all.

We can see that the number of motor driver nodes does not affect the overall system load on this topology by checking the Scheduling plot on Figure 32 and the width of the total transmission of the driver network stayed exactly the same and the continuous transmission line is not interrupted either. This means no messages are being late, missing deadlines or colliding within driver networks.

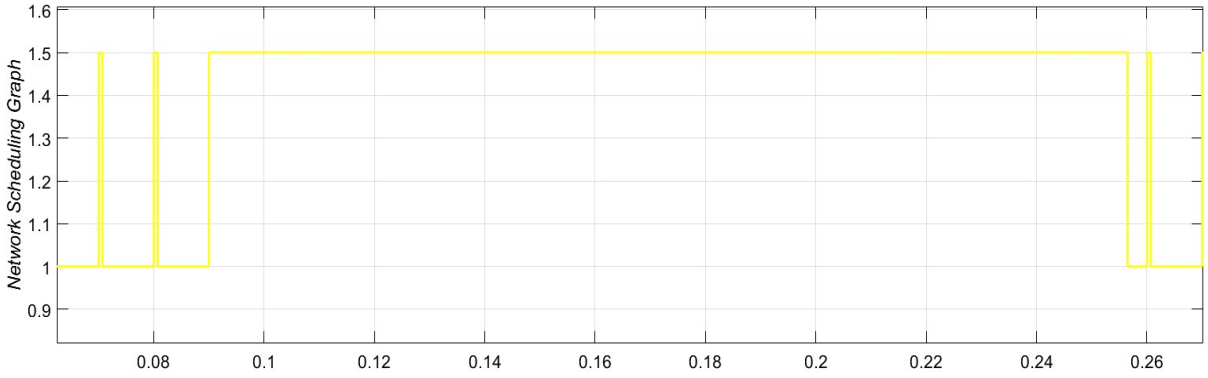

Figure 32: Topology L 12_60 Large Network Scheduling

5.2.3 – Multi Elevator Car Test

Finally, we can test if this topology is appropriate for the real system, we need to be able to see if any messages are delayed, late or completely missed while running simultaneously.

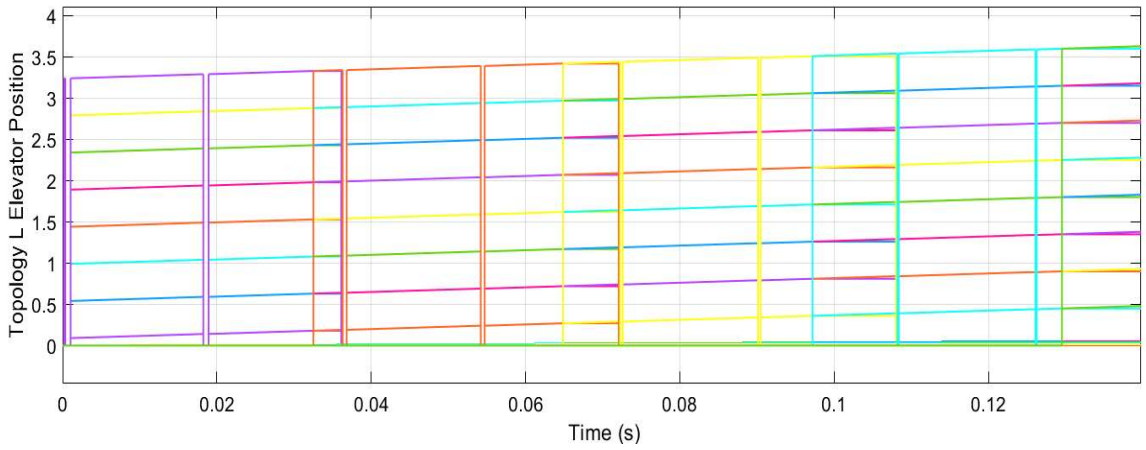

Figure 33: Topology L 8_40 Multi-car Network Sensor Data

In figure 33 we can see all eight driver networks' sensor data in a single plot. Even though we are running eight elevator cars all simultaneously, since every elevator car occupy different driver networks there are no cases that a driver network can suffer from extended load.

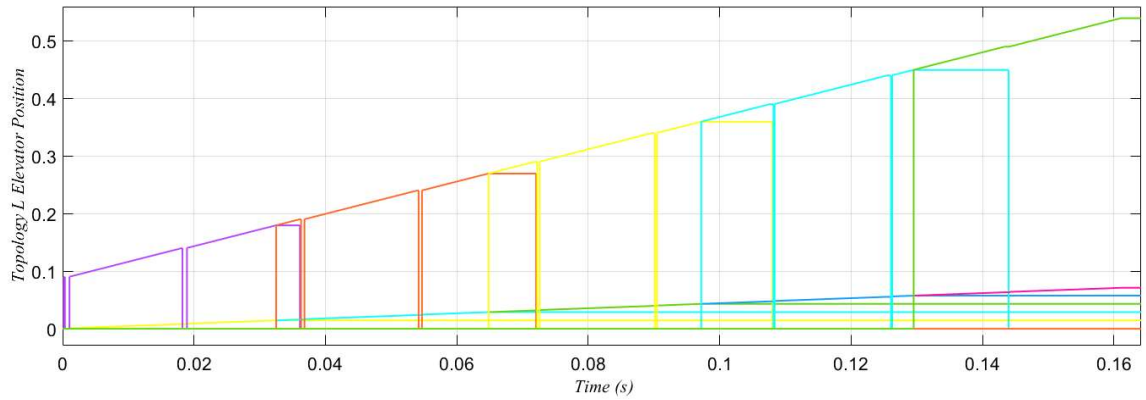

Figure 34: Topology L 8_40 Multi-car Network Sensor Data

On Figure 34 we can see one of the eight driver networks' sensor data. As we grouped the motor drivers to separate networks, it is expected that the sensor data transmission within each network is not affected by the total number of motor driver nodes. In fact, the only parameter that can affect the driver networks is the amount of motor drivers per gateway computer hence the amount of motor driver per driver network. Throughout our simulation results here, we always kept the number of nodes per network constant as five since it was a design parameter. The real system is planned to have five motor drivers per gateway computer as well. That is how we can make sure if our simulation model and our results are realistic or not.

5.3 – Topology R

Topology R has a few similarities with Topology L when we consider the motor drivers being grouped by gateway computers which relay the messages coming from the main computer. This side of the topology, alike Topology L, has a separate network called "Outer Network" handling messages from main computer to the gateway computer using Ethernet protocol. The rest is different as the inner network on Topology R uses a ring topology with EtherCAT protocol.

Since all the messages are carried in a datagram for every 5 motor drivers, increasing the number of nodes does not really affect the overall performance of the system, just like we have seen the same effect on Topology L with small and large network comparison. It only increases the workload of the main computer ever so slightly as it needs to deliver the message to more gateway computers to be relayed to motor drivers of those gateways. As the network scheduling graph of a larger network seems exactly the same as Figure 20, it is not included.

One other thing we can observe if the system is capable of performing correctly is the position data from sensor nodes transferred to the respective NIC which then puts this data in a message on datagram, and the correct NIC receives it to pass the message to the respective motor driver and the motor driver displays the position data. We can see the sensor data on Figure 35.

