Publish-subscribe based Middleware for Heterogeneous Critical Infrastructure Systems Communication

By

Titus Okathe

A Thesis Submitted in Partial Fulfillment

Of the Requirements for the Degree of

Master of Applied Science

In

The Faculty of Engineering and Applied Science

Department of Electrical/Computer Engineering

University of Ontario Institute of Technology

July 2014

© Titus Okathe 2014
Acknowledgement

I would like to thank first and foremost my supervisors, Shahram Heydari, Vijay Sood, and Khalil El-Khatib. Their encouraging words, guidance, and patience were indispensable to the completion of this thesis. I would also like to thank my loving parents for their support, emotionally and financially throughout my studies. Lastly, to all my friends and fellow researchers in the Advanced Networking Technologies and Security Lab, I say thank you.
Abstract

Critical Infrastructures (CIs) are physical assets and organizations responsible for the production and distribution of society’s vital goods and services. The increasing interconnection of CIs has resulted in interdependencies which effect the propagation of failure from one infrastructure to another. Therefore a publish-subscribe based communication system for dissimilar CIs is presented.

The proposed system improves the manageability of CIs by providing an exchange medium for status information and alerts. It achieves this via a uniform architecture within and across infrastructure boundaries, that maintains data restrictions that reflect real life organizational, administrative, and policy boundaries.

Finally the proposed system is modeled using the OMNET++ simulation framework, and a network performance study investigating scalability is presented. Scalability was found to depend on service time per packet, subscription density, and number of clients per router. However, further work in the areas of QoS management, reliability/robustness, security, and network optimization is required.

**Keywords**: Critical infrastructure, Publish-Subscribe, Interdependency, OMNET++
# Table of Contents

Table of Contents

1 Introduction

1.1 Problem Statement

1.2 Contribution

1.3 Thesis Organization

2 CI interdependency modelling and Information Sharing

2.1 CI Monitoring and control

2.2 Related Works

2.2.1 Integrated Risk Reduction of Information-based Infrastructure Systems (IRRIIS)

2.2.2 Tool for systemic risk analysis and secure mediation of data exchanged across linked CI information infrastructures (MICIE)

2.2.3 GridStat

2.2.4 Unified Incident Command and Decision Support (UICDS)

2.3 Summary

3 Publish-Subscribe And overlay Networks

3.1.1 Basic Architecture

3.1.2 Types of Publish-Subscribe Model

4 System Architecture

4.1.1 Broker

4.1.2 Routers

4.1.3 Publishers

4.1.4 Subscribers
4.2 Network Architecture

4.2.1 Message Structure

4.2.2 Start up and Service Discovery

4.2.3 Matching Subscriptions

4.3 System Publish Subscribe Model

4.3.1 Data model

4.3.2 Subscription Language Model

4.3.3 Matching/Filtering Events

5 SIMULATION MODEL

5.1 Overview

5.2 Simulation Modules

5.2.1 Simple Modules

5.2.2 Compound Modules and Nodes

5.3 OMNET++ Packet

5.4 Routing Mechanism

6 Simulation Results and Analysis

6.1 Simulation Setup

6.2 Network Performance and Subscription Density

6.2.1 Packet Arrival Rate

6.2.2 End to End delay

6.2.3 Processing Delay

6.2.4 Queuing Delay

6.2.5 Data Inter-arrival time

6.2.6 Queue Size
7 Conclusion and Future Works .................................................................76
8 References..............................................................................................78
Figures

Figure 1  Commercial versus industrial network architecture ........................................10
Figure 2  Overview of the MIT Architecture ..................................................................14
Figure 3  MICIE Architecture  [18] ................................................................................15
Figure 4  SMGW Architecture  [18] ................................................................................16
Figure 5  System Architecture diagram .........................................................................28
Figure 6  Broker Module Architecture ..........................................................................29
Figure 7  Router Architecture ........................................................................................30
Figure 8  Network stack of router and clients ..................................................................31
Figure 9  Unicast datagram frame ..................................................................................33
Figure 10 Multicast datagram frame ...............................................................................34
Figure 11 Hierarchical topology ......................................................................................35
Figure 12 Flat Topology ...................................................................................................36
Figure 13 System client start up timing diagram ..............................................................37
Figure 14 Publisher timing diagram .................................................................................38
Figure 15 Subscription Flow chart .................................................................................39
Figure 16 Subscription Matching flow chart ..................................................................40
Figure 17 UML Diagram data model .............................................................................42
Figure 18 Data Advertisement UML class diagram ..........................................................43
Figure 19 Subscription Class UML diagram ..................................................................44
Figure 20 Matching and Filtering Classes ......................................................................45
Figure 21 Simple subscription XML document .............................................................46
Figure 22 OMNET++ Component Architecture .............................................................47
Figure 23  CloudBrokerNode Schematic .................................................................52
Figure 24  EdgeRouterNode Schematic .................................................................53
Figure 25  Simple Network showing nodes connected to form a hierarchical network ....55
Figure 26  Simulation Model Packet Class Diagram ................................................56
Figure 27  ForwardRule UML Diagram ..................................................................57
Figure 28  Total number of active subscriptions in the network: Series one-five client per
   group ..................................................................................................................61
Figure 29  Packet arrival rate for networks ..................................................................62
Figure 30  Maximum End to End delay by Network ....................................................63
Figure 31  Processing delay and its effect on End-to-End delay ....................................64
Figure 32  Mean maximum end to end delay versus the service time .........................65
Figure 33  Queueing time with a limited queue length versus infinite queue length in
   logarithmic time scale .........................................................................................67
Figure 34  Percentage of Subscriptions satisfied within 1% ......................................68
Figure 35  Data Inter-arrival rate comparison at 10 µs service time ...............................70
Figure 36  Data Inter-arrival rate comparison at 200µs service time ............................70
Figure 37  Data inter-arrival time comparison 500µs ...................................................71
Figure 38  Data inter-arrival time comparison 1 ms .....................................................71
Figure 39  End to end delay comparison between a limited queue size and an unlimited
   queue size at 10 µs service time ........................................................................72
Figure 40  End to end delay comparison between limited and unlimited queue size at 50us
   service time .......................................................................................................72
Figure 41  End to End delay comparison at 200 µs service time between limited and
   unlimited queue scenarios ..................................................................................73
Figure 42  Comparison of maximum queuing time between limited and unlimited queue at 200 µs...............................................................................................................................................74

Figure 43  End to End delay comparison at 500us between limited and unlimited queue scenarios........................................................................................................................................75

Figure 44  End to End delay comparison between limited and unlimited queue comparison at 1 ms service time per packet ........................................................................................................75
Tables

Table 1 DCS versus SCADA comparison .................................................................9

Table 2 Difference between Commercial networks and Industrial networks [15] ..........11

Table 3 Generate topology parameters ....................................................................58

Table 4 Test Network parameters .............................................................................60

Table 5 Average queuing for data packets versus the number of clients per router ....66

Table 6 Data Inter-arrival rate performance ..................................................................68
CHAPTER ONE

1 INTRODUCTION

Modern infrastructures are increasingly becoming interconnected. Life as we know it depends heavily on the production and distribution of certain goods and services, in the absence of which society cannot function. The collection of physical assets, processes and organizations that are responsible for the production and distribution of these essential goods and services are referred to as “Critical Infrastructures” (CIs)[1]. The term CIs encompasses a number of sectors including telecommunications; electric power generation and distribution systems; government services; banking and finance; water distribution systems and waste water management systems; transportation (railways, airways, waterways); and emergency services[2]. CIs constitute a large collection of individual components that interact with one another and with their environment. These interactions leads to the emergence of complex behaviours that cannot be quantified by simply summing up the behaviours of the individual components [2]. Consequently, complex relationships, dependencies and interdependencies arise between the CIs and between their components.

Formally, one infrastructure is dependent on another if the state of the former is influenced by the state of the latter or is correlated with the state of the latter. When this dependency is bidirectional, it is referred to as interdependency. In practice, most critical infrastructures exhibit interdependency. Dependencies may be classified in a number of ways; one classification divides them into physical, cyber, geographic, and logical dependency [2].

Physical dependency exists between infrastructures that rely on the flow of materials from one infrastructure to another, for instance between a gas power plant and the gas pipeline infrastructure. Cyber dependency arises when there is a reliance on information flow between infrastructures, for example between the supervisory control and data acquisition (SCADA) network of the electric grid and the electric grid itself. Geographic dependency emerges between infrastructures that are in close spatial proximity. This type
of dependency can be found between road networks, natural gas pipelines, underground electricity cables, and or telecommunication fibers that usually follow the same path in a city of municipality. Finally, logical dependency is a type of dependency that does not arise as a result of any of the previously mentioned type of dependency but as a result of human decisions and or policies. For example the halting of commercial airlines in the US after the 911 attack that resulted in slow business leading to layoffs in the airline industries and even some airlines operators filing for bankruptcy [3].

In the course of its operational lifetime, the normal operation of CIs is often interrupted due to a number of reasons. These reasons may be benign such as planned maintenance, system upgrades etc., or they may be threatening, such as component failures, human errors, acts of terrorism, or natural disasters. The effect of undesirable events on a particular CI may become compounded by its dependencies on other CIs. Furthermore, as the CI in which the event occurs fails, it may cause its dependent CIs to fail or at the very least diminish the quality of their operation. This propagation of failure from one infrastructure to another interdependent infrastructure is termed a cascade failure. Cascade failure is a direct consequence of the interdependency between critical infrastructures [4-6]. However the intricate nature of CI independencies makes it difficult to predict how a failure would propagate through the network of interconnected CIs. Further exacerbating this uncertainty is the limited information available to CI operators about the state of interdependent CIs. Therefore CI-CI communication is seen as one way of improving CI resilience.

1.1 Problem Statement

CIs exhibit interdependencies; however, each CI has separate systems for monitoring and controlling the state of the CI. An operator of a given CI has no way of knowing what is going on in a CI which provides services it depends on. Consequently when failures cascade, it is often too late to respond adequately when the system operator becomes aware of the problem. Therefore most solutions to disturbances have been local approaches, attempting to solve a problem that may have a much wider scope. This thesis attempts to solve this problem by facilitating CI-CI communication, both at the system level and at the
component level. Put simply, if CIs depend on one another, they should communicate with one another.

The main objective of this thesis is to design a comprehensive middleware layer that could provide a mechanism to collect and exchange status and alarm data from multiple heterogeneous CIs. The design for the middleware layer should aim to provide:

- **Scalability**: That is the system should maintain reasonable performance when the number of CIs or components in CIs increase.
- **Data filtering**: This refers to not just filtering in the traditional sense of matching data to specific criteria specified by the end user, but also the ability to modify the data before delivering it. This is especially useful in cases where the end user is not authorised to have access to certain aspects of the data for instance for privacy reasons.
- **Flexibility**: The system should be able support new applications without a need for reconfiguration of devices or a system upgrade. In other words it should be relatively trivial to add new applications on top of the existing system.

1.2 **CONTRIBUTION**

This thesis presents a uniform architecture based on publish-subscribe middleware technology to facilitate the communication between dissimilar CIs. This architecture is called the publish-subscribe middleware for CI communication (PSMCC). The novel approach here is to apply the same principles for CI internal networks as well as the external network interconnecting them. This helps to simplify the process of inter communication as new infrastructure or technologies need to be installed.

The proposed design and analysis includes:

- **Architecture**: An architecture for CI communication that allows the communication of heterogeneous CI systems.
- **Data format**: It specifies a data model that is able to accommodate structured or binary data.
- **Detailed cases/mechanisms**: Description of mechanisms for the following operational cases - subscribing, publishing, subscription matching, and packet forwarding.

- **Performance analysis**: Analysis of an example network under varying conditions to investigate the parameters that have the highest effect on performance.

To show the viability of this design it is modeled in the OMNET++ simulation framework and tested for scalability, and the results presented.

### 1.3 Thesis Organization

The rest of this thesis is organised as follows. Chapter two elaborates on the challenges of CI-CI communication by looking at some works that have attempted to enhance CI-CI communication. Chapter three presents background on the concept of publish-subscribe and message-oriented-middleware. In chapter four, the proposed architecture is presented as well as the operation of its constituent parts. Next, chapter five presents the realisation of the architecture in the OMNET++ simulator. In chapter six the simulation scenarios and results are presented. Chapter seven concludes this thesis with the conclusions and future work.
CHAPTER TWO

2 CI INTERDEPENDENCY MODELLING AND INFORMATION SHARING

There have been a number of efforts to model the effects of interdependency among multiple heterogeneous infrastructures. In [3] the authors present a summary of these many efforts. Although there has been much progress in the area of modelling CI dependencies there are however a number of limiting factors. By far, the most challenging issue is the availability of data required to develop the models in the first place. This kind of data includes topology information, which addresses how the infrastructure is interconnected. Also of importance is how these infrastructures are controlled in terms of normal and emergency procedures. Furthermore, government and corporate policies may also influence how an infrastructure is operated. This data is not easily available as most CIs are privately operated and therefore access to some of these data may be restricted. However there have been a number of government programs to mediate the exchange of information [7, 8].

Another barrier to CI modeling is the timely availability of data, especially when real-time monitoring and risk assessment is required. In this case it becomes important for a CI operator to have timely access to information about the state of their infrastructure as well as infrastructures that they may depend on. A practical example is a gas power plant that needs to know if there is going to be a potential loss of gas supply due to a gas valve failure. In this case the utility operator can anticipate a potential decrease in electricity output and therefore take proactive actions to mitigate the incident if and when it occurs. Also integrating models into existing systems for validation is difficult if not impossible. This is because most CI communication networks are not very flexible and the diversity of protocols makes it difficult to develop new applications on top of existing communication infrastructure.
CIs are complex system of systems, as they comprise of many systems and subsystems that interact and change as a result of these interactions. Therefore they may be classified as complex adaptive systems (CASs) [2, 9]. CASs are made up of populations of agents. CASs agents are individual actors in a complex environment that are capable of interacting with their environment as well as other agents. In [2], an agent is seen as having a location, capabilities, and memory. The agent’s location defines its location in a physical or abstract space. Its capabilities refer to what it can do from that location, and memory refers to what it has experience or its history. As seen from a CAS perspective, the management of multiple CIs requires a holistic approach. In other words, a given infrastructure cannot be adequately managed without taking into account its interactions with other infrastructures and its environment. Consequently, to predict the future state of a given infrastructure, it is important to consider inputs from its environment as well as the state of other interconnected infrastructures, where interconnected here refers to the existence of one or more forms of dependencies between the infrastructure under consideration and other infrastructures. This requires an unprecedented collaboration between CI systems, in the form of sharing status information across organisational boundaries. Furthermore it opens up the opportunities for managing multiple dissimilar CI systems. Increasingly CI systems are becoming more integrated as cities grow and many areas become urbanised. By allowing infrastructure operators (humans or systems) to have access to information that affects their operations they can take a more proactive rather than reactive approach towards managing these infrastructures. This has the effect of improving infrastructure resilience. Where resilience here is not necessarily the absence of failure, but the ability of a system to fail in a predictable way that is easy to recover from.

In its Action Plan [10], Public Safety Canada lists information sharing as a key aspect of improving CI protection and resilience. Similarly, the European Commission proposed the development of a Critical Infrastructure Warning Network (CIWIN) under its European Programme for Critical Infrastructure Protection (EPCIP)[11]. In the United States, the Department of Homeland Security (DHS) also has started several initiatives with regards to CI protection and information sharing. In particular, the National Infrastructure
Coordinating Center supports situational awareness, information sharing and collaboration amongst CI partners, assessment and analysis, and decision support [12].

As information sharing is important to CI resilience, this thesis proposes an architecture for CI-CI communication. The approach here is to consider primarily the exchange of alerts and status information. In this light there are two primary challenges with regards to information sharing: what to exchange, and how to exchange information.

Firstly, the “what” deals with what kind of data needs to be exchange, this varies from one type of infrastructure to another. Furthermore, the relevance of a given data varies for different infrastructures. For instance a transformer’s voltage reading may be relevant to the power utility but irrelevant to a telecommunication company whose antenna depends on the power supplied by the transformer in question. A better information would be whether the power will continue to be supplied or not. However, an adjacent utility company may need the voltage at the transformer as this directly affects their interconnected network. Consequently, it may be seen from this example that information is contextual, hence information has to be tailored to the receiver of the information. This problem of what to share is complex one and there have been a number of works addressing this very problem [13, 14].

Secondly, the ‘how’ deals with the medium of exchange. There are a number of technologies available for information exchange. In terms of communication technologies this varies from infrastructure to infrastructure. This variation presents a challenge when managing multiple CIs. However, current communication technologies provide a way to abstract from this physical medium and present a more consistent overview of the network via overlay network technologies. This second problem is what this thesis attempts to solve: Providing a means to allow for the communication between multiple heterogeneous CI systems, in a manner that is independent of the final use of the data.
2.1 CI MONITORING AND CONTROL

The term *critical infrastructure* covers a wide range of sectors. Each sector has different communication needs, and therefore has developed independent methods to meet these needs. For the purpose of discussing CI communication, two distinct classes of CIs are identified: these are Utility Infrastructures and Services Infrastructures. Utility CIs includes electricity, telecommunication, water, oil and gas, and transportation. In contrast Services Infrastructures include banking and finance, emergency services, and government services. This distinction is necessary as the members in each group have similar communication requirements. For instance in utility CIs the aim is usually to monitor the state of the equipment out in the field to ensure they are working optimally. In these types of systems there is also a need to take corrective action by means of some form of control system when undesirable events occur or the system is heading towards unstable operating conditions. This control is usually in the form of nudging a process back within operating limits. Furthermore, these type of CIs are more geographically disperse and contain a greater number of equipment than the services type. Also Utility Infrastructures tend to be more geographically and physically dependent. In contrast the services CIs communication needs are less about controlling a physical system but about the timely delivery of relevant information such as location, severity etc. of relevant events and or transmitting data usually to an end user which is usually a person. Therefore data in these systems are designed around being human readable. Services style infrastructures usually employ existing commercial internet technologies and enterprise systems.

As the main focus of this thesis is the utility style infrastructures it is important to discuss the current communication systems available in these types of critical infrastructures. Industrial Control Systems (ICS) is the general term for the communication network found in Utility style CIs. These networks are used to collect data about the state of the infrastructure as well as take corrective actions when the system state strays from the optimal levels. ICS systems are classified either as Distributed Control Systems (DCS) or Supervisory Control And Data Acquisition (SCADA) systems. This classification is based on the size of the ICS and the primary purpose of the network. For instance SCADA networks are used primarily for monitoring geographically sparse infrastructures such as a
national transmission line or gas pipelines covering hundreds of kilometers. Whereas a DCS may be found in a manufacturing plant and be used primarily for process control. It is common for SCADA and DCS systems to be integrated to provide the ICS solution for most application especially in industries like power generation and distribution. Table 1 highlights the differences between SCADA and DCS.

Table 1 DCS versus SCADA comparison

<table>
<thead>
<tr>
<th></th>
<th>DCS</th>
<th>SCADA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Driven</td>
<td>Process Driven</td>
<td>Event Driven</td>
</tr>
<tr>
<td>Small geographic areas</td>
<td>Small geographic areas</td>
<td>Large geographic areas</td>
</tr>
<tr>
<td>Suited to large integrated systems such as chemical processing and electricity generation</td>
<td>Suited to multiple independent systems such as discrete manufacturing and utility distribution</td>
<td></td>
</tr>
<tr>
<td>Good data quality and media reliability</td>
<td>Good data quality and media reliability</td>
<td>Poor data quality and media reliability</td>
</tr>
<tr>
<td>Powerful, closed-loop control hardware</td>
<td>Powerful, closed-loop control hardware</td>
<td>Power efficient hardware, often focused on binary signal detection</td>
</tr>
</tbody>
</table>

ICS networks differ from commercial networks in a number of ways. First and foremost, ICS systems are found mainly in industrial domains where the machinery needs monitoring and or control. Although requirements differ from industry to industry, they may be broadly classified into discrete manufacturing, process control, building automation, utility distribution, transportation, and embedded systems. Secondly, the architecture of ICS is different from commercial networks in the depth of the network. These systems contain a more varied collection of protocols and are more hierarchical than their commercial counterparts. Figure 1, shows the typical structure of an ICS versus a commercial network.
ICS networks further differ from commercial networks in the severity of failures. ICS systems are designed to have higher availability and lower response time than commercial networks. This is because most control applications are very time critical and the difference between a stable state and an unstable one could be a few microseconds. Table 2 summarises the differences between commercial and industrial networks.
Table 2 Difference between Commercial networks and Industrial networks[15]

<table>
<thead>
<tr>
<th></th>
<th>Industrial</th>
<th>Commercial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primary Function</td>
<td>Control of physical equipment</td>
<td>Data processing and transfer</td>
</tr>
<tr>
<td>Application Domain</td>
<td>Manufacturing, processing and utility distribution</td>
<td>Corporate and home environments</td>
</tr>
<tr>
<td>Failure Severity</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Reliability Required</td>
<td>High</td>
<td>Moderate</td>
</tr>
<tr>
<td>Round Trip Times</td>
<td>250 µs – 10 ms</td>
<td>50+ ms</td>
</tr>
<tr>
<td>Determinism</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Temporal Consistency</td>
<td>Required</td>
<td>Not Required</td>
</tr>
<tr>
<td>Data Composition</td>
<td>Small packets of periodic and aperiodic traffic</td>
<td>Large, aperiodic packets</td>
</tr>
<tr>
<td>Hierarchy</td>
<td>Deep, functionally distinct hierarchies with many network protocols and</td>
<td>Shallow, integrated hierarchies with uniform protocol stacks and</td>
</tr>
<tr>
<td></td>
<td>physical standards</td>
<td>physical standards</td>
</tr>
<tr>
<td>Operating Environment</td>
<td>Hostile conditions, often featuring high levels of heat, dust and</td>
<td>Clean environments, often specifically intended for sensitive</td>
</tr>
<tr>
<td></td>
<td>vibration</td>
<td>equipment</td>
</tr>
</tbody>
</table>

It is important to note that not all systems fall within the parameters shown in Table 2, as there are applications within each network type that have atypical requirements. As an example the Advanced Metering Infrastructure (AMI) for smart grid can tolerate delays in excess of several seconds to even days as the data collected is not always time critical. Similarly some commercial network may not have large aperiodic data, but data may be predictable as in video conferencing applications.

Although ICS systems are well developed, the problem arises when trying to integrate multiple networks. The wide spectrum of network protocols and physical standards make interoperability difficult, sometimes even within the same organization. However with the proliferation of internet technologies, conventional protocols and technologies previously only applied in commercial products are now being found in ICS
systems as well. For example TCP/IP based technologies are now been used in industrial systems, and wireless technologies are becoming more prevalent in these systems.