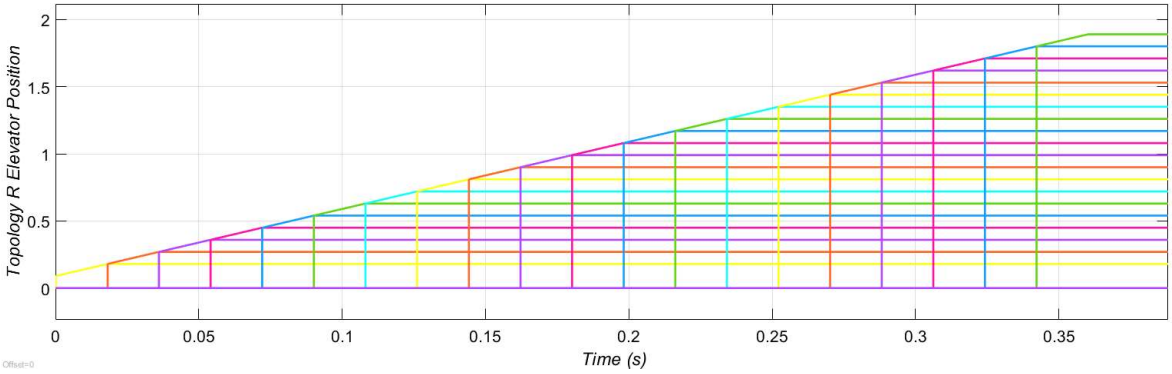

Figure 35: Topology R 4_20 Network Sensor Data

5.2.4 – COMPARISON OF THREE TOPOLOGIES

Given the results we discussed on 5.1, 5.2 and 5.3, Topology G is a simpler network topology compared to Topology L when we consider the ease of implementation and configuration change such as adding or removing a node. But this comes with the cost of being less efficient and less effective. As the network gets larger and the number of node increases, the performance of the topology decreases as much. When we switched to a large network on Topology G, we proved that, it is not going to be sufficient to use this topology for the real system and after testing the multicar test as well, it was for sure that Topology G cannot handle the communication traffic therefore fails to be solution for our problem in the real system.

On the other hand, we have seen the performance of Topology L, which was stable through every test. As there were multiple networks, the load on the network was divided at every situation and this let the topology to never become ineffective. This behavior of Topology L lets us add as many drivers to the network (as long as other components do not fail) as we want, and still the load on any specific driver cannot be increased more than a certain value.

Similarly, on Topology R as we are separating the motor driver and sensor nodes into groups under gateway computers, we divide the load into the number of gateway computers which guarantees that as much as we increase the overall number of motor drivers in the elevator system, every group will have a certain lower amount of (5 in our applications but can be arranged to any other amount depending on the system) motor drivers so the performance of the system is not going to change at all.

Unlike Topology L though, on Topology R it is significantly more challenging to solve any issues regarding with the EtherCAT network or its nodes compared to CANBUS and Ethernet, even though it is easier to pinpoint the problem on EtherCAT.

6. CONCLUSION

In this thesis, three different network topologies are introduced and simulated to control a long armature linear motor with multiple elevator cars. To be able to simulate various network topologies on MATLAB, a generic simulation environment was created utilizing the real-time control system toolbox, TrueTime. The advantages and disadvantages of all three network topologies are explained clearly and their results are compared in terms of stability and effectivity.

In Topology G, the simplicity of the topology allows for easy installation and maintainability while giving up on performance directly proportional to the amount of motor drivers on the network. Having all motor drivers connected to a single gateway computer creates bottleneck in respect of communication delay up to the point that the whole system fails especially with multiple elevator cars running simultaneously.

In Topology L, the gateway computers undertake the mission of the sensors along with relaying the messages from main computer to the motor drivers. And grouping a small number of motor drivers under each gateway computer decreased the communication delay significantly (section 5.2.2). This structure that Topology L and Topology R demonstrate, enables all motor drivers to have nearly the same amount of communication delay allowing the motors to work smoothly and in harmony, not experiencing any fluctuations. Of course, on Topology L there are still small spikes in delays particularly when the main computer is transmitting necessary information to the gateway computers to relay the messages to the motor drivers to be able to manage the elevator cars, but this is not the case with Topology R as it simply adds the message coming from main computer to the datagram and passes it to the following NICs, nothing changes from the perspective of any of the NICs or the nodes that are connected to them. These messages are crucial to drive the elevator car, but the frequency of these messages can be arranged so that they are sufficiently frequent but as low as possible. But regardless of the spikes, the delay levels are always within the margin of error and acceptable on both Topology L and R.

Therefore, we can recommend one of these two topologies for the communication of long armature linear motor with multiple elevator cars but not Topology G because of its shortcomings.

Another important appendage of this thesis is the usage of MATLAB scripts to create Simulink models instead of using the graphical user interface (GUI) by using drag and drop method. As using the GUI method can be very frustrating with models that consist of tens and hundreds of kernel and network blocks that you need to put and connect one by one, the script method allows the usage of for loops that can handle re-iterative jobs while building the model. Another very important additional note on using the script method is to be able to make any changes on all of the related nodes at once by changing it on the script. By using uncomplicated MATLAB scripts, it is possible to enlarge, shrink or edit a network topology effortlessly by changing a few parameters and settings. Although it is time consuming at first, as the development of a project goes on, using this method instead of graphical user interface method clearly pays of in the long run.

BIBLIOGRAPHY

[1] W.D. Jones. How to build a mile-high skyscraper. Spectrum, IEEE, 44(6):52-53, June 2007.

[2] T. Ishii. Elevators for skyscrapers. Spectrum, IEEE, 31(9):42-46, Sep 1994

[3] Kita H. Markon, S. and H. Kise. Control of Traffic Systems in Buildings:Applications of Modern Supervisory and Optimal Control (Advances in Industrial Control). Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2006

[4] Kita H. Suzuki H. Sudo T. Takahashi, S. ans S. Markon. Simulation-based optimization of a controller for multi-car elevators using a genetic algorithm for noisy fitness function. The 2003 Congress on Evolutionary Computation, CEC '03, 3:1582-1587 Vol.3, Dec. 2003

[5] Koseki T. Miyatake, M. and S. Sone. Design and traffic control of multiple cars for an elevator system driven by linear synchronous motors.In Proc. Symposium on Linear Drives for Industry Applications, number AP-22, pages 94-97, 1998.

[6] Koseki T. Miyatake, M. and S. Sone. Scheduling for high density transport in ropeless lift systems using multiple cars with transferability between shafts. Computers in Railways, pages 375-384, Aug. 1996

[7] Koseki T. Miyatake, M. and S. Sone. A proposal of a ropeless lift system and evaluation of its feasibility,. IEEJ Trans. IA, 119(11):1353-1360, 1999

[8] Osawa Watanable T. Yamaguchi, H. and H. Yamada. Brake control characteristics of a linear synchronous motor for roplesess elevator. IEEE Trans. On Magnetics, 37(5):3732-3736, 2001.

[9] Inc, S., (2002) . Networking Complete. Third Edition. San Francisco: Sybex

[10] Groth, David; Toby Skandier (2005). Network+ Study Guide, Fourth Edition. Sybex, Inc. ISBN 0-7821-4406-3.

[11] Sosinsky, Barrie A. (2009). "Network Basics". Networking Bible. Indianapolis: Wiley Publishing. p. 16. ISBN 978-0-470-43131-3. OCLC 359673774. Retrieved 2016-03-26.

[12] Licesio J. Rodríguez-Aragón: Tema 4: Internet y Teleinformática. retrieved 24 April 2013. (in Spanish)

[13] Protocol, Encyclopædia Britannica, retrieved 24 September 2012

[14] Comer 2000, Sect. 1.3 - Internet Services, p. 3, "Protocols are to communication what algorithms are to computation"

[15] Broadcast Communication Networks. In National Programme on Technology Enhanced Learning (NPTEL) [online]. June 2013. Available at http://nptel.iitk.ac.in/ courses/ Webcourse-contents IIT/ %20Kharagpur/Computer%20networks/pdf/

[16] Javvin Company. In Protocol Dictionary [online]. Available at http://www.javvin.com/ protocolToken.html

[17] D. Henriksson. TrueTime Simulation of Networked Computer Control Systems. In preprints of 2nd IFAC Conference on Analysis and Design of Hybrid Systems. Alghero, Italy, 7-9 June 2006.

[18] M. Andersson. Simulation of Wireless Networked Control Systems. In 44th IEEE Conference on Decision and Control, and the European Control Conference 2005. Seville, Spain, December 12-15, 2005,476-481.

[19] A. Cervin, M. Ohlin, D. Henriksson. Simulation of Networked Control Systems Using TrueTime. In 3rd International Workshop on Networked Control Systems. Nancy, France, 2007

[20] A. Cervin, D. Henriksson, M. Ohlin. TrueTime 2.0 Beta Reference Manual, Department of Automatic Control, Lund University, June, 2010

[21] A. Cervin, D. Henriksson, B. Lincoln, J. Eker, K. Arzen. How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime. In Control Systems, IEEE 23(3):16-30.