Furthermore, several limitations of ICS networks including low bandwidth, low computing power, low memory, etc. are now becoming a thing of the past with the current increase in computing power and power efficiency of embedded devices.

However, not all CI systems are industrial in nature as highlighted earlier, and a network to support disseminating alerts and distribution of information will need to meet a different set of requirements. First the style of interaction in current ICS networks is one-to-one, because the system is either collecting data about a specific device or sending commands to change the state of a control device. Secondly, ICS networks do not use a uniform set of protocols, rather the communication protocol used varies depending on the application or the age of equipment [15]. Finally the data provided by these system are usually domain specific highly technical data, like the voltage reading of a transformer or the operating frequency etc. Data in this format may not be useable outside a specific domain. In contrast the goals of inter-CI communication is not to know the specifics but rather to obtain information about offered services and how those services are likely to change and where. Similarly, the communication requirements for emergency services can be summarized as knowing what event has occurred and where that event occurred. Therefore, integrating these different systems requires that information from multiple sources be delivered to multiple users either as is, or modified to suit the needs of the destination application.

2.2 RELATED WORKS

There have been a number of projects whose goals are aligned with providing a communication architecture for CI-CI communication, especially in the context of sharing information for the purpose of improving resiliency and the effects of cascade failures. In the next subsections, each of the projects is highlighted with its contributions.
2.2.1 Integrated Risk Reduction of Information-based Infrastructure Systems (IRRIIS)

IRRIIS is one of the earliest programs focused on improving the resilience of CI systems. It began in February 2006 under the 6th Framework Programme and ended in July 2009. It was focused around developing models and tools for analysing, simulating and managing dependent and interdependent CIs. It did this by introducing a language for describing risk as well as a set of middleware communication technologies to exchange this information among dependent CI systems. According to [16], the main goals for the IRRIIS were:

- Determining a sound set of public and private sector requirements based upon scenarios and related data analysis;
- Designing, developing, integrating and testing communication components suitable for preventing and limiting cascading effects as well as supporting recovery and service continuity in critical situations;
- Developing, integrating, and validating new and advanced modelling and simulation tools integrated into a simulation environment for experiments and exercises; and
- Validating the functions of the middleware communication components using the simulation environment and the results of the scenario and data analysis.

The Middleware Improved Technology (MIT), provides the communication technologies that supports the communication between CI systems of different types. MIT is the middleware backbone that allows the communication between multiple dissimilar CI systems. It consists of the MIT Communication Tool, Risk Estimator (RE); CRIsis management and Planning System (CRIPS) decision support tool; Tools for Extraction and functional status (TEFS); and the Incident Knowledge Analyser (IKA) [16]. The MIT Communication Tool represents the communication backbone of the IRRIIS model. It supports the exchange of information between dependent CIs using the risk management language (RML) via Web services. The MIT communication backbone is designed to use current internet based technology. It uses the TCP (Transmission Control Protocol) for guaranteed exchange of information.
IRRIIS is a multi-faceted programme and developed a number of outcomes including a federated simulator (SimCIP) for modeling interdependent CIs. To mitigate the problems of data format exchange and relevance across domains it employs the TEFs to extract the data from SCADA systems and uses the XML based RML as a format to exchange the extracted information. However from Figure 2, it can be seen that it did not directly use the raw data from the infrastructure, rather the Risk Estimator (RE) uses the input from the SimCIP simulator. Data from the infrastructure such as that from the SCADA system is transformed before it is used via the TEFs module.

Figure 2  Overview of the MIT Architecture
2.2.2 Tool for systemic risk analysis and secure mediation of data exchanged across linked CI information infrastructures (MICIE)

The European Commission MICIE FP\& ICT-SEC project (Tool for systemic risk analysis and secure mediation of data exchanged across linked CI information infrastructures)[17] , was part of the European Programme for Critical Infrastructure Protection (EPCIP) and was aimed at developing a Critical Infrastructure Warning Information Network (CIWIN) for European Union member states. The project aimed to improve CI resilience by providing real time risk level that measures the likelihood that a given CI would be unable to provide its services with the required QoS as a result undesired events in the reference CI and/or in its interdependent CIs. The MICIE system was designed to provide a method to discover distributed information relevant for the alerting system, overcome the disparity of this information from multiple CIs, and finally the means to exchange this information securely over the internet. The architecture is shown in [17].

![MICIE Architecture](image-url)
The MICIE project is by far the most similar in terms of goals with the current project, however the approach of the MICIE project was to have the secure mediation gateway (SGMW) act as the communication element between multiple CIs. The primary goals of the SMGW included: providing a secure cross-CI communication infrastructure; CIs critical event discovery and propagation of relevant information to trusted interdependent CIs; composing of CIs critical events and semantic inferences; and the extension of risk prediction from a single CIs to multiple interdependent CI [18]. SMGW achieves these goal using five components these are: Data/Metadata database, information Discovery framework, communication engine, SMGW manager, and the auditing engine.

![SMGW Architecture](image)

**Figure 4  SMGW Architecture [18]**

The MICIE project like the IRRIIS tries to enhance the resilience of CIs by integrating status information into analytical tools that help predict the future states of a CI. MICIE differs from IRRIIS in that instead of data from a simulator (SimCIP) it uses the raw data from the infrastructure[18]. The Adaptor module of the SMGW acts in a similar manner as the TEFS module in IRRIIS as it interfaces with the CI monitoring system. Furthermore the MICIE project has been tested on a real pilot system using the interdependencies between a telecommunication infrastructure and the electrical power infrastructure [17].
2.2.3 GridStat

All the projects considered so far have dealt with multiple CIs. The GridStat project is different in that it tries to address the limitations of current SCADA systems by making it possible for any entity within the network to receive the data from any other entity on the network irrespective of location. Its relevance here is that unlike the previous projects that do not consider the internal communication networks of the CI, GridStat deals exactly with this part of the network. To achieve this GridStat provides a reliable QoS managed middleware designed to meet the stringent tolerance imposed by power system monitoring and control. In the GridStat system there are two planes; these are the control and data plane. The control plane consists of hierarchically connected collection of QoS brokers. These brokers are responsible for controlling the functions of the data plane. In the data plane there are routers, publishers and subscribers. The routers are responsible for forwarding data from publishers to subscribers that have expressed interest in the data the publishers publish.

A distinguishing factor of GridStat is the concept of rate filtering. This refers to the ability of the system to provide the same data to different subscribers at different rates base on their subscription requirements. This means not every data published by a publisher is delivered to the subscribers just those that occurred at an interval specified by the subscribers. This ability of the routers to forward packets downstream only when a subscriber exists that requires the data helps to limit the amount of data routed through the network. Furthermore GridStat data plane connections between routers can be seen as data virtual paths capable of supporting QoS requirements. It uses the Common Object Request Broker Architecture (CORBA) protocol, a form of remote procedure call (RPC) for distributed object. This differs from the use of web services as in MICIE or IRRIIS, and provides lower latency.

Although GridStat provides a means for distributing power system data, it does not provide a means to connect multiple infrastructures of different types. Therefore in its current form it cannot support a multiple infrastructure types. However, it does pave the
way for a more inclusive communication architecture design for CIs that may be extended beyond the boundaries of similar infrastructure to dissimilar infrastructures.

2.2.4 Unified Incident Command and Decision Support (UICDS)

The UICDS project is also another project in the area of providing a system to coordinate multi-agency efforts in the response to a crisis. It is mention here because it address the requirements of emergency services that are closely tied to the utility style infrastructures. It is an initiative of the Department of Homeland Security in the United States of America. The UICDS system supports emergency services by providing relevant information about a potential threat or ongoing incident quickly to multiple agencies usually within a specific geographic area where such event is occurring [7]. UICDS uses an Agreement and Profile Services to extract information relevant to a specific group of first responders. Agreements represent relationships between UICDS organizations, and Profiles represent the relationships between UICDS applications. These relationships allow users to collect, analyze, and display information relevant to their decision process. UICDS has incorporated a number of standards to define and exchange information which include: National Information Exchange Model (NIEM); Common Alerting Protocol (CAP); EDXL-Distribution Element; EDXL-Resource Messaging; UCore Digest, Open Geospatial Consortium (OGC) Web Map Context; OGC Web Mapping Service; OGC Web Feature Service; OGC Sensor Observation Service; KML, GeoRSS, and Atom+GML; and Law Enforcement Information Technology Standards Council (LEITSC).

2.3 SUMMARY

Each of the projects considered so far have dealt with specific aspects of the CI-CI communication. The MICIE and IRRIIS project addresses the problem of CI-CI communication at the network boundary of CIs and do not address the communication issues within the networks. The assumption is made that data can be collected in a timely manner. Unfortunately this is not always the case. Furthermore they have considered communication between these CIs by using a communication entity (SMGW, MIT) that connects multiple CIs at the control or administrative level. In essence these systems can be considered as operating at the CI network boundaries. This is because they collect data
from SCADA systems, transform the data for their specific approaches and exchange this data between CIs. However, a disadvantage of this approach is that data granularity is lost. Also by transforming the data before sharing it becomes more difficult to integrate a different application in the future that requires a different form of data, this results in the need to add new adapters.

The works discussed so far bring to fore the complexity of CI-CI communication. However, there are important lessons to learn from these projects that provides insights into the nature and requirements of CI-CI communication. In the next chapter a unified communication architecture is presented. This system attempts to unify the needs of communication internally in a CI and externally between CIs. This approach allows interconnected CIs to seamlessly exchange data and have any component within or outside the CI receive relevant data by simply subscribing. This ensures that whether the application is a risk estimator, a monitoring system, or some new application, it can be supported by the network. In the next section the key concept of publish-subscribe is presented as it is a major part of the system design.
CHAPTER THREE

3 PUBLISH-SUBSCRIBE AND OVERLAY NETWORKS

Publish-subscribe communication paradigm lends itself naturally to the problem of information sharing in critical infrastructures as the goal is to provide information from multiple sources to multiple destinations. Publish-subscribe is an interaction pattern that defines the exchange of messages between publishing clients and subscribing clients [19]. Publish-subscribe architectures offer the ability to decouple publishers and subscribers in space, time and synchronization [20]. Publish-subscribe systems are usually implemented as overlay networks above the network layer and use the resources of the underlying network, therefore they can span across multiple kinds of physical networks.

In this type of system subscribers express their interest in receiving messages and publishers publish messages without a specific recipient for the messages. This is unlike the client-server interaction in which the sending client knows the destination of the message. This provides an anonymous and decoupled information exchange. Therefore a publish-subscribe system naturally supports a many-to-many style communication where data sources publish and data sinks subscribe. There are a number of publish-subscribe systems which differ in the way subscribers express interest in messages, the structure and format of those messages, the architecture of the system, and in the supported degrees of decoupling. Publish-Subscribe interactions can be found in middleware abstraction, enterprise application integration, system and network monitoring, and selective information dissemination.

There are many interpretations of the publish-subscribe operation. Therefore publish-subscribe may be interpreted as an asynchronous communication style, a messaging paradigm, a message routing approach, an event matching approach, or a design pattern depending on the domain of application[19].

As an asynchronous communication style, emphasizes is placed on data dissemination and the resulting qualities of service. This view of publish-subscribe system
is mainly concerned with efficient distribution of data from many sources to many destinations. In some cases it may take the form of data dissemination from a single publisher to many subscribers and therefore be similar to a multicast system.

The messaging paradigm interpretation of publish-subscribe system emphasizes the asynchronous and decoupled nature of the interaction between data sources and data sinks. This view of publish-subscribe can be found in products such as the Java Message Service Specification (JMS)[21]. These kinds of publish-subscribe although focuses mainly on asynchronous one-to-one communication via message queues, it may also include publication, subscription, and filtering. Filtering in these is more often implemented as subscriber-side filtering where the subscriber receives every message but discards the ones irrelevant to it.

The event filtering and matching approach emphasizes the selective filtering capabilities of publish-subscribe. In this approach subscriptions represent filtering expressions and publications represent observations about events in the environment that subscribing entities need to be selectively notified of.

Finally as a design pattern, publish-subscribe is based on the Observer design pattern [22]. The Observer pattern describes a technique for expressing one-to-many dependency between objects in a system. The object whose state is monitored is referred to as the subject, and the objects notified when there is a change in the state of the subject are called observers. This view of publish-subscribe violates the anonymity property of the typical publish-subscribe. This is because the implementation requires the subject to know its dependent observers and the observers to register if they are interested in a subject’s state.

3.1.1 Basic Architecture

A publish-subscribe system is made up of publishers, subscribers, and publish-subscribe message broker(s), also referred to as message router(s)[19].

The role of publishers and subscribers is typically performed by applications built with the publish-subscribe abstraction. Therefore a system client can be a publisher and a subscriber simultaneously. Publishers report on events by publishing messages to the
publish-subscribe system. Similarly subscribers express interest in messages by registering subscriptions with the publish-subscribe system. The publish-subscribe system then evaluates publications against registered subscriptions to determine matching subscriptions. This process of matching varies in complexity from channel based publish-subscribe to content-based publish-subscribe schemes. Publication once matched are usually not stored and are hence transient, except in state-based publish-subscribe systems and the Subject spaces model[23, 24].

The function of matching the publication to subscription is performed by the message brokers in the publish-subscribe system. There may be one or more message brokers depending on whether the system follows a centralized approach or a distributed approach.

Publish-Subscribe enables the decoupling of publishers and subscribers in three ways, these are: decoupling in space, decoupling in time, and decoupling in location. Decoupling in space means that clients can be physically distributed, for instance they may not be in the same network or continent. Decoupling in time means clients of a publish-subscribe system do not have to be available at the same time, subscribers and publishers can come and go. Decoupling in location means that clients of the system are unaware of each other’s identity, hence enabling anonymous communication. Although clients may be anonymous to one another they are not anonymous to the system in most cases, as they would have a system identification or address.

A publish-subscribe system is characterised by three interdependent models. They are the subscription language model, the publication data model, and the matching semantic. Together these models define how subscribers subscribe to messages, publishers advertise publications, messages are published, subscribers are notified of matching messages, and publications are matched.

The subscription language model defines how subscribers express interest in publications. It determines how expressive subscriptions can be for a given publish-subscribe system. For instance in a content-based model, subscriptions are defined as a Boolean function over Boolean predicates. These Predicates test for equality, binary
relations, or string operators over attribute values in publications. In some systems the
subscriber and the consumer are separate entities. These kinds of systems define the
subscriber as the entity that specifies subscription and the consumer as the entity that
receives notifications when certain subscriptions match a publication.

The structure, format, and the content type of publication messages are determined
by the publication data model. Publications are the messages generated by publishers and
represent events of interest about the state of the system or other monitored system or world.
These events define asynchronous state transitions of interest to subscribers, while the
publication is the message that conveys this occurrence to any interested subscribers.
Although no distinction is usually made between publications and events in practice, events
are state transitions and the publication is the message emitted as a result of the state
change. Often the concept of publication may be refined by the introduction of
notifications. A notification is a message sent by the publish-subscribe system to
subscribers with matching subscriptions, whereas the publication is the message published
by the publisher about an event. Systems that use notifications do so to enable subscribers
have a push or pull interaction with the publish-subscribe system. For instance they may
choose to receive the messages immediately or defer it to a later time if the processing
system is currently busy. However, this requires the publish-subscribe system store
matching publications. It is not required that a notification be identical to the publication
that triggered it. It is possible for a system to define notification semantics that specify
which values of publications to forward to subscribers. Furthermore notifications may
apply transformations to publications before notifying subscribers. In addition some
publish-subscribe systems may define advertisements. An advertisement is similar to a type
in a programming language, or schemas in databases and it specifies the kind of information
the publisher will publish.

Finally, matching semantic defines the conditions under which a publication
matches a subscription. This may be defined in a number of ways, for example in content-
based publish-subscribe, subscription are usually conjuncts of Boolean predicates. Hence
a publication matches the subscription if and only if it matches all predicates in the
conjunction. However, it is not necessary that the matching semantic define exact matches.
For instance the Approximate Toronto Publish-Subscribe (A-ToPSS) [25] project uses a model based on fuzzy set theory and possibility distribution which is an approximate matching semantic. Other forms of matching semantics includes: probabilistic matching semantics, and similarity-based semantics.

3.1.2 Types of Publish-Subscribe Model

There are a number of publish-subscribe models in use. These are channel-based, topic-based, content-based, type-based, state-based and subject spaces.

3.1.2.1 Channel-based Publish-Subscribe Model

In this model of publish-subscribe system, publishers publish messages to event channels and subscribers receive messages by listening to specific channels. This model of publish-subscribe does not perform matching in the form of matching publications to subscriptions. Therefore by listening to specific channel subscribers express their interest to receive all messages published to that channel. A rudimentary form of message-filtering may be achieved by the use of multiple channels to divide the publication space.

The channel separates the interaction between publishing data sources and subscribing data sinks. In this model of publish-subscribe system the data model is defined by the type of message supported by the channel-based communication abstraction. Also the subscription language model is defined by the programming language or library that lets application developers pick the channels on which to listen for messages. This type of publish-subscribe may support subscriber side filtering that may be comparable to content-based publish-subscribe models [26].

3.1.2.2 Topic-based Publish-Subscribe Model

In this model of publish-subscribe interaction, the data sources publish messages to specific topics and data sinks subscribe to receive messages of a given topic. Topic-based publish-subscribe may also be referred to as subject-based publish-subscribe. Topics may be known in advance to the clients of the system or be discoverable by clients. Topics are
a part of the message itself, and the publish-subscribe system is only able to interpret these topics and not the contents of the message.

A publication matches a subscription if the topic associated with the publication matches the subscription expression. In basic implementations topics are strings that represents a name, subject, or topic that classifies the messages in the publish-subscribe system. In more advanced forms, topics are drawn from a hierarchical topic space. Topic spaces allow subscribers to subscribe to parts of the topic hierarchy. For instance a system may define the following topics: “ABC/Turtle/DE5”, “ABC/Turtle/AD4”, and “DDD/Turtle/VBNM”. A subscriber on this system may subscribe to “ABC/*/AD4”, and all matching publication would be “ABC/Turtle/DE5” and “ABC/Turtle/AD4”. Similarly a subscriber subscribing to the topic “*/Turtle/*” will match all three topics.

The data model in topic-based publish-subscribe is defined by the topics that can be associated with a message. In simple models the data model allows application developers to organize messages by defining a simple collection of topics and more sophisticated systems may allow developers select topics from a hierarchical topic space.

The subscription language model is dependent on the publication data model. The subscribers in a flat publication data model express interest in receiving messages of a given topic by specifying the exact topic or by specifying a regular expression that matches a subset of the topic space. In a hierarchical model, subscribers may specify any part of the hierarchy using a wildcard notation.

Topic based publish-subscribe models differ in the qualities of service offered by the system to its clients, such as reliability, topic persistence, message ordering constraints, message delivery guarantees, and message delivery latencies constraints [27].

3.1.2.3 Content-based Publish-Subscribe Model

This is a publish-subscribe interaction model in which the content of message is used to make notification decisions. In these systems the publish-subscribe message brokers can interpret the header as well as the content of the message.
The data model in content-based systems is usually modelled as a set of attribute-value pairs. These attribute-value pairs may be implicitly typed or explicitly typed. In explicitly typed systems, a type information is associated with each attribute value pair. Implicitly typed systems obtain the type of the attribute-value pair from the operator specified by the subscription referencing the attribute-value pair [28]. Other data models used in content-based system includes eXtensible Markup Language (XML)[29], Resource Description Framework (RDF) [30] and strings.

The subscription language model is dependent on the data model. In an attribute-value pair system the subscriptions are represented as Boolean functions over predicates. A *predicate* is an attribute-operator-value triple that evaluates to a true or false. Other subscription language are XML path language (XPath) [31] for XML based systems, RDF Query Language (RQL) [32] for RDF based systems, regular expressions, and keywords. Matching has the traditional interpretation in this model. The distinguishing factor being that matching is done against the message content and not a simple string like topic-based systems.

Content-based systems differ in the publication data model employed, the subscription language model, the matching semantics, and the system architecture. In terms of architecture, the system may exhibit a centralized or distributed architecture. In centralized systems, the system clients (publishers and subscribers) connect to the same publish-subscribe system. In distributed systems the clients connect to one of many publish-subscribe systems interconnected in federation.

**3.1.2.4 Type-based Publish-Subscribe Model**

Type-based publish-subscribe is a high-level instance of the publish-subscribe interaction model. This variant of publish-subscribe aims to leverage the advantages of statically typed and object-oriented programming languages to make the integration and use of the publish-subscribe paradigm easier for application developers.

One of the weakness of traditional publish-subscribe engines is that the data model generally defines events as plain structures containing event properties, rather than objects,
and subscriptions are expressed using SQL-like grammar based on these properties. As a result they only support predefined event types which represent maps of property-value pairs. Type publish-subscribe attempts to bridge the gap between programming language type system and the publish-subscribe model. A type-based publish-subscribe provides type safety and encapsulation guarantees while maintaining routing and filtering mechanisms efficiency.

In a type-based publish-subscribe events are instances of the application-defined types. It eliminates the need for topics as in topic-based systems by replacing the topics with application-defined types. Although type based publish subscribe provides a more flexible system in terms of extensibility. It does introduce a number of complications such as object serialization semantic. One approach is to serialize the object in the form of text like XML as found in web services or JSON [33], with encryption to provide some form of security. However, languages like Java and .NET have in built object serialization capabilities that enable the serialization of objects over a network stream [34]. A similar functionality is available in the C++ via the boost libraries [35]. Furthermore there are programming language independent formats like protocol buffers a Google project described in [36]. It is beyond the scope of this thesis to propose a specific serialization mechanism. However [37], presents a benchmark for several serialization engines for java with the fastest times around the 6 ms mark.
CHAPTER FOUR

4 SYSTEM ARCHITECTURE

The proposed publish-subscribe middleware for CI communication (PSMCC) uses a content based publish-subscribe with a typed based system. It consists of four basic components: the brokers, information routers, publishers, and subscribers. The brokers and routers together make up the publish-subscribe middleware infrastructure, which is the message brokers. Together these two entities provide the system clients with the necessary functionalities to exchange information.

Figure 5 shows the basic system architecture. The Broker and the network of routers represent the message broker infrastructure. Similarly, the publisher and subscriber entities represent the system clients. A client may also be both a publisher and a subscriber as shown in the diagram.

The brokers represent the system controllers while the information routers perform the function of matching publications to subscriptions and forwarding matching publications
to subscribers. The system supports a simple advertisement message, used by the data sources, to register an intent to publish on the system via the broker. Similarly subscribers express their interest in specific publications using a subscription request message.

### 4.1.1 Broker

The broker is the network controller. It authenticates the clients of the systems and allocates network resources. The broker allocates paths and controls how the routers forward data from publishers to subscribers. Figure 6 shows the architecture of the broker module.