[22] D. Henriksson, O. Redell, J. El-Khoury, A. Cervin, M. Törngren, K. Arzen. Tools For Real Time Control Systems Co-Design. In H. Hansson (ed.), ARTES- A Network For Real Time Research and Graduate Education in Sweden 1997-2006, Department of Information Technology, Uppsala University. Sweden, 2006.

[23] Karayağız, K. (2013). Network topologies for long armature linear motors (master thesis).

[24] Fidge, C. (2002). Real-Time Scheduling Theory.

[25] Arvind, K., Ramamritham, K., & Stankovic, J. A. (1991). A local area network architecture for communication in distributed real-time systems. *Real-Time Systems*, *3*(2), 115-147.

[26] Benítez-Pérez, H., Ortega-Arjona, J. L., Méndez-Monroy, P. E., Rubio-Acosta, E., & Esquivel-Flores, O. A. (2018). *Control Strategies and Co-Design of Networked Control Systems: Considering Time Delay Effects* (Vol. 13). Springer.

APPENDIX A : TRUETIME KERNEL BLOCK

Init function name is used to set the function that is going to be initializing the kernel. More than one kernels can be initiated with the same function, in fact we have one initialization function to initialize every kernel on the network. To be able to do that, in the initialization code we need to be able to separate different kernels from each other. This is where init function argument comes in, once every unique kernel has a different init function argument given according to a sequence to make things easier, it is as simple as an "if-else statement" to check which kernel should go to which function. Analog input and outputs can be thought as wired electrical connections to or from the kernel, so it is the number of input and output ports our kernel going to have. Network and node number is the slightly more complicated one; it allows us to set which networks are we going to connect the kernel to and in each one of those networks we also need to specify which node number is the kernel going to have. There is a special syntax for this, which goes like the following:

If we have only one network, let it be network number 1 and on that network the kernel is node number 4, then we can write: "[1 4]" or "[4]" as the default network is "1" and if nothing is written for network it uses the default.

If we have multiple networks, let them be network number 1 and 3 and on network 1, kernel is the node number 4 and on network number 3, kernel is node number 6, then we can write: "[1 4;3 6]". As we can see, we need to separate networks from each other with a ";" (semicolon) and in the beginning and end we have "[" and "]" square brackets. A kernel can connect to as many networks as we want, although it is important for us to follow the syntax.

In the kernel function, we can create a task that can simulate periodic and aperiodic actions. To be able to create a task we need to define some parameters, such as release time, worst case execution time, relative and absolute deadlines, priority, period [19], [21]. An example init function with periodic task creation and parameter definition is shown below:

```
function example_init_func(arg)
% example node
% Initialize TrueTime kernel
ttInitKernel('prioDM') % priority-based deadline-monotonic scheduling
% Periodic dummy task with higher priority
starttime = 0.0;
period = 0.005;
data = period*arg;
ttCreatePeriodicTask('example_task', starttime, period, 'example_code', data);
```

The “ttInitKernel()” function initializes a node by specifying the scheduling algorithm given as a parameter. The built-in default scheduling algorithm is fixed-priority scheduling “prioFP” but we can specify other methods such as rate monotonic scheduling “prioRM”, earliest deadline first (EDF) “prioEDF” and deadline monotonic “prioDM” [17].

TrueTime blocks are event-driven and they can handle external interrupts which are correlated to external interrupt channels of the computer node. When the corresponding interrupt channel’s signal changes, it triggers an interrupt. This can be used for simulating disturbed controllers where a signal or measurement arrive to the node on the network. An interrupt can be handled by a user-defined interrupt handler when it occurs. An interrupt usually has a higher priority level. The syntax of an interrupt handler is first; name, then; priority, and then; function code name [21]. An example interrupt handler is shown below:

```
%Creating an example network interrupt handler
data = 'example_task';
ttCreatehandler('example_network_handler',1,'example_network_handler_code',data);
ttAttachNetworkHandler('example_network_handler')
```

A task execution can be preemptive or non-preemptive and there are three seperate priority levels. Highest priority level is the interrupts, then comes kernel and lowest priority level is the task. At kernel level and task level, dynamic priority scheduling can be used but at interrupt level only fixed priority scheduling can be used. The priority of a task is defined by user in a priority function in each scheduling point. This simplifies simulating different scheduling algorithms. For most scheduling algorithms that are commonly used, there is a pre-defined priority function [19], [21].

When the simulation runs, kernel executes the function code for the task and interrupt handlers. The function code can be divided into segments, and after the simulation is complete, each segment’s execution time is returned as output of the function code.[30]

During simulation, every time the function code is called it saves the current segment that has been executed, and if the task has been running for the specified execution time, it continues from the next segment [22]. An example function code of a sensor is shown below:

```
function [execution_time, data] = example_sensor_function_code(segment, data)

persistent message

switch segment
    case 1
        %Measurement value taken from the analog input
        message = ttAnalogIn(1);
        execution_time = 0.0005;
    case 2
        %Send a message containing 120 bits to node number 4
        ttSendMsg(4, message, 120);
        execution_time = 0.0003;
    case 3
        %Completed
        execution_time = -1;
end
```

This function code is of a simple sensor's which the function is divided into three segments. The first segment is where the data is acquired from the analog input by the sensor which took 0.5 ms to execute, second segment is where the message acquired in the first segment is sent to node number 4 and took 0.3 ms to execute, and finally the last segment did nothing and sent a "-1" execution time that shows the end of execution. The structure "data" acts as the local memory which allows us to keep variable values such as the sensor read value between segment executions [20].

APPENDIX B : TRUETIME NETWORK BLOCK

A node can send and receive messages by using “ttSendMsg()” and ”ttGetMsg()” built-in functions on a network that they are in. For more detailed look over these built-in functions, please check the TrueTime Manual.

A TrueTime network supports the following network types: CSMA/CD (Ethernet), CSMA/AMP (CANBUS), Round Robin (Token Bus), FDMA, TDMA (TTP), Switched Ethernet, WLAN (802.11b), and ZigBee (802.15.4). We will be using CSMA/CD (Ethernet) and CSMA/AMP (CANBUS) network protocols in this project.

CSMA/CD (Ethernet)

CSMA/CD stands for Carrier Sense Multiple Access with Collision Detection. If network is not idle, a node that wants to transmit must wait until the network is free, and if a message is transmitted within 1 microsecond of another a collision occurs. In the event of a collision the second node must back off for a period that can be calculated by:

$$t_{backoff} = minimumframedatarate \times R$$

where

$$R = rand(0.2^K - 1) \quad \text{(discrete uniform distribution)}$$

and K is the number of collisions in a row. K can be maximum 10 and minimum frame size cannot be 0. After waiting for $t_{backoff}$ amount of time, the node will try to transmit again.

CSMA/AMP (CANBUS)

CSMA/AMP stands for Carrier Sense Multiple Access with Arbitration on Message Priority. As expected from a real-life application of CANBUS, if the network is busy, then sending node will wait until it becomes free in the simulation. If a collision occurs, the message with the highest priority will continue to be transmitted. Explicit to simulation, if the collision is occurring between two equal priority messages an arbitrary choice is made to transmit which message to be transmitted first. Of course, in real life application, CANBUS nodes all have unique identifiers, which is used to compare priorities, so it is impossible to have two equal priorities [20].

APPENDIX C

These simple steps are followed while creating a new network topology model:

- The parameters are introduced (Configuration)
- Check if a model with the same name already exists, if it does then erase
- Create a new empty model using new_system function of MATLAB
- Open TrueTime library using open_system function of MATLAB
- Now we can use other MATLAB functions such as add_block, add_line and set_param as well as some for loops and if statements. At the end when our model is created, we can use save_system command to save the model we just created.

To be able to change any parameter easily we created a separate config file that holds all the parameters and the function code that creates the topology calls the config in the beginning to have access to all the parameters. This way all important parameters that the user may want to change to test a different configuration are in front of the user similar to a user interface.