![Broker Module Architecture](image)

*Figure 6  Broker Module Architecture*

The broker stores all subscription information on the network. Consequently it consists of a subscription database. Based on the information stored in the subscription database it is able to allocate the resources on the network to meet the requirements of the system clients.

The publication database stores the publication information for all active publishers on the network. This includes their address and information about the type of data they publish and how they publish that data. For instance the publisher may publish events at
regular intervals or infrequently. An infrequent publisher only publishes events when there is a change in a particular process.

The broker also performs security functions. It authenticates and authorises the clients of the system to either publish or subscribe to information. It may also implement security policies that limits the parts or kinds of data that may be visible outside its domain. This is an important function in cases where the data needs to be anonymized due to privacy concerns.

Finally, the network information database stores information about the underlying network. This information is necessary to allow the broker optimize the use of network resources.

4.1.2 **Routers**

The routers represent application layer (overlay) routers. They are responsible for forwarding data from the publishers to the subscribers and for matching the published data to the subscriptions. The basic architecture of the router is shown below in Figure 7.

![Figure 7 Router Architecture](image-url)
The forwarding table stores the information about how to forward packets. An entry in the table is a destination-predicate pair. It is used by the matching engine to find the destination for incoming packets. Furthermore, the routing engine helps to optimize packet forwarding. It uses a minimum spanning tree algorithm routed at its node to find the optimal path to reach all matching subscribers. Thus, the routers implement a multicast for data, to minimize the bandwidth utilization.

![Figure 8: Network stack of router and clients](image)

The diagram of Figure 8, shows the network stack of the routers as well as that of the system clients. The connecting lines represent physical connections between the nodes which may be wired, wireless, or both, depending on the particular system. Essentially, the routers may have more than one physical link between them.

The grey box in Figure 8 represents the external network which may be a backbone network or even the internet. The user layer refers to the application that uses the system to publish or subscribe to information. In the model, the TCP/IP network stack has been chosen because of it is the standard used in practice in modern networks (industrial or commercial).

It is important to note that the routing function is assumed to be performed by an overlay network. Overly networks are logical networks implemented above a physical network usually just above the Transport layer of the OSI internet standard model. Examples include Pastry [38] and CHORD [39].
4.1.3 Publishers

The publishers are the clients of the system that generate events or data that is of interest to the subscribers. The publisher is only a role played by a client therefore a client may be both a publisher of data and a consumer of data. For example a monitoring application may subscribe to sensor information and generate alarms (new events) when it observes an undesirable pattern. The alarms it generates may then be subscribed to by a control system or even a system administrator that may then take appropriate action.

In practice, publishing devices may be any device from sensors to enterprise servers. A single device may also contain more than one publisher application, each application being associated with a particular data type. In the model applications are identified using a Global Unique IDentifier (GUID) and well as its parent device Internet Protocol (IP) address and a port number.

4.1.4 Subscribers

Subscribers represent applications or systems that are interested in receiving the information about a set of publications. They express this interest by sending a subscription request to the broker through a leaf router. The leaf router is the network router to which the subscriber are connected to. The leaf router is not a simple IP based router but is also capable of processing the content of the packets flowing through it as described in section 4.2.1. Similar to publishers, subscribers may be small devices or large enterprise servers.

4.2 NETWORK ARCHITECTURE

The proposed system is built on top of the internet technologies and provides an abstraction from the underlying physical network technologies. In CI systems as already discussed the communication technologies used varies from application to application even within the same organisation. There are a number of protocols in used for both wired and wireless communication technologies. However, by employing an overlay network, the system can offer a uniform interface to its clients without requiring a major change to existing communication infrastructure. Furthermore, the TCP/IP communication stack is already widely available in many applications in CI systems. Therefore, building an
abstraction on top of this layer seems a logical choice. In order to understand the system architecture from a networks perspective this section first presents the application layer datagram prototype used in the simulation model. Next each component’s (broker, routers etc.) behaviour is presented in detail.

4.2.1 Message Structure

<table>
<thead>
<tr>
<th>Bits</th>
<th>3</th>
<th>7</th>
<th>11</th>
<th>15</th>
<th>19</th>
<th>23</th>
<th>27</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Kind</td>
<td>MD</td>
<td>MC</td>
<td>TYPE</td>
<td>Address Length</td>
<td>Payload Length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Source Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Destination Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9 Unicast datagram frame

Figure 9, shows the datagram frame for the packet used in the simulation model. The packet consists of source address and destination address fields. For unicast messages this represents the sending client’s system address and the destination client’s address. Note the address is not an actual network address but an abstraction of one, for the purpose of modelling the behavior of the system. In practice the address is a tuple of the applications identifier, the IP address of its host and its listening port number. These three values ensure that it is unique within the context of the network. This means that a more than one application running on the same host will have different combinations such as [378998, 192.168.1.45, 4989] and [377877, 192.168.1.45, 4988].
A multicast datagram is shown in destination address vector holds the address to all destinations for the multicast packet. The address length field is only meaningful for multicast packets as it specifies the number of destination addresses for the datagram.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>3</th>
<th>7</th>
<th>11</th>
<th>15</th>
<th>19</th>
<th>23</th>
<th>27</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Kind</td>
<td>MD</td>
<td>MC</td>
<td>TYPE</td>
<td>Address Length</td>
<td>Payload Length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Source Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Destination Address 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Destination Address 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination Address N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 10  Multicast datagram frame*

The **Mode field** (MD) is used to model request-reply message interactions. The mode flag is set if the datagram is a reply and unset if the packet is a request. This allows the system to model client-server interactions for control messages. Control messages are used by the message brokers (brokers and routers) to change the behavior of the system. They may also be sent by clients to register intent to use the system. In practice this would use the TCP protocol since control messages are should be guaranteed delivery.

The **Kind field**, is used to specify if the message is a control or data message. It is set for data messages and unset for control messages.

Messages can either be unicast or multicast. The **Multicast flag** (MC) models this difference. This flag is set for multicast messages and unset for unicast messages.

The **Type field** is used to specify the type of control message. The model defines the following control messages:

- **Forward Packet**, this type of control message is sent by the broker to the routers under its control to modify their forwarding table. It is carries a *forwarding rule* payload. The forwarding rule consists of a subscription predicate and the next hop address.
- **Ping Packet**, this type of control message is sent by the system clients to locate the nearest router to connect to. This is how the system clients join the network.

- **Publication Packet**, this type of control message originates from publishing clients and carries the *publication definition*. The publication definition specifies the type of data published by a publisher, including how it intends to publish the data. This is what is encapsulated as the advertisement object introduced in section 3.5.3. This information is used by the brokers to match subscriptions against publications at subscription time.

- **Subscription Packet**, this type of packet carries the subscription information. It is used by the subscribing clients to register subscriptions on the network.

- **Query Packet**, this type of packet is used by the brokers to query other brokers for publications matching a local subscription. When a broker receives a subscription request from a local subscriber, it also queries other network for matching publications. This also carries subscription information.

![Hierarchical topology](image)

*Figure 11  Hierarchical topology*

The network architecture introduced may be connected either as a flat topology or hierarchically. In Figure 11, a hierarchical topology is shown. Each network labeled unit is
equivalent to that shown in Figure 5. This hierarchical arrangement allows more general rules to be set higher up the hierarchy. This allows the system to reflect geographic and administrative boundaries. In this arrangement when a subscription request is received by a leaf broker, it contacts its tier broker for matching publications in other units under the administration of the tier broker. The tier broker then acts as a trusted middle man between the leaf brokers. The leaf brokers determine the kind of data that subscribers from other units can access or subscribe to. For instance if the data contains sensitive parts, this may be removed before being sent to a subscriber outside the leaf broker’s domain. This can be achieved by the routers on the network.

In a flat topology, the brokers operate as peers. There is no tier broker to act as a middle man. Therefore when a leaf broker receives a subscription request for data outside its domain it contacts its peer leaf brokers for matching publications. Any peer with a matching publication replies to the leaf broker where the request originated and a path is created for matching publication from the source unit to the destination unit. Figure 12 shows a schematic of a flat topology.
The next section introduces in details the concept of publish-subscribe and how it is incorporated in the model.

The topologies introduced so far provides flexibility to the system. The hierarchical topology may be suitable in situations where a central controller is required with many sub controllers. In contrast peer-wise broker system may be suitable for simpler systems or at the top of a hierarchy where no central authority exists.

4.2.2 Start up and Service Discovery

![System client start up timing diagram]

An important part of any publish-subscribe system is how the system clients discover publications and how they connect to the network. There are many possible ways to achieve this in practice. One such way is to broadcast presence information on a local network on a predefined port. Clients supporting the required service would listen on this port and respond to incoming requests. This is by far the simplest method however it is rather inefficient if all the clients on the network do not support the required service. An alternative is to have a predefined multicast group and port through which clients can discover the services supported by network connected devices. This approach is used in the Simple Service Discovery Protocol (SSDP) a part of the Universal Plug and Play protocol.
Another technique for service discovery presented in RFC6763 is the Domain Name Service- Service Discovery (DNS-SD) that uses DNS packets to advertise services available on a network. The advantage of a DNS-SD is that it is not limited to a link local address.

The model does not specify a specific service discovery protocol as this would be implementation specific. Hence the assumption is made that each network device is connected directly to at least one information router on the local network. This router referred to as the leaf router represents the clients access point to the publish-subscribe infrastructure. When a publisher or subscriber connects to a local network a packet is sent to broadcast its presence and discover the leaf router. When a router supporting the service receives the packet it responds with its full address which in this case is global unique id (GUID) and IP address, and a connection port. Figure 13 shows the messages timing diagram for a system client at start up. After receiving information to connect to the publish-subscribe network. The next step differs slightly depending on whether the system client is a publisher or a subscriber.

Publishers after receiving a leaf router reply sends out an advertisement packet. This contains the information about what kind of data they will be publishing and how they

![Publisher timing diagram](image)
would publish the data. In this preliminary model the publisher can either publish in a periodic manner or sporadically.

The timing diagram for a publisher is shown in Figure 14. The publisher sends out a request that is forwarded to the broker. The broker responds granting the publisher permission to publish data on the network, after storing the details of the new publication. On receiving the broker’s reply the publisher can then start sending its data packets.

![Figure 15 Subscription Flow chart](image)

The process for a subscriber is more involved as it depends on whether the system has hierarchical or flat topology of brokers. Similar to a publisher after obtaining the application address of the leaf router, the subscriber sends a subscription request, via the router to the broker. The broker on receiving the request queries its publication database to find local publishers that are publishing data matching the new subscription. This includes
not just the data but how the data is published. For instance, say a subscription is received for the topic ‘*.*.water.level’ matching publications ‘xyz.abc.water.level’ and ‘xyz.rtc.water.level’. In addition, the subscription specifies an interval of a minimum of at least 200 ms between readings however say ‘xyz.abc.water.level’ only publishes every second. Then the subscription cannot be satisfied by this publication even though the topic matches. In contrast ‘xyc.rtc.water.level’ is published every 400 millisecond, therefore it would satisfy this subscription completely. If no local subscriptions are found or the subscriptions topic is not in the local domain. The search is expanded to other brokers. If matching publications are found a reply is sent to the subscribing and network paths setup to allow the matching publications reach the subscriber. This is summarised in Figure 15.

4.2.3 Matching Subscriptions

![Subscription Matching flow chart](image)

Figure 16 Subscription Matching flow chart

When a subscription request is made, the broker attempts to match the subscription
request with active publications, this process is in three parts. First it matches the subscription object type with the publication object type. Secondly it matches the subscription topic with the publication topic. Finally it confirms that the publication QoS can satisfy the subscription QoS requirements.