Now that we have a plan to follow, we can go in further detail creating a model below:

Configuration

First, we begin by the configuration file, which is the part we define all the parameters and set them according to the model we would like to create.

```
% Specify the name of the model to create
Topology Name = 'topology g';
```

After naming the model to create, we are ready to define parameters. The parameters that we define here are going to be used by other scripts while and after we create the model, so if we want to change a value and test again, we need to create the model from the beginning after the changes.

```
% Introduce the parameters
Num_of_Gate = 1;
Num_of_Drivers = 30;

% the amount of motor drivers in between every sensor node
Sense_node_gap = 5;
Num_of_Sensor_Nodes = Num_of_Drivers / Sense_node_gap;

%multicar section
Num_of_cars = 2;
%please choose numbers that can divide number of drivers
without remainder and that do not exceed num of sensor nodes
(max equal to it)

Num_of_drivers_per_car = Num_of_Drivers / Num_of_cars;
Num_of_sensor_per_car = Num_of_Sensor_Nodes / Num_of_cars;
%multicar end
%max motor speed we want to test
(5 - 10 - 20)(m/s)
motor_speed = 5;
%initialization function for all Truetime blocks
init_function='initialization';

Outer_Network_No = 50000;

driver_length = 0.09;
%% the length of one motor driver
sensor_offset = 0.09;
%% sensor position is 0.09 far from the bottom of the mover.

first_driver_ID = 1;
last_driver_ID = Num_of_Drivers;

Max_Driver_No=1000; %max number can be 1000
Max_Gateway_No=1000; %max number can be 1000

Const2=40000; %parameter used for naming the 2nd scope of the Gateway
```

After defining every parameter, we need, we are ready to create the model, so this sums up the configuration file and we can move to the create topology code.

Create Topology (Topology G)

The very first thing we need to do before creating any models is to clear the workspace so that some parameters that are left from previous runs are erased and we can start fresh. This is especially important as some parameters' values are changed while the simulation is running and in the case, we do not clean up, these changed parameters can cause some errors. To clear the workspace, we use:

```
%clean workspace
clear all
%clc
```

Now that we cleared the workspace, we can call our parameter definitions by calling the config.m file. In MATLAB to call another user script function we just need to write the name of the function without the ".m" extension.

```
% call config.m for all the variables and settings required
config;
```

Then, we can check if any other file with the same topology name as we want to create exists. If it exists, we need to remove that before creating a new one.

```
% Specify the name of the model to create on config file
fname = Topology_Name;

% Check if the file already exists and delete it if it does
if exist(fname,'file') == 4
    % If it does then check whether it's open
    if bdIsLoaded(fname)
        % If it is then close it (without saving!)
        close_system(fname,0)
    end
    % delete the file
    delete([fname,'.slx']);
end
```

After that, we can create the new system by using the MATLAB function new_system:

```
% Create the system
new_system(fname);
```

Just like we open the Simulink Library to use the common blocks in Simulink, we need to open the TrueTime Library to be able to use (copy) its blocks. Since we created our system,

we can open both TrueTime Library and our newly created system with the same command:

```
% Open truetime library and the created system to copy(add) blocks
open_system('truetime');
open_system(Topology_Name);
```

Once these are all done, we are ready to add blocks, set those blocks parameters and connect them to one another. This is the part we create our model, we will be using the built-in functions that are “add_block()”, set_param() and “add_line()” respectively. These functions take multiple parameters each and they are shown below:

```
%add_block: creates a copy of the "src"(source) block to the "dest"
destination, with specified parameters
add_block('src', 'dest', 'param1', 'value1', 'param2', 'value2',...)


%set_param: allows to set parameters of the specified "obj" block
set_param(obj, 'param1', 'value1', 'param2', 'value2',...)

%add_line: adds a connecting line from a block's output port "oport" to
another block's input "iport"
add_line('sys', 'oport', 'iport')
```

As we will have detailed examples from our system, we are not going into too much detail here. For more detailed info on any command on MATLAB including the three above, there is built-in reference on MATLAB, accessible by typing “help” followed by the command the reference is needed for.

Adding Simulink Blocks

In our model, as much as we use TrueTime blocks, we also need some Simulink blocks, such as clock, gain, display, constant, sum, display and sine wave. We can go over a few examples of how they are used below:

```
% Add Clock and Display
add_block('built-in/Clock', [gcs,'/Clock'],'Position', [800 20 830
50]);
add_block('built-in/Display', [gcs,'/Display'],'Position', [850 20 900
50]);
% Connect Clock and Display
add_line(Topology_Name,'Clock/1','Display/1')


% Add Gain
add_block('built-in/Gain', [gcs,'/Gain'],...
    'Position', [850 150 880 180],...
    'Gain',num2str(motor_speed),'SampleTime','-1');
% Connect Clock and Gain
add_line(Topology_Name,'Clock/1','Gain/1');

% Add Sum
add_block('built-in/Sum',[gcs,'/','Sum'])

% Set Sum Block Parameters
set_param([gcs,'/','Sum'],...
    'inputs','++',...
    'position',[900,150,930,180])

% Connect Gain and Sum
add_line(Topology_Name,'Gain/1','Sum/1');


%GOKALP additional cars sums and connections
for d = 1:1:(Num_of_cars-1)
    add_block('built-in/Sum',[gcs,'/',['Sum' num2str(d)]]);
    set_param([gcs,'/',['Sum'
num2str(d)]],'inputs','++','position',[900+100*d,150+100*d,930+100*d,18
0+100*d]);

    add_line(Topology_Name,'Sum/1',['Sum' num2str(d) '/1']);
    add_block('built-in/Constant', [gcs,['/Constant'
num2str(d)]],'Position',
[850+100*d,200+100*d,880+100*d,230+100*d],'Value',num2str(Num_of_driver
s_per_car*d*driver_length),'SampleTime','inf');
    add_line(Topology_Name,['Constant' num2str(d) '/1'],['Sum'
num2str(d) '/2']);
end
```

```
% Add Constant
add_block('built-in/Constant', [gcs,'/Constant'],...
    'Position', [850 250 880 280],...
    'Value',num2str(driver_length),'SampleTime','inf');

% Connect First Output of Constant Block to Second Input of Sum Block
add_line(Topology_Name,'Constant/1','Sum/2');
```

After executing it all, what we have on Simulink looks like this:

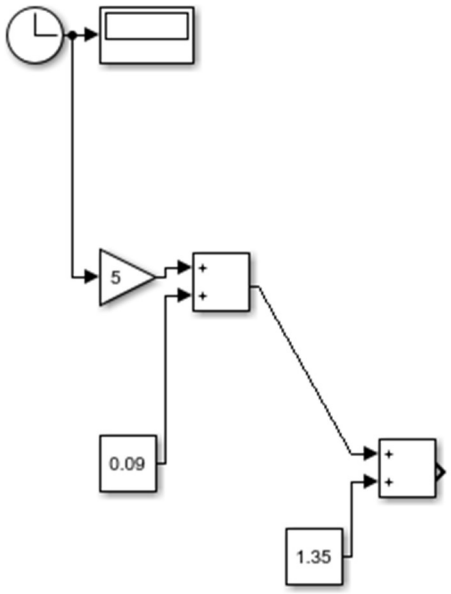

Figure 36: Create Topology, Clock, Gain, Sum

Scope blocks by default show only the last 5000 data points, however we can force to show all data points by setting the parameter "LimitDataPoint" to "off" to have a better and more fair evaluation on the result.