After a subscription is matched the broker sets up paths from the publishers leaf routers to the subscriber leaf router. This is done using a forward packet, which is one of the control packets.

4.3 **SYSTEM PUBLISH SUBSCRIBE MODEL**

4.3.1 **Data model**

In the proposed model, the data model implements a mixture of a hierarchical topic-based model and a type-based publish-subscribe model. This is to allow the use of topic based publish-subscribe model and type based publish-subscribe. In this case the topic is yet another property of an object, while preserving the extensibility provided by type based publish subscribe. For instance it becomes possible to have object types that define a specific data with topic that reflect the specific CI, such as Alert class with a topic such as ‘abc.genco.area1.disruption’ and ‘cvh.watercorp.zone1.pressure.fluctuation’.

As described in section 3.1.2.4, type-based publish subscribe uses programming language types for data model. The *BaseData* class represents an abstract base class from which messages are derived. The *TopicData* class encapsulates a data type that has a topic just like in topic-based publish subscribe paradigm. The *TimestampData* represents a data that has a timestamp.
In the simulation model, the *BaseData* is the abstract base class for all messages published by publishers. The application developer extends this class to define new class types. The *PeriodicData* class represents messages that are published at regular intervals. To show the flexibility of a using programming language types, a new type may be added to the system named *XmlData* type. The new class simply extends the *TimestampData* as it does not follow the topic-based paradigm. In the case of the *XmlData* the content of the XML document is evaluated during matching. This class may be used to represents an XML document or structure.

The model also defines an advertisement data structure that is used by publishers to register their data types with the broker, shown in Figure 18. This is encapsulated by the Advertisement object. The data structure consists of a specification field that defines the qualities of the data. The qualities here refer to additional information provided by the publishing client such as whether it intends to publish periodically or otherwise, packet size descriptions, publication interval if applicable etc. The sample data field is used by the publisher to provide a dummy object to the publish-subscribe system. This provides a means for the publish-subscribe system to acquire the type information for the messages that will be sent through system.
4.3.2 Subscription Language Model

The subscription model uses a template class and Predicates to subscribe to data. This leverages the flexibility of type based publish-subscribe. Since the data model are based on programming language types the subscription are based on selecting a class type to subscribe to. Next the subscription contains a number of predicates which are Boolean expressions against the public members (properties) of the selected data type.
The subscription model shown in Figure 19, has two properties. The first, requirements, provides information about how this subscriber intends to receive matching data. For instance it may want to receive every matching data in at a fixed rate. Furthermore the requirements field may also specify latency and delay requirements.

Secondly, the \textit{getPredicate()} method shown is an abstract method which allows the subscriber to define its own implementation. For instance the \textit{PeriodicSubscription} class is an example used during the simulation experiments. It embodies a subscription to a \textit{PeriodicData} (data published at regular intervals). Also it uses a \textit{PeriodicPredicate} as the predicate type. The \textit{PeriodicPredicate} is explained in more detail in the next section.

4.3.3 Matching/Filtering Events

The \textit{Predicate} base class defines the mechanisms for matching publications to subscription. Depending on the implementation matching may be stateful or stateless. Stateful matching refers to a matching where the current match depends on previous matches. For instance a predicate that tracks changes in a value requires the previous states of the value in question to compute the change in value. In other words whether the current message matches or not depends not just on its value but the value of the previous messages.
Stateless matching refers to matching in which the previous messages have no effect on whether the current message matches or not.

![Matching and Filtering Classes](image)

The *PeriodicSubscription* class follow the stateful matching paradigm, as only message matching a given time-series is delivered to the subscriber. The *MultiPredicate* is a compound predicate that is made up of two simple or compound predicates to build more complex predicates. For example consider the subscription defined by the statement: “topic=electric.*.*.consumer.aedgfihghj9.power, type=uoit::ants::PeriodicData, rate=100”. The **topic**, represents the data object topic to which this subscriber wants to subscribe. Furthermore it is represented as a wild card notation, consequently all messages whose topic match the wildcard will be a match for this subscription request. The **type** represents the data type which this subscriber is interested. The subscription type must have the property topic otherwise the subscription is not valid. This is because not all data types have a topic. For instance the *XmlData* type introduced in section 3.5.3 has no topic associated with it. Next, the **rate** defines how often the subscriber wants to receive data, in this case 100 data points per second. Although the subscription is shown here as text, it is only for illustration purposes and to make generating the objects during simulation simpler. A more robust example is shown below in Figure 21.
<?xml version="1.0" encoding="UTF-8"?><subscription>
  <requirement>
    <parameter id="interval">100ms</parameter>
  </requirement>
  <datatype>
    <classname>uoi.t.ants.PeriodicData</classname>
  </datatype>
  <predicate>
    <classname>uoi.t.ants.PeriodicPredicate</classname>
    <constraint value="electric.*.consumer.aedgfhghj9.power" property="topic"/>
  </predicate>
</subscription>

Figure 21  Simple subscription XML document
CHAPTER FIVE

5 SIMULATION MODEL

5.1 OVERVIEW

The PSSMC model was implemented in the OMNET++ simulator. OMNET++ is an object-oriented modular discrete event network simulation framework. It provides a generic architecture and hence has found application in various problem domains from modeling communication hardware to validating hardware architectures. In general it can be suitably applied to any system where discrete event approach is suitable and the interactions between entities can be modeled as message exchanges.

OMNET++ provides the tools for writing simulations. At the core of OMNET++ is the component architecture for simulation models. Models are composed from reusable components called modules. Modules can be connected to one another through ports called gates to form compound modules. Similarly compound modules can also be connected through their gates to build more complex modules. A complete simulation model in OMNET++ is called a Network. The framework allows for unlimited nesting. Modules communicate with one another by passing messages through their gates on predefined paths called connections or directly to their destination.

Figure 22 OMNET++ Component Architecture
Modeling in OMNET++ begins by writing simple modules. Simple modules are the active modules in OMNET++, and they are written in C++ using the simulation class library. Simple modules occupy the innermost level of nested modules, and they can be connected to form compound modules.

Figure 22 shows the component architecture of the OMNET++ simulation framework. The top-level module in OMNET++ is usually a Network which contains other modules (Simple and Compound). Compound modules may contain simple modules as well as other compound modules. The arrows in Figure 21, represent connections. Connections are represented in the OMNET++ framework as *channels*.

The Simulation model consist of a number of simple modules that models the publish-subscribe system. There are four primary modules, these are: *CloudBroker*, *InfoRouter*, *Publisher*, and *Subscriber*. Together the *CloudBroker* and *InfoRouter*, represents the message brokers of the publish-subscribe system. The Publisher module represents a publishing client and the subscriber module a subscribing client. Other simple modules present in the model are the *BurstyPublisher*, *EdgeRouter*, *CloudEdgeRouter*, *EdgeInfoRouter*, and the *TierBroker* simple modules. The previously mentioned modules are all specializations of the four primary modules.

In OMNET++ each module is defined in three separate files. These are a C++ header file, a C++ source file, and a NED file. The NED file is a network description file used by the OMNET++ platform.
5.2 SIMULATION MODULES

5.2.1 Simple Modules

Simple modules in the model are divided into five groups, these are the publishers, subscribers, brokers, routers and utility.

5.2.1.1 Publishers

There are two simple modules in this group and they represent a publisher that publishes data at fixed interval and a publisher that publish data at an irregular intervals. The regular interval publisher module is simply called the Publisher module and the irregular interval publisher is called a BurstyPublisher. Both models of the following parameters:

1. **PacketSize**: This is used to set the size of the data packet. It is defined as a integer that represents the byte length. The value can either be a constant such as “292B” or a distribution for example, uniform (512KiB, 1024KiB).

2. **Publication**: This is a string that defines the publication. In the current model this is a simple comma separate list that contains the type of publication e.g. Periodic or Aperiodic. It also consist of a topic e.g. “electric.BBB.zone1.breaker.aedgfhgfhj1.power”. Finally is defines a publication rate, that is packets per second that a publisher publishers. For a bursty publisher however this value is ignored.

3. **Interval**: This parameter is only valid for the BurstyPublisher. It is usually a random distribution with an upper and lower limit. The upper limit represents the maximum interval and the lower limit a lower interval.

5.2.1.2 Subscribers

Subscribers group consists of the Subscriber simple module. This module models a subscribing client system. The Subscriber simple module consists of a subscription parameter used to define the subscription for this subscriber. The Subscriber module is the
sink in the model and is where most of the statistics is collected during experiments. The statistics collected include the following:

1. **End to end delay**: this represents the end to end delay or latency of the data packets received by this subscriber.
2. **Service time**: this is the total delay accumulated by the data packet due to processing delay at intermediate nodes. This is the sum total of the processing delay at the routers in the path between the publisher of this packet and the subscriber.
3. **Queueing time**: this is the total time the data packet spent in queues before arriving at the subscriber.
4. **Delay time**: this is the total delay as a result of link length. It is equivalent to the propagation delay.
5. **Hop count**: this is the total number of nodes the data packet crossed before arriving at the subscriber.
6. **Data size**: this records the size of data packets arriving at the subscriber, this is important for experiments where the packet size is not fixed.
7. **Local publication**: this is used to record the total number of publications in matching this subscriber’s subscription within its home network.
8. **External publication**: It is used to record the total number of publications matching the subscriber’s subscription in other networks that is outside its home network.

### 5.2.1.3 Brokers

There are two broker simple modules, these are the *cloudbroker* module and the *tierbroker* modules. The *cloudbroker* module represents a broker that directly controls a number of routers directly and is responsible for subscriptions and publications within it domain. A *tierbroker* represents and administrative broker that acts as a liaison between several *cloudbrokers* in a hierarchical administrative topology.
5.2.1.4 Routers

The routers in the model include two simple modules, these are EdgeRouter and the InfoRouter. The EdgeRouter module represents the boundary router that connects a multiple networks. It acts as a gateway for each self-contained network. InfoRouters represents the internal routers of the network is the access point of publishers and subscribers in the model.

The statistic collected at the InfoRouter module includes the total number of data packets processed, the packet arrival rate, and the number of subscriptions matching the publishers attached to this InfoRouter. Similarly the EdgeInfoRouter collects similar statistics. Two other router modules defined in the model besides the two previously mentioned are the CloudEdgeRouter and EdgeInfoRouter modules. The CloudEdgeRouter is a subclass of the EdgeInfoRouter while the EdgeInfoRouter is subclass of the InfoRouter. The only additions these add to the base class are an extra gate for input and output this was important during the implementation of the EdgeRouterNode.

5.2.1.5 Utility Modules

The utility modules are simple modules used in the module that are not essentially a part of the model. These include the LineQueue, QueueProcessor, RequestQueue, DataChannel, and PortConverter modules.

The LineQueue module models an egress queue, in which packets are queued for transmission. In contrast the RequestQueue models an ingress queue, where packets wait to be processed. The QueueProcessor models the processing delay and together with the RequestQueue module models the processing delay and ingress queuing delay. The QueueProcessor has a service time parameter to define the how long it takes to process each packet, this in return affects the ingress queuing delay as packets queue until they are processed.

The DataChannel module extends the OMNET++ platform data rate channel and simply updates each packets propagation delay statistics. Next the PortConverter is a module used to connect the other submodules together and has no functional value. Finally
the WANNet module represents the Wide Area Network (WAN) and acts as the network that connects multiple networks.

5.2.2 Compound Modules and Nodes

The simulation library consist of a number of compound modules and nodes used to realise the publish-subscribe model.