Adding a clock, a display or a sum block is simple as it is shown above, but what if we need to add many multiples of a block, for example the gateway computers, or motor drivers. We need to add so many of them, that adding them line by line is as frustrating as drag and drop method on Simulink. Another problem is that even if we add line by line, we need to name each block different, as MATLAB does not allow the user to use the same name twice. To solve these problems, we can use some algorithms with the help of "for

loops" and "if-else conditional statements" to create multiples of the same block with a naming sequence that we prefer. We can create a for loop with the number of gateway computers, and inside the loop we can create another for loop with the number of motor drivers per gateway so that we can have nested loops and creating the network topology will be so much easier and practical. When we need to change a parameter for all the motor drivers for example, we will be changing one line and it will affect all of them. But before we add in the motor drivers and gateway computers, we need to add a network block since, all these nodes need to be specified to which network they are going to be connected to. The following example shows how to add and set the parameters of a network block:

```
% Generate OUTER NETWORK BLOCK (Set Block Parameters, Add Scope, Connect
Blocks)
add_block('truetime/TrueTime Network',[Topology_Name '/Outer Network'],...
    'Position', [10 (20) 70 (70)]);

set_param([Topology_Name '/Outer Network'],...

'nnodes',num2str(Num_of_Gate+1),'nwnbr',num2str(Outer_Network_No),'nwtype','C
SMA/CD (Ethernet)','rate','100000000','minsize','1144');

add_block('built-in/Scope', [gcs,['/Scope' num2str(Outer_Network_No)]],...
    'Position', [100 (10) 130 (40)],'LimitDataPoints','off');
add_line(Topology_Name,'Outer Network/1',...
    ['Scope' num2str(Outer_Network_No) '/1']);
```

The code above creates our "outer network" which gateway computers and main computer are connected to. As we have a network to specify while creating the gateway computers now, we can see how they are created.

```
% Generate MAIN COMPUTER BLOCK (Set Block Parameters, Add Sine Wave Block,
Add Scope, Connect Blocks)
add_block('built-in/Sin', [gcs,'/Sine Wave'],...
    'Position', [10 120 30 140],'SampleTime','0' );
add_block('truetime/TrueTime Kernel', [Topology_Name '/Main Computer'],...
    'Position', [60 110 120 160]);
set_param([Topology_Name '/Main Computer'],...
    'sfun',init_function,'args',['[' num2str(Outer_Network_No)
']'],'ninputsoutputs','[1 1]',...
    'nwnodenbr',['[', num2str(Outer_Network_No),' ',
num2str(Num_of_Gate+1),']']);
add_line(Topology_Name,'Sine Wave/1','Main Computer/1' );
add_block('built-in/Scope', [gcs,['/Scope' num2str(Outer_Network_No+1)]],...
%Outer_Network_No+1
    'Position', [160 120 190 150],'LimitDataPoints','off');
```

```
add_line(Topology_Name,'Main Computer/2',['Scope' num2str(Outer_Network_No+1)
'/1']);  %Outer_Network_No+1

for j=0:1:(Num_of_Gate-1)

    %network number
    Network_ID=j+1;
    Gate_ID=(j+1)*100+1;  %used for naming Gateway Computers



    % Generate DRIVER NETWORKS (Set Block Parameters, Add Scope,
    Connect Blocks)
          add_block('truetime/TrueTime Network',[ [Topology_Name '/Driver
      Network'] num2str(Network_ID)],...
        'Position', [210 (400+900*j) 270 (450+900*j)]);
          set_param([ [Topology_Name '/Driver Network']
      num2str(Network_ID)],...

      'nnodes',num2str(Num_of_Drivers+Num_of_Sensor_Nodes+1),'nwnbr',num2str(
      Network_ID),'nwtype','CSMA/CD
      (Ethernet)','rate','100000000','minsize','512');

      add_block('built-in/Scope', [gcs,['/Scope' num2str((j+1))]],...
        'Position', [310 (390+900*j) 330
      (410+900*j)],'LimitDataPoints','off');
      add_line(Topology_Name,['Driver Network' num2str(Network_ID) '/1'],...
        ['Scope' num2str((j+1)) '/1']);

      % Generate GATEWAY(GW) COMPUTERS
      add_block('truetime/TrueTime Kernel', [ [Topology_Name '/Gateway
      Computer'] num2str(Network_ID)],...
        'Position', [(370) (400+900*j) (430) (450+900*j)]);


      % Set Block Parameters of First GW
      if j==0
      set_param([ [Topology_Name '/Gateway Computer']
      num2str(Network_ID)],...
            'sfun',init_function,'args',['['
      num2str(Network_ID*Max_Driver_No) ']'],'ninputsoutputs','[0 9]',...
            'nwnodenbr',['[', num2str(Network_ID),' ', '1',';'
      num2str(Outer_Network_No),' ', '1]']);
    end

      % Add Scope to gateway computers, Connect Blocks
          add_block('built-in/Scope', [gcs,['/Scope DATA'
      num2str(Gate_ID)]],...
              'Position', [(470) (360+900*j) (490)
      (380+900*j)],'LimitDataPoints','off');
    add_line(Topology_Name,['Gateway Computer' num2str(Network_ID)

'/1'],['Scope DATA' num2str(Gate_ID) '/1']);
```

```
        add_block('built-in/Scope', [gcs,['/Scope Schedule'
    num2str(Gate_ID)]],...
            'Position', [(470) (440+900*j) (490)
    (460+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Gateway Computer' num2str(Network_ID)
    '/2'],['Scope Schedule' num2str(Gate_ID) '/1']);

%Mux 1
    add_block('built-in/Mux',[gcs,'/Mux_Data' num2str(Network_ID)],...
        'orientation','right',...
        'inputs',num2str(Num_of_Drivers),...
        'position',[1050 (150+80*((Num_of_Drivers/2)-1)+900*j) 1100
(180+80*((Num_of_Drivers/2)-1)+900*j)])

    add_block('built-in/Scope', [gcs,['/Scope MUX_DATA'
num2str(Network_ID)]],...
        'Position', [1150 (150+80*((Num_of_Drivers/2)-1)+900*j) 1200
(180+80*((Num_of_Drivers/2)-1)+900*j)],'LimitDataPoints','off');

    add_line(Topology_Name,['Mux_Data' num2str(Network_ID) '/1'],['Scope
MUX_DATA' num2str(Network_ID) '/1'  ]);


    %Mux 2
    add_block('built-in/Mux',[gcs,'/Mux_Sch' num2str(Network_ID)],...
        'orientation','right',...
        'inputs',num2str(Num_of_Drivers),...
        'position',[1050 (300+80*((Num_of_Drivers/2)-1)+900*j) 1100
(330+80*((Num_of_Drivers/2)-1)+900*j)])

    add_block('built-in/Scope', [gcs,['/Scope MUX_SCH'
num2str(Network_ID)]],...

        'Position', [1150 (300+80*((Num_of_Drivers/2)-1)+900*j) 1200
(330+80*((Num_of_Drivers/2)-1)+900*j)],'LimitDataPoints','off');

    add_line(Topology_Name,['Mux_Sch' num2str(Network_ID) '/1'],['Scope
MUX_SCH' num2str(Network_ID) '/1'  ]);


% Generate MOTOR DRIVERS (Set Block Parameters, Add Scope, Connect Blocks)
    for i=1:1:(Num_of_Drivers)
        Driver_ID=(j+1)*1000+i; %used for naming Motor Drivers

% SENSOR Block creation
    if mod(i,5) == 0

        sensor_no = (i / 5);
        sensor_net_no = 1100 + sensor_no;
        sensor_node_no = Num_of_Drivers + sensor_no + 1;
```

```
        add_block('truetime/TrueTime Kernel', [ [Topology_Name '/Sensor
Node'] num2str(sensor_no)],...
        'Position', [(370) ((80*i)-220) (430) ((80*i)-170)]);

        set_param([ [Topology_Name '/Sensor Node'] num2str(sensor_no)],...
        'sfun',init_function,'args',['[' num2str(sensor_net_no)
']'],'ninputsoutputs','[1 9]',...
        'nwnodenbr',['[', '1',' ',num2str(sensor_node_no),']']);


%adding scope to sensor nodes
        add_block('built-in/Scope', [gcs,['/Scope Sensor DATA'
num2str(sensor_no)]],...
        'Position', [(470) ((80*i)-240) (490) ((80*i)-
220)],'LimitDataPoints','off');
        add_line(Topology_Name,['Sensor Node' num2str(sensor_no)
'/1'],['Scope Sensor DATA' num2str(sensor_no) '/1']);

        add_block('built-in/Scope', [gcs,['/Scope Sensor Schedule'
num2str(sensor_no)]],...
        'Position', [(470) ((80*i)-150) (490) ((80*i)-
130)],'LimitDataPoints','off');                add_line(Topology_Name,['Sensor
Node' num2str(sensor_no) '/2'],['Scope Sensor Schedule' num2str(sensor_no)
'/1']);

end
add_block('truetime/TrueTime Kernel', [ [Topology_Name '/Motor Driver']
num2str(Network_ID*Max_Driver_No+i)],...            'Position', [(570 )
(20+80*(i-1)+900*j) (590 ) (60+80*(i-1)+900*j)]);

        set_param([ [Topology_Name '/Motor Driver']
num2str(Network_ID*Max_Driver_No+i)],...
            'sfun',init_function,'args',['['
num2str(Network_ID*Max_Driver_No+i) ']'],...
            'ninputsoutputs','[0 5]','nwnodenbr',['[', num2str(j+1),' ',
num2str(i+1) ']']);



add_block('built-in/Scope', [gcs,['/Scope D' num2str(Driver_ID)]],...
            'Position', [(650 ) (10+80*(i-1)+900*j) (680) (40+80*(i-
1)+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i) '/1'],['Scope D' num2str(Driver_ID)
'/1']);

        add_block('built-in/Scope', [gcs,['/Scope Sch'
num2str(Driver_ID)]],...  %Const2+Driver_ID
            'Position', [(650 ) (50+80*(i-1)+900*j) (680) (80+80*(i-
1)+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i) '/2'],['Scope Sch' num2str(Driver_ID)
'/1']);  %Const2+Driver_ID
```

```
        %mux1
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i) '/1'],['Mux_Data'
num2str(Network_ID) '/' num2str(i) ]);

        %mux2
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i) '/2'],['Mux_Sch'
num2str(Network_ID) '/' num2str(i) ]);

    end

end

%SENSOR NODE - SUM CONNECTIONS
        for b = 1:1:(Num_of_sensor_per_car)
            add_line(Topology_Name,'Sum/1',['Sensor Node' num2str(b)
'/1']);

        end

        for c = 1:1:(Num_of_cars - 1)
            for b = 1:1:(Num_of_sensor_per_car)
                add_line(Topology_Name,['Sum' num2str(c) '/1'],['Sensor
Node' num2str(b + Num_of_sensor_per_car*c) '/1']);

            end
        end
% Set a couple of model parameters to eliminate warning messages
set_param(gcs,'StartTime','0.0');
set_param(gcs,'StopTime','2.0');
set_param(gcs,'Solver','ode45');

% Save the model
save_system(fname);

% Open the model
uiopen(['C:\Users\username\Desktop\network_topologies\directory\to\your
\topology\folder\' Topology_Name '.slx'],1)

% Close truetime window
close_system('truetime');
```