5.2.2.1 CloudBrokerNode

This is the compound module that represents a Broker module. It consists of the number of interfaces and the CloudBroker simple module.

![CloudBrokerNode Schematic](image)

*Figure 23 CloudBrokerNode Schematic*

The schematic of this module is shown in Figure 23 above. It consists of two sets of interfaces the cldinterfaces represents it connection to the unit this broker controls. The waninterfaces connects the broker to the wide area network and to other brokers or tier broker as the case may be.
5.2.2.2 *EdgeRouterNode*

This module represents a router node that connects it network to the wide area network (WAN) and to other networks. The EdgRouterNode like the CloudBrokerNode consist of two sets of interfaces, the first connects it to the local network and the other connects it to the WAN. Figure 24 shows a schematic of the EdgeRouterNode.

![EdgeRouterNode Schematic](image)

*Figure 24  EdgeRouterNode Schematic*

5.2.2.3 *InfoRouterNode*

This module represents the router within the infrastructure communication network that is part of the publish-subscribe infrastructure. It is a basic model consisting of the
routing logic contained in the InfoRouter simple module and a number of interfaces connecting it to publishers and subscriber as well as to other routers.

### 5.2.2.4 PublisherNode

This module represents a publisher on the network. It consist of the Publisher simple module and a NetInterface simple module. The publisher node can either have

### 5.2.2.5 SubscriberNode

The subscriber node contains the subscriber simple module and a NetInterface module. This forms the subscriber client in the simulation model.

### 5.2.2.6 TierBrokerNode

The TierBrokerNode contains a tier broker simple module and a NetInterface simple module. The TierBrokerNode represents a broker that is not directly in control of a network but is part of the administrative hierarchy of brokers.

### 5.2.2.7 WanNetNode

This module represents the wide area network and is basically a router with characteristics like delay that reflects the latency experience in a wide area network.
The networks used in simulation are built by combining these compound modules to build networks.

Figure 25  Simple Network showing nodes connected to form a hierarchical network

The network in Figure 25 shows a simple network consisting of two units (cloud0 and cloud1). The items e0 and e1 are edge router nodes for the networks cloud0 and cloud1 respectively. There is one publisher in cloud0 and two subscribers.
5.3 OMNET++ Packet

The OMNET++ simulator provides a base class of packets, called the `cPacket` class, which is meant to be extended by the modeller to model the specifics of the message type. It has been extended as shown in Figure 26 as the AppPacket class. The packet class shown above is an object oriented version of the datagram introduced in section 4.2.1. The greyed out properties are not part of base packet model, they are only used to collect statistics for the packet.

![Simulation Model Packet Class Diagram](image)

*Figure 26 Simulation Model Packet Class Diagram*
5.4 **Routing Mechanism**

In routing packets from the source to destination, the built-in OMNET++ cTopology class was used to obtain the network topology, and a simple minimum spanning tree algorithm was employed to implement a multicast route from the source publishers to the destination subscribers. The algorithm builds on Dijkstra’s shortest path algorithm. When a router receives a data packet, it checks its forwarding rules to see what destinations the packet needs to be routed to. Next, it finds the neighbor that connects it with the destinations. If more than one neighbor connects it to the destination, it finds the one with the minimum cost in terms of path length and bandwidth.

![ForwardRule UML Diagram](image)

*Figure 27 ForwardRule UML Diagram*

Figure 27 shows the structure of the forward rule. The Predicate is the construct supplied by the subscriber that encapsulates its subscription. The priority property is used by the broker to define how this subscription is to be handled with respect to other subscriptions. Higher priority subscribers may have their packets delivered for instance.
CHAPTER SIX

6 SIMULATION RESULTS AND ANALYSIS

In simulating the behaviour of the network, a simple network was chosen with two units. Each unit consisted of five routers, a broker, and an equal number of publishers and subscribers. This was to model the case where every subscriber is also a publisher. The router topology was generated using the Boston university Representative Internet Topology generator (BRITE) [40]. The generation model chosen was the Waxman random topology model [41]. The Waxman topology model, models the growth of computer networks geographically. Nodes are uniformly distributed in the plane and edges are added according to a probability function which depends on the Euclidean distance between the nodes. This probability of an edge between two nodes \( a \) and \( b \) is given by,

\[
P(\{a,b\}) = \beta \exp \frac{-d(a,b)}{L\alpha}
\]

(1)

Where \( d(a,b) \) is the distance from node \( a \) to \( b \), \( L \) is the maximum distance between two nodes, and \( \alpha \) and \( \beta \) are parameters in the range \([0, 1]\). Large values of \( \beta \) results in graphs with a higher edge densities, while small values of \( \alpha \) increase the density of short edges relative to longer ones [41].

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Node Placement</th>
<th>( \beta )</th>
<th>( \alpha )</th>
<th>Growth Type</th>
<th>Max BW / Min BW (Mbps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5_routers_00</td>
<td>5</td>
<td>Random</td>
<td>0.15</td>
<td>0.1</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_01</td>
<td>5</td>
<td>Random</td>
<td>0.15</td>
<td>0.2</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_02</td>
<td>5</td>
<td>Random</td>
<td>0.15</td>
<td>0.3</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_03</td>
<td>5</td>
<td>Random</td>
<td>0.15</td>
<td>0.4</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_04</td>
<td>5</td>
<td>Random</td>
<td>0.15</td>
<td>0.5</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_05</td>
<td>5</td>
<td>Random</td>
<td>0.3</td>
<td>0.1</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_06</td>
<td>5</td>
<td>Random</td>
<td>0.3</td>
<td>0.2</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_07</td>
<td>5</td>
<td>Random</td>
<td>0.3</td>
<td>0.3</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_08</td>
<td>5</td>
<td>Random</td>
<td>0.3</td>
<td>0.4</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
<tr>
<td>5_routers_09</td>
<td>5</td>
<td>Random</td>
<td>0.3</td>
<td>0.5</td>
<td>Incremental</td>
<td>100/100</td>
</tr>
</tbody>
</table>
Table 3 shows the topologies generated using the BRITE tool. This topology was used as a template to generate the corresponding router connections in OMNET++ environment.

A number of parameters was collected during each simulation run, these includes:

- Latency or end to end delay.
- Propagation delay
- Queueing delay
- Subscriptions per router
- Ingress queue length
- Number of Local publications matching each subscription
- Number of publications outside the subscriber’s home network matching its subscription.
- Packet Inter-arrival time at each router.

The next section shows results collected in more detail and the simulation setup.

6.1 SIMULATION SETUP

The simulation network used consisted of two separate networks of five routers randomly chosen from the set in Table 4. Each network consisted of equal numbers of publishers and subscribers. The ratio of routers to clients was increased by a factor of two for each network. In addition there was more than one set of network with the same parameters and distinguished by a trailing ‘0’ or ‘1’, for instance “Net_05_10_05_0” and “Net_05_10_05_1” have the same parameters except the router networks differ. The summary of the different networks are as shown in Table 4. The network label shown in Table 4 is used in the figures that follow to identify the network in a plot.
Table 4 Test Network parameters

<table>
<thead>
<tr>
<th>Network Names</th>
<th>Network label</th>
<th>Number of routers</th>
<th>Number of Clients per group</th>
<th>Number of publishers per router</th>
<th>Number of subscribers per router</th>
<th>Clients per router</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net_05_10_05_0</td>
<td>N-01</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Net_05_10_10_0</td>
<td>N-02</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>Net_05_10_05_1</td>
<td>N-11</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Net_05_10_10_1</td>
<td>N-12</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>Net_05_20_05_0</td>
<td>N-03</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Net_05_20_10_0</td>
<td>N-04</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>Net_05_20_05_1</td>
<td>N-13</td>
<td>5</td>
<td>5</td>
<td>20</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>Net_05_20_10_1</td>
<td>N-14</td>
<td>5</td>
<td>10</td>
<td>20</td>
<td>20</td>
<td>80</td>
</tr>
<tr>
<td>Net_05_40_05_0</td>
<td>N-05</td>
<td>5</td>
<td>5</td>
<td>40</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>Net_05_40_10_0</td>
<td>N-06</td>
<td>5</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>160</td>
</tr>
<tr>
<td>Net_05_40_05_1</td>
<td>N-15</td>
<td>5</td>
<td>5</td>
<td>40</td>
<td>40</td>
<td>80</td>
</tr>
<tr>
<td>Net_05_40_10_1</td>
<td>N-16</td>
<td>5</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>160</td>
</tr>
</tbody>
</table>

6.2 NETWORK PERFORMANCE AND SUBSCRIPTION DENSITY

The first parameter considered during the simulation is the subscription density. For a publish-subscribe system this refers to the number of publishers a subscription matches. The higher the number of publications a subscription matches the higher the number of packets that needs to be routed to the subscriber. Therefore a higher density network will require significantly more network resources than a less dense network. This increasing number of subscription is shown in Figure 28, for the networks: Net_05_10_05_0, Net_05_20_05_0 and Net_05_40_05_0.
From Figure 27, it is easy to see how the number of subscriptions increase as a function of both the clients/router and as well as the subscription density. This property is explored as scalability factor to see how the network parameters, such as end-to-end delay, are affected by as the subscriptions increase.

![Total number of Subscriptions in the network](image)

*Figure 28 Total number of active subscriptions in the network: Series one-five client per group*

### 6.2.1 Packet Arrival Rate

The packet arrival rate shows how many packets the router have to process per second. The higher the number the greater the network load. Figure 28, below shows the average packet arrival rate. This is a function of the number of publishers per router as well as the subscription rate requested by the subscribers.
End to End delay represents the network latency. This is the time it takes for a packet to travel through the network from the publishers to the subscribers. In Figure 30, the maximum end to end delay for each network configuration is shown. These values are made up mostly of the propagation delay as the experiment was ran using a small packet.
size (1024 byte) and a small processing delay (10µs). This result is to provide a base for other experiments.

![Maximum End to End Delay](image)

**Figure 30  Maximum End to End delay by Network**

6.2.3 Processing Delay

In a network the processing delay refers to the amount of time required for a node (router) to process a packet and send it to the egress queue to be put on the line. In the light of a type based publish-subscribe system, this would typically be the amount of time to deserialize the packet, make a routing decision, serialize the packet, and send it to the output interface. In queuing theory this refers to the service time per element in the queue. If this time is larger than the average packet inter-arrival time then the ingress queue tends
to grow out of bounds and the network would experience congestion or packet loss in a case where the queue capacity is bounded.

Figure 31  Processing delay and its effect on End-to-End delay
In Figure 31, the mean end to end delay as measured at the subscribers is shown. Each coloured series is a plot of the individual mean value for each subscriber on the network; a total of 100 subscribers. At 450 microseconds the end to end delay becomes undesirable as it becomes greater than a few milliseconds. This is because the inter-arrival time has exceeded the service time greatly, hence the queue length grows rapidly. This experiment was performed with packets with a fixed size of 10KiB.

Figure 32 summarises the results in Figure 31. It shows how the end to end delay varies with the service time per packet. From the Figure as long as the service time per packet is less than the 400 microsecond threshold the end to end delay is less than one millisecond. However, there is a sharp rise between 400 and 450 microsecond service times, as the system fails and queue lengths grow, resulting in congestion. This behaviour can be explained as an increase in the nodal delay due to high queuing delays.
This scenario attempts to investigate the effect of a slow processor on the system performance. It shows that as the service time per packet increases the system performance degrades. However, in section 6.2.6 it is found that this may be improved albeit with some loss in data.