As we see above, we can create kernel blocks in loops to have multiples of it, as we can also create network blocks in loops to have multiples of. The example above first creates one main computer on the outer network, then goes into a for loop and in that loop it creates a network block as the driver network and then it also creates a kernel block as the gateway computer connected to both the outer network and the driver network. Later on, with a second for loop within the first one we can create the motor drivers the same way we created the gateway computers.

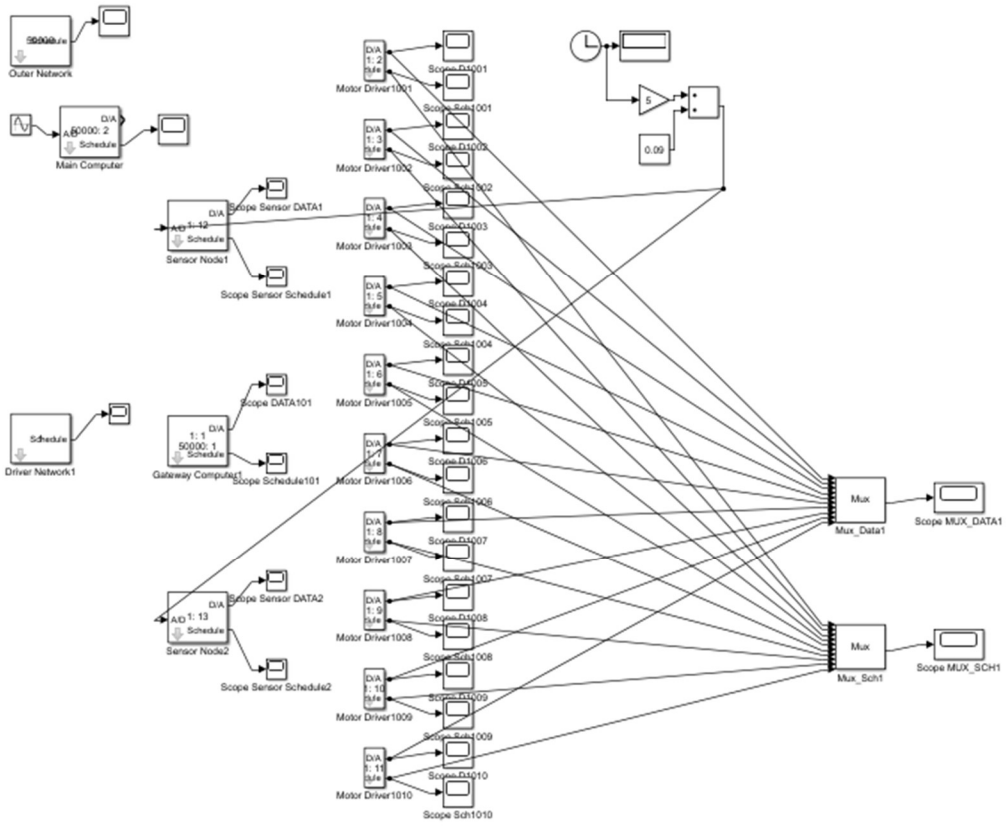

Figure 37: Topology G Simulink Model

TrueTime Scripts and Functions

Once our create topology function is ready, we need to prepare the initialize function for the kernel and network blocks. If we try to run the create topology function without the initialization function, we will get errors as networks and kernels are not initialized. We can have separate initialization functions as well as having all of them in one file. In our project we gathered all in one function to manage them easier. We need to define parameters, create tasks and interrupt handlers in the initialization function.

Initialization Function

```
function initialization(mode)

% calling configuration file for parameters
config;

% Initialize TrueTime kernel
ttInitKernel('prioDM');   % deadline-monotonic scheduling

% Define the parameters
msg.sender_ID=0; %% node number of the sender node
msg.outer=0; %% the message sent by main computer via outer network
msg.pos_of_sensor=0; %% position of the sensor
msg.des_ID=0; %% node number of the destination node
msg.timestamp_sender=0;

data.kernel_ID = mod(mode,1000);
data.network_ID = (mode - data.kernel_ID)/1000;

data.num_of_drivers=Num_of_Drivers;
data.num_of_gateways=Num_of_Gate;
data.sensor_offset=sensor_offset;
data.driver_length=driver_length;
data.Sense_node_gap=Sense_node_gap;

data.pos_down=(data.network_ID - 1) * data.num_of_drivers *
data.driver_length + (data.kernel_ID - 1)*data.driver_length;
%%lower limit of the position of the driver

data.pos_up= (data.network_ID - 1) * data.num_of_drivers * data.driver_length
+ (data.kernel_ID)*data.driver_length;
%%upper limit of the position of the driver

if data.network_ID<data.num_of_gateways
    data.lower_gw_network_no=1000+(data.network_ID); %GOKALP ID
end
```

```
if data.network_ID>1
    data.upper_gw_network_no=1000+(data.network_ID-1); %GOKALP ID-1
end

if mode == 50000 % (main computer)
    starttime = 0.0;
    period = 0.010;
    ttCreatePeriodicTask(['Main_computer_task' num2str(mode)], starttime,
period, 'Main_computer',data);
end

% (gateway computers)
if (mode == 1000)
    prio = 1.0;
    ttCreateHandler(['Gateway_computers_outer_network_Task' num2str(mode)],
prio, 'Gateway_computers_outer_network', data);
    ttAttachNetworkHandler(50000,['Gateway_computers_outer_network_Task'
num2str(mode)])
end

% (sensor nodes)
if (mode - mod(mode,100) == 1100)
    starttime = 0.0;
    period = 0.000100; %GOKALP 10kHz
    ttCreatePeriodicTask(['Sensor_nodes_Task' num2str(mode)], starttime,
period, 'Sensor_nodes', data);
end

% (drivers)
if ((data.kernel_ID>0) && (mode - mod(mode,100) ~= 1100))
    % Network handler
    prio = 1.0;
    ttCreateHandler(['Motor_Drivers_Task' num2str(mode)], prio,
'Motor_Drivers',data);
    ttAttachNetworkHandler(data.network_ID,['Motor_Drivers_Task'
num2str(mode)])
end
```

APPENDIX D

Create Topology (Topology R)

```
clear all
%clc
% call config.m for all the variables and settings required
config;

% Specify the name of the model to create on config file
fname = Topology_Name;

% Check if the file already exists and delete it if it does
if exist(fname,'file') == 4
    % If it does then check whether it's open
    if bdIsLoaded(fname)
        % If it is then close it (without saving!)
        close_system(fname,0)
    end
    % delete the file
    delete([fname,'.slx']);
end

% Create the system
new_system(fname);

% Open truetime library and the created system to copy(add) blocks
open_system('truetime');
open_system(Topology_Name);

% Add Clock and Display
add_block('built-in/Clock', [gcs,'/Clock'],'Position', [1000 20 1030 50]);
add_block('built-in/Display', [gcs,'/Display'],'Position', [1050 20 1100
50]);

% Connect Clock and Display
add_line(Topology_Name,'Clock/1','Display/1');

% Add Gain
add_block('built-in/Gain', [gcs,'/Gain'],...
    'Position', [1050 150 1080 180],...
    'Gain',num2str(motor_speed),'SampleTime','-1');

% Connect Clock and Gain
add_line(Topology_Name,'Clock/1','Gain/1');

% Add Sum
add_block('built-in/Sum',[gcs,'/','Sum'])
```

```
% Set Sum Block Parameters
set_param([gcs,'/','Sum'],...
    'inputs','++',...
    'position',[1100,150,1130,180])

% Connect Gain and Sum
add_line(Topology_Name,'Gain/1','Sum/1');

% Add Constant
add_block('built-in/Constant', [gcs,'/Constant'],...
    'Position', [1050 250 1080 280],...
    'Value',num2str(driver_length),'SampleTime','inf');

% Connect First Output of Constant Block to Second Input of Sum Block
add_line(Topology_Name,'Constant/1','Sum/2');

% Generate OUTER NETWORK BLOCK (Set Block Parameters, Add Scope, Connect
Blocks)
add_block('truetime/TrueTime Network',[Topology_Name '/Outer Network'],...
    'Position', [10 (20) 70 (70)]);

set_param([Topology_Name '/Outer Network'],...

'nnodes',num2str(Num_of_Gate+1),'nwnbr',num2str(Outer_Network_No),'nwtype','C
SMA/CD (Ethernet)','rate','100000000','minsize','1144');

add_block('built-in/Scope', [gcs,['/Scope' num2str(Outer_Network_No)]],...
    'Position', [100 (10) 130 (40)],'LimitDataPoints','off');
add_line(Topology_Name,'Outer Network/1',...
    ['Scope' num2str(Outer_Network_No) '/1']);


% Generate MAIN COMPUTER BLOCK (Set Block Parameters, Add Sine Wave Block,
Add Scope, Connect Blocks)
add_block('built-in/Sin', [gcs,'/Sine Wave'],...
    'Position', [10 120 30 140],'SampleTime','0' );
add_block('truetime/TrueTime Kernel', [Topology_Name '/Main Computer'],...
    'Position', [60 110 120 160]);
set_param([Topology_Name '/Main Computer'],...
    'sfun',init_function,'args',['[' num2str(Outer_Network_No)
']'],'ninputsoutputs','[1 1]',...
    'nwnodenbr',['[', num2str(Outer_Network_No),' ',
num2str(Num_of_Gate+1),']']);
add_line(Topology_Name,'Sine Wave/1','Main Computer/1' );

add_block('built-in/Scope', [gcs,['/Scope' num2str(Outer_Network_No+1)]],...
%Outer_Network_No+1
    'Position', [160 100 190 130],'LimitDataPoints','off');
add_line(Topology_Name,'Main Computer/1',['Scope' num2str(Outer_Network_No+1)

'/1']);  %Outer_Network_No+1
```

```
add_block('built-in/Scope', [gcs,['/Scope' num2str(Outer_Network_No+2)]],...
%Outer_Network_No+2
    'Position', [160 150 190 180],'LimitDataPoints','off');
add_line(Topology_Name,'Main Computer/2',['Scope' num2str(Outer_Network_No+2)
'/1']);  %Outer_Network_No+2


% Nested for loop is used to create Driver Networks,Gateway Networks,Motor
Drivers and Gateway Computers.

%if-else statement is necessary for satisfying the conditions related to
different blocks.
for j=0:1:(Num_of_Gate-1)
    Network_ID=j+1;  %network number
    Gate_ID=(j+1)*100+1;  %used for naming Gateway Computers Scopes

    % Generate Driver NETWORKS (Set Block Parameters, Add Scope, Connect
Blocks)
    add_block('truetime/TrueTime Network',[ [Topology_Name '/Driver Network']
num2str(Network_ID)],...
        'Position', [210 (400+900*j) 270 (450+900*j)]);
    set_param([ [Topology_Name '/Driver Network'] num2str(Network_ID)],...

'nnodes',num2str(Num_of_Drivers+2+1),'nwnbr',num2str(Network_ID),'nwtype','CS
MA/CD (Ethernet)','rate','100000000','minsize','125*8');

    add_block('built-in/Scope', [gcs,['/Scope' num2str((j+1))]],...
        'Position', [310 (390+900*j) 330
(410+900*j)],'LimitDataPoints','off');
    add_line(Topology_Name,['Driver Network' num2str(Network_ID) '/1'],...
        ['Scope' num2str((j+1)) '/1']);

    %creating NIC
    for i=1:1:(Num_of_Drivers+2)
        Driver_ID=(j+1)*1000+i; %used for naming Motor Drivers
        NIC_ID=(j+1)*1000+i;

        add_block('truetime/TrueTime Kernel', [ [Topology_Name '/NIC']
num2str(Network_ID*Max_Driver_No+i)],...
            'Position', [(550 ) (20+80*(i-1)+900*j) (590 ) (60+80*(i-
1)+900*j)]);

        if i<3
            set_param([ [Topology_Name '/NIC']
num2str(Network_ID*Max_Driver_No+i)],...

            'sfun',init_function,'args',['['
num2str(Network_ID*Max_Driver_No+i) ']'],...

            'ninputsoutputs','[1 1]','nwnodenbr',['[', num2str(j+1),' ',
num2str(i) ']']);
```

```
        else
            set_param([ [Topology_Name '/NIC']
num2str(Network_ID*Max_Driver_No+i)],...
            'sfun',init_function,'args',['['
num2str(Network_ID*Max_Driver_No+i) ']'],...

            'ninputsoutputs','[0 1]','nwnodenbr',['[', num2str(j+1),' ',
num2str(i) ']']);

        end

        %add scope
        add_block('built-in/Scope', [gcs,['/NIC Scope D'
num2str(Driver_ID)]],...
            'Position', [(650 ) (10+80*(i-1)+900*j) (680) (40+80*(i-
1)+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['NIC' num2str(Network_ID*Max_Driver_No+i)
'/1'],['NIC Scope D' num2str(Driver_ID) '/1']);

        add_block('built-in/Scope', [gcs,['/NIC Scope Sch'
num2str(Driver_ID)]],...  %Const2+Driver_ID
            'Position', [(650 ) (50+80*(i-1)+900*j) (680) (80+80*(i-
1)+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['NIC' num2str(Network_ID*Max_Driver_No+i)
'/2'],['NIC Scope Sch' num2str(Driver_ID) '/1']);  %Const2+Driver_ID


    end



     % Generate GATEWAY(GW) COMPUTERS
    add_block('truetime/TrueTime Kernel', [ [Topology_Name '/Gateway
Computer'] num2str(Network_ID)],...
        'Position', [(370) (20+900*j) (430) (60+900*j)]);

    set_param([ [Topology_Name '/Gateway Computer'] num2str(Network_ID)],...
            'sfun',init_function,'args',['['
num2str(Network_ID*Max_Driver_No) ']'],'ninputsoutputs','[0 1]',...
            'nwnodenbr',['[', num2str(Outer_Network_No),' ',
num2str(j+1),']']);

        add_line(Topology_Name,['Gateway Computer' num2str(Network_ID)
'/1'],['NIC' num2str(Network_ID*Max_Driver_No+1) '/1']);

         %add scope