6.2.4 Queuing Delay

The queuing delay was another parameter collected during the experiments. It was important to see how this parameter grew with network scale. The first set of experiments was carried out in the case where there is no limit to the length of the queue that is with a system having infinite buffer size. A second set of experiments sets a limit on the queue length, when the queue length is full the oldest message in the queue is removed and the new message is added. This behaviour is to prioritize newer data over older data. However, there are other priority schemes that may be implemented but they have not be investigated.

Table 5 Average queuing for data packets versus the number of clients per router

<table>
<thead>
<tr>
<th>Clients/Router</th>
<th>Low (ms)</th>
<th>Medium (ms)</th>
<th>High (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Max</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.0473</td>
<td>0.0963</td>
<td>0.0911</td>
</tr>
<tr>
<td>40</td>
<td>0.1347</td>
<td>0.1429</td>
<td>0.1114</td>
</tr>
<tr>
<td>80</td>
<td>0.3922</td>
<td>0.3257</td>
<td>0.2624*</td>
</tr>
<tr>
<td>160</td>
<td>&gt; 1s</td>
<td>&gt; 1s</td>
<td>&gt; 1s</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.0024</td>
<td>0.0043</td>
<td>0.0049</td>
</tr>
<tr>
<td>40</td>
<td>0.0058</td>
<td>0.0091</td>
<td>0.0067</td>
</tr>
<tr>
<td>80</td>
<td>0.0250</td>
<td>0.0273</td>
<td>0.0209*</td>
</tr>
<tr>
<td>160</td>
<td>211.97*</td>
<td>&gt; 1s*</td>
<td>&gt; 1s</td>
</tr>
<tr>
<td><strong>Min</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>40</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>80</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>160</td>
<td>0.0000</td>
<td>0.0005</td>
<td>0.0054</td>
</tr>
</tbody>
</table>

* This value is inconsistent as the reported value was calculated from the data points available as not all the subscribers received the required packets at this point.

In Table 5, the average queuing time for data packets for each network is shown. At 160 clients per router the queuing time rises beyond a reasonable value, which is greater than one second. This is for a system with an unlimited queue length and is provided here for comparison. In a real system queue length is not infinite therefore a second set of experiment considering a limited queue size is shown in Figure 32 below.
In the image of Figure 33, there are missing data points this was because some of
the data was dropped when the queue becomes full, hence those subscribers did not receive
data. However, the maximum queuing time with a limited queue size for the packets that
did arrive at the subscriber is significantly smaller in comparison with that of an infinite
queue. This shows a trade-off of quality for performance, but this is not a total disadvantage
as with proper priority queue systems the data can be classified into different QoS groups.
These groups could then be guaranteed different QoS parameters according to the available
the network resources.

![Figure 33 Queueing time with a limited queue length versus infinite queue length in logarithmic time scale](image)

### 6.2.5 Data Inter-arrival time

The data inter-arrival time represents the interval of time that elapses between each
successive data packet received by the subscribers. This was an important parameter to
record as it shows whether the system can meet the QoS requirement imposed by
subscribers. In this case this is the minimum interval between each notification. As an example say a subscriber subscribes to a topic, “abc.efg.123.*” at a minimum interval of 100ms between data points, that is 10 notifications per second. If the number of clients being served are few does the system meet this requirement, and how does the performance degrade as the number of clients served increases? This parameter attempts to measure this drift from the requested subscription rate. Table 6 shows as a percentage of total subscribers the number of subscribers whose data inter-arrival time falls outside the requested interval by one percent and five percent.

Table 6 Data Inter-arrival rate performance

<table>
<thead>
<tr>
<th>Clients/Router</th>
<th>Low (%)</th>
<th>Medium (%)</th>
<th>Heavy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Less than 1 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>75.00</td>
<td>56.00</td>
<td>31.00</td>
</tr>
<tr>
<td>40</td>
<td>40.50</td>
<td>27.50</td>
<td>13.00</td>
</tr>
<tr>
<td>80</td>
<td>28.00</td>
<td>21.75</td>
<td>1.50</td>
</tr>
<tr>
<td>160</td>
<td>18.75</td>
<td>22.00</td>
<td>2.00</td>
</tr>
<tr>
<td>Less than 5 %</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>75.00</td>
<td>56.00</td>
<td>31.00</td>
</tr>
<tr>
<td>40</td>
<td>41.00</td>
<td>29.00</td>
<td>13.00</td>
</tr>
<tr>
<td>80</td>
<td>28.25</td>
<td>22.00</td>
<td>17.00</td>
</tr>
<tr>
<td>160</td>
<td>18.75</td>
<td>22.00</td>
<td>2.00</td>
</tr>
</tbody>
</table>

![Figure 34 Percentage of Subscriptions satisfied within 1%](image-url)

68
From Table 6, the performance degrade as the number of subscribers increase. In Figure 34, these values are plotted against the total subscriptions at each router. From Figure 34 it is clear that the performance of the system does fall off as the number subscriptions increase.

6.2.6 Queue Size

So far the results presented are for scenarios where the only restriction has been the number of routers. However, the routers have been assumed to have unlimited processing power and an unlimited buffer size, but this is not the case in practice. Hence, to more closely reflect a real system two sets of scenarios was carried out:

- First, the service time of the routers was increased in steps from 10 microseconds to 1000 microseconds.
- Finally, the previous experiment was carried out this time with a fixed queue size of 20 packets, and the queue was set to favour newer packets over older packets.

The parameter collected was the maximum end to end delay at the subscribers and the data inter-arrival time, to compare what happens as the service time grows and with a limited queue size.

Figure 35 shows this comparison for a service time of 10 microseconds per packet. The plot shows the data inter arrival time requested by the subscribers overlaid by the mean data inter-arrival time recorded. It is clear that under both conditions the performance is unaffected as the maximum queue size of 20 packets is not reached if the packets are processed quickly enough.

Figure 36 shows the same result this time at 200µs. A performance difference is becoming apparent at this point. Finally in Figure 37 and Figure 38, there is a marked difference between the limited and unlimited queue size scenarios. This difference arises as the queue capacity is reached the system starts dropping less important packets. In this scenario newer packets are being favoured over older packets. Hence it is clear that by limiting the buffer size the system actually performs better albeit with some data loss.
Figure 35 Data Inter-arrival rate comparison at 10 µs service time

Figure 36 Data Inter-arrival rate comparison at 200µs service time
Figure 37  Data inter-arrival time comparison 500µs

Figure 38  Data inter-arrival time comparison 1 ms
Next the end to end delay on the system is considered under the same conditions to see the impact on the system performance. In Figure 39, the end to end delay for the limited queue size and unlimited queue size at 10 µs service time is shown. At the service time there is no marked difference between the limited queue size scenario and the unlimited queue size scenario.
The data shows a bimodal property because the end to end delay is dependent on the number of hops from source to destination. Since some of the subscriptions are for data outside the home network these packets travel through a larger number of nodes hence their much greater end to end delay. This relationship is expressed in equation (2).

\[ d_{total} = N (d_{proc} + d_{queue} + d_{trans}) + d_{prop} \]  

(2)

The term \( d_{total} \), represents the total end to end delay. The term \( N \) represents the total number of Nodes between the source and destination. The terms \( d_{proc} \), \( d_{queue} \), and \( d_{trans} \) represents average nodal processing delay, average nodal queueing delay, and average nodal transmission delay. Their sum represents the average nodal delay. This value has much larger effect the higher the number of nodes the packet transverses on its way to the destination. Finally the term \( d_{prop} \), represents the total propagation delay. It is determined by the characteristics of the physical links between the source and destination. Generally it is determined by the length of the link.

![Comparison at 200us service time](image)

*Figure 41 End to End delay comparison at 200 µs service time between limited and unlimited queue scenarios*

In Figure 40, the end to end delay for both scenarios is greater as the service time per packet is fivefold higher at 50 microseconds. However, there is no clear performance difference at this point. The maximum end to end delay at the point is just under the 1 millisecond as the delay is predominantly propagation delay (delay due to link length).
At 200 microseconds some difference is begin to arise between the limited and unlimited queue size scenarios. With the limited queue size performing better, as the queuing delay is begin to affect the performance of the unlimited queue scenario as shown in Figure 42.

![Comparison Maximum Queuing Time at 200us service time](image)

*Figure 42 Comparison of maximum queuing time between limited and unlimited queue at 200μs*

Finally, at 500μs and 1ms the end to end delay has become excessive with the unlimited queue and the maximum queuing time would tend to infinite at this point. Consequently it would be more practical to have limited queue size and sacrifice some data loss for operability. However, it would be important to investigate some other forms of priority queue depending on the application and what QoS requirements is imposed on the system by active subscribers.
Figure 43  End to End delay comparison at 500us between limited and unlimited queue scenarios

Figure 44  End to End delay comparison between limited and unlimited queue comparison at 1 ms service time per packet
CHAPTER SEVEN

7 CONCLUSION AND FUTURE WORKS

This thesis presented a communication architecture based on the publish-subscribe communication paradigm for the exchange of information between heterogeneous CI systems. The proposed architecture was built using the OMNET++ simulator for the purpose of analyzing the network performance of the model. The network performance study showed a strong correlation between network performance and the number of clients serviced by each router. Furthermore, the network performance degraded heavily when the service time per packet was higher than 200 µs. There was also a marginal improvement when the queue capacity was limited and the queue optimized to favour newer packets over old ones.

The analysis of the proposed architecture shows a lot of promise, however it would benefit from a number of improvements, such as the following:

- **Data Prioritization**: Although the scenarios presented in section 6.2.6 uses data prioritization to selectively drop older packets, this is not the only approach to prioritize data. An important case is to have an expiration time, such that the packet is only relevant within a given time frame. One application of this form of optimization is monitoring data. In this case the most current information has priority over older data. Also, information such as alerts may selectively have higher priority than, say, a simple notification. Hence, the priority bit available in the message header of Figure 8 and Figure 9, may be used for this purpose in future iterations. If implemented over an IP based network, the IPv6 differentiated services of RFC2474 may be incorporated to achieve this behaviour.

- **Reliability**: Another aspect of the design not fully considered is the reliability. Since the system is supposed to provide a backbone for the communication between CIs, it is important to investigate ways to ensure the system itself is...
resilient to failures and available. One important piece in the system architecture is the broker who acts as the network controller. If it becomes offline, new subscriptions may not be processed and publications would fail. Also, at the router level, what happens when a router goes offline within a network? These are areas to investigate and implement a recovery mechanism.

- **Security**: Furthermore, besides reliability, there is the question of security. Security concerns, in this case, include the authorization of system clients, confidentiality, privacy, and authentication. There are a lot of existing systems to provide authorization and authentication like the Kerberos server [42]. The confidentiality of data refers to being able to restrict certain information to only authorized users. In this case subscription for certain information outside an infrastructure may be denied. This is separate from privacy. Furthermore, to implement privacy certain aspects of the data can be removed before delivering it to subscribers that is the data is anonymized. This is one of the advantages of publish-subscribe or message notification, that is the separation of published data and notifications. Two users subscribing to the same data do not necessarily receive the same data structure.

- **Quality of Service (QoS)**: The current system only provides one parameter for QoS namely the data inter-arrival rate. However, other QoS guarantees could be incorporated to improve the number of options available to subscribers, and consequently provide more parameters to optimize.

- **Validation**: It is often not enough to simulate the behaviour of a system as real systems have behaviours that are difficult to simulate. To validate the model, it would be important to implement a prototype and have a pilot test on a real network to see how the system reacts in the real world.
8 REFERENCES