        add_block('built-in/Scope', [gcs,['/Scope Gate D'
num2str(Network_ID)]],...
            'Position', [(470) (900*j-10) (500)
(900*j+20)],'LimitDataPoints','off');
        add_line(Topology_Name,['Gateway Computer' num2str(Network_ID)
'/1'],['Scope Gate D' num2str(Network_ID) '/1']);
```

```
        add_block('built-in/Scope', [gcs,['/Scope Gate Sch'
num2str(Network_ID)]],...
            'Position', [(470) (50+900*j) (500)
(80+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Gateway Computer' num2str(Network_ID)
'/2'],['Scope Gate Sch' num2str(Network_ID) '/1']);



        %create sensor node
        add_block('truetime/TrueTime Kernel', [ [Topology_Name '/Sensor
Node'] num2str(Network_ID)],...
        'Position', [(370) (150+900*j) (430) (190+900*j)]);

        set_param([ [Topology_Name '/Sensor Node'] num2str(Network_ID)],...
        'sfun',init_function,'args',['['
num2str(Network_ID*Max_Driver_No+100) ']'],'ninputsoutputs','[1 1]',...
        'nwnodenbr',['[', num2str(Network_ID),' ', num2str(8),']']);

        add_line(Topology_Name,'Sum/1',['Sensor Node' num2str(Network_ID)
'/1']);

        add_line(Topology_Name,['Sensor Node' num2str(Network_ID)
'/1'],['NIC' num2str(Network_ID*Max_Driver_No+2) '/1']);


          %add scope
        add_block('built-in/Scope', [gcs,['/Scope Sensor D'
num2str(Network_ID)]],...
            'Position', [(470) (120+900*j) (500)
(150+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Sensor Node' num2str(Network_ID)
'/1'],['Scope Sensor D' num2str(Network_ID) '/1']);

        add_block('built-in/Scope', [gcs,['/Scope Sensor Sch'
num2str(Network_ID)]],...
            'Position', [(470) (180+900*j) (500)
(210+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Sensor Node' num2str(Network_ID)
'/2'],['Scope Sensor Sch' num2str(Network_ID) '/1']);


        % Add Mux
        %Mux 1
        add_block('built-in/Mux',[gcs,'/Mux_Data' num2str(Network_ID)],...
            'orientation','right',...
            'inputs',num2str(Num_of_Drivers),...
            'position',[950 (150+80*((Num_of_Drivers/2)-1)+900*j) 1000
(180+80*((Num_of_Drivers/2)-1)+900*j)])
```

```
        add_block('built-in/Scope', [gcs,['/Scope MUX_DATA'
num2str(Network_ID)]],...
            'Position', [1150 (150+80*((Num_of_Drivers/2)-1)+900*j) 1200
(180+80*((Num_of_Drivers/2)-1)+900*j)],'LimitDataPoints','off');

        add_line(Topology_Name,['Mux_Data' num2str(Network_ID) '/1'],['Scope
MUX_DATA' num2str(Network_ID) '/1'  ]);


        %Mux 2
        add_block('built-in/Mux',[gcs,'/Mux_Sch' num2str(Network_ID)],...
            'orientation','right',...
            'inputs',num2str(Num_of_Drivers),...
            'position',[950 (300+80*((Num_of_Drivers/2)-1)+900*j) 1000
(330+80*((Num_of_Drivers/2)-1)+900*j)])

        add_block('built-in/Scope', [gcs,['/Scope MUX_SCH'
num2str(Network_ID)]],...
            'Position', [1150 (300+80*((Num_of_Drivers/2)-1)+900*j) 1200
(330+80*((Num_of_Drivers/2)-1)+900*j)],'LimitDataPoints','off');

        add_line(Topology_Name,['Mux_Sch' num2str(Network_ID) '/1'],['Scope
MUX_SCH' num2str(Network_ID) '/1'  ]);


    %creating motor drivers
    for i=3:1:(Num_of_Drivers+2)
        Driver_ID=(j+1)*1000+i+2; %used for naming Motor Drivers
        %NIC_ID=(j+1)*1000+i;

        add_block('truetime/TrueTime Kernel', [ [Topology_Name '/Motor
Driver'] num2str(Network_ID*Max_Driver_No+i-2)],...
            'Position', [(750 ) (20+80*(i-1)+900*j) (790 ) (60+80*(i-
1)+900*j)]);

        set_param([ [Topology_Name '/Motor Driver']
num2str(Network_ID*Max_Driver_No+i-2)],...
            'sfun',init_function,'args',['['
num2str(Network_ID*Max_Driver_No+i+100-2) ']'],...
            'ninputsoutputs','[1 2]','nwnodenbr',['[',']'

        add_line(Topology_Name,['NIC' num2str(Network_ID*Max_Driver_No+i)
'/1'],['Motor Driver' num2str(Network_ID*Max_Driver_No+i-2) '/1']);

        %add scope
        add_block('built-in/Scope', [gcs,['/Scope D' num2str(Driver_ID)]],...
            'Position', [(850 ) (10+80*(i-1)+900*j) (880) (40+80*(i-
1)+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i-2) '/1'],['Scope D' num2str(Driver_ID)
'/1']);
```

```
        add_block('built-in/Scope', [gcs,['/Scope Sch'
num2str(Driver_ID)]],...  %Const2+Driver_ID
            'Position', [(850 ) (50+80*(i-1)+900*j) (880) (80+80*(i-
1)+900*j)],'LimitDataPoints','off');
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i-2) '/2'],['Scope Sch' num2str(Driver_ID)
'/1']);  %Const2+Driver_ID


        %mux1
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i-2) '/1'],['Mux_Data' num2str(Network_ID)
'/' num2str(i-2) ]);

        %mux2
        add_line(Topology_Name,['Motor Driver'
num2str(Network_ID*Max_Driver_No+i-2) '/2'],['Mux_Sch' num2str(Network_ID)
'/' num2str(i-2) ]);
    end

    if(Network_ID == 1)
    % Add Mux
        %Mux 1
        add_block('built-in/Mux',[gcs,'/Mux_Data_MUX'
num2str(Network_ID)],...
            'orientation','right',...
            'inputs',num2str(Num_of_Gate),...
            'position',[1350 (150+80*((Num_of_Drivers/2)-1)+900*j) 1400
(180+80*((Num_of_Drivers/2)-1)+900*j)])

        %Mux 2
        %GOKALP
        add_block('built-in/Mux',[gcs,'/Mux_Sch_MUX' num2str(Network_ID)],...
            'orientation','right',...
            'inputs',num2str(Num_of_Gate),...
            'position',[1350 (300+80*((Num_of_Drivers/2)-1)+900*j) 1400
(330+80*((Num_of_Drivers/2)-1)+900*j)])


        add_block('built-in/Scope', [gcs,['/Scope ALL_DATA'
num2str(Network_ID)]],...
            'Position', [1450 (150+80*((Num_of_Drivers/2)-1)+900*j) 1500
(180+80*((Num_of_Drivers/2)-1)+900*j)],'LimitDataPoints','off');

        add_line(Topology_Name,['Mux_Data_MUX' num2str(Network_ID)
'/1'],['Scope ALL_DATA' num2str(Network_ID) '/1'  ]);

        add_block('built-in/Scope', [gcs,['/Scope ALL_SCH'
num2str(Network_ID)]],...
            'Position', [1450 (300+80*((Num_of_Drivers/2)-1)+900*j) 1500
(330+80*((Num_of_Drivers/2)-1)+900*j)],'LimitDataPoints','off');
```

```
          add_line(Topology_Name,['Mux_Sch_MUX' num2str(Network_ID)
'/1'],['Scope ALL_SCH' num2str(Network_ID) '/1'  ]);

    end


        add_line(Topology_Name,['Mux_Data' num2str(Network_ID)
'/1'],['Mux_Data_MUX' num2str(1) '/' num2str(Network_ID)]);
        add_line(Topology_Name,['Mux_Sch' num2str(Network_ID)
'/1'],['Mux_Sch_MUX' num2str(1) '/' num2str(Network_ID)]);

end

% Set a couple of model parameters to eliminate warning messages
set_param(gcs,'StartTime','0.0');
set_param(gcs,'StopTime','0.5');
set_param(gcs,'Solver','ode45');

% Save the model
save_system(fname);

% Open the model
uiopen(['C:\Users\*username*\Desktop\EtherCAT\' Topology_Name '.slx'],1)

% Close truetime window
close_system('truetime');
%close('Figure 1');
```