Graph-based Modeling and Simulation of Emergency Services Communication Systems

Jardi Martinez Jordan

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2024

Committee:
Michael Stiber
Munehiro Fukuda
Johnny Lin

Program Authorized to Offer Degree:
Computing and Software Systems

University of Washington

Abstract

Graph-based Modeling and Simulation of
Emergency Services Communication Systems

Jardi Martinez Jordan

Chair of the Supervisory Committee:
Michael Stiber
Computing and Software Systems

Emergency Services Communication Systems (ESCS) are evolving into Internet Protocol (IP)-based communication networks, promising enhancements to their function, availability, and resilience. This increase in complexity and cyber-attack surface demands a better understanding of these systems' breakdown dynamics under extreme circumstances. Existing ESCS research largely overlooks simulation and the little work that exists focuses primarily on specific cybersecurity threats and neglects critical factors such as the non-stationarity of call arrivals. This paper introduces a robust, adaptable graph-based simulation framework and essential mathematical models for ESCS simulation. The framework uses a graph representation of ESCS networks where each vertex is a communicating finite-state machine that exchanges messages along edges and whose behavior is governed by a discrete event queuing model. Call arrival burstiness and its connection to emergency incidents are modeled through a cluster point process. The model applicability is demonstrated through simulations of the Seattle Police Department ESCS. Ongoing work is developing GPU implementations of these models and exploring the use of simulations in cybersecurity tabletop exercises.

TABLE OF CONTENTS

LIST OF FIGURES

GLOSSARY

ALI: Automatic Location Information. System used to identify the physical location of a caller in real-time.

ANI: Automatic Number Identification. Technology use to automatically determine a call's originating phone number.

CALLER REGION: In this thesis, a caller region represents a geographical area where emergency calls originate.

CAD: Computed-aided Dispatch. System that assists first responders in efficiently providing emergency services.

CFSM: Communicating Finite-state Machine. Finite State Machine (FSM) with operations to receive and send messages.

DDOS: Distributed Denial of Service. Cyber-attack intended to disrupt services provided by a host within a network.

EENA: European Emergency Number Association. European homologue of the National Emergency Number Association (NENA).

EMS: Emergency Medical Services. Emergency services that provide on-site treatment for medical emergencies and transportation to emergency medical facilities.

ESCS: Emergency Services Communication System.

ESN: Emergency Service Number. Code assigned to each line of the MSAG.

FCC: Federal Communications Commission. Federal entity that regulates interstate and international communications by radio, television, wire, satellite, and cable.

FIRST RESPONDER: An emergency responder. This could be a police unit, fire truck, or emergency vehicle.

I3: The i3 standard defines protocols, interfaces and systems that underlie the NG911 architectural framework.

ICERT: Industry Council for Emergency Response Technologies. Trade association dedicated to advocating for providers of commercial public safety response technology, software developers, and affiliated entities.

LOST: Location-to-Service Translation. Protocol that provides the operations for mapping locations to service Uniform Resource Indentifiers (URIs).

LTD: Long Term Definition framework architecture. The architectural framework developed by EENA for the NG112 system.

MSAG: Master Street Address Guide. Database of physical street names, address names, emergency services and other codes necessary for routing emergency calls within the 911 system.

NENA: The National Emergency Number Association. The aim of NENA is to promote the technological progress, accessibility, and adoption of a nationwide emergency telephone number system in the United States.

NFPA: National Fire Protection Association. Non-profit organization with a focus on fire safety.

NG911: Next Generation 911. The modern IP-based 911 system architecture.

NG112: Next Generation 112. European homologue of NG911 architecture.

FSM: Finite-state Machine. Abstract machine that can be in exactly one of a finite number of states at any given time.

GPS: Global Positioning System. Satellite-based radio system that provides users with positioning, navigation, and timing services.

ILECS: Incumbent Local Exchange (telephony) Carriers. Telephone companies that provided local services when the Telecommunications Act of 1996 was enacted.

OP: Operations Management. Processes seeking to optimize the efficient use of resources to produce good and services that meet customer requirements.

PSAP: Public Safety Answering Point. Call center responsible for answering emergency calls for police, firefighting, and ambulance services.

PTSN: Public Switched Telephone Network. Originally a network of fixed-line analog telephone system, the PTSN is composed of all the world's telephone networks.

SIP: Session Initiation Protocol. Protocol used for starting, maintaining, and terminating communication sessions.

SRDB: Selective Routing Database. The database containing the links between telephone numbers and ESNs used to route 911 calls.

TLS: Transport Layer Security. Cryptographic protocol that provides communications security over a computer network.

VOIP: Voice over Internet Protocol (IP). Technology used to provide voice calls using an internet connection instead of the Public Switched Telephone Network (PTSN).

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my chair, Dr. Michael Stiber, for his patience, motivation, enthusiasm, and invaluable knowledge. His continuous support and guidance helped shape the direction of this research and my academic growth. I would also like to extend my appreciation to my committee members, Dr. Munehiro Fukuda and Dr. Jonnhy Lin.

I thank my fellow members of the Intelligent Lab Network for their contribution to Graphitti and this research. In particular, Bennet Ye and Jacob White who developed tools for ESCS graph visualization and editing, and Divya Kamath and Xiang Li who made substantial architectural improvements to Graphitti.

Lastly, I extend my appreciation to Scott Sotobeer for providing invaluable insights into the working of ESCSes and their associated policies.

DEDICATION

To my dear wife, Juni.

Without her unconditional support, this thesis would still be in the realm of dreams.

Chapter 1

INTRODUCTION

An Emergency Services Communication System (ESCS) encompasses the collection of organizational, electronic, and virtual elements designed for answering emergency calls and coordinating responses. Consequently, ESCS systems serve as the critical link that connects individuals with essential first responders during emergencies. Despite their pivotal role in disaster preparedness, these systems can exhibit vulnerabilities under such extreme circumstances [10, 13, 45]. As such, it is essential to understand when and how their proper functioning starts to collapse.

ESCSes are undergoing significant infrastructure updates. In the United States, the Next Generation 911 (NG911) initiative is converting the Enhanced 911 (E911) system into an Internet Protocol (IP)-based system [34]. This IP-based ESCS infrastructure is also being implemented in Ecuador [8], Europe [27] and Asia [30]. The updated infrastructure promises enhancements to the functions, availability, and resilience of these systems by improving the inter-connectivity between Public Safety Answering Points (PSAPs) and emergency responders; enhancing call localization; and allowing ESCSes to receive text and multimedia messages on top of traditional voice calls. However, to deliver on those promises one must understand and mitigate the significant threat presented by cyberattacks, such as Distributed Denial of Service (DDoS), to the availability of the services provided by ESCSes [2, 32, 54].

PSAPs tend to be the main focus of modeling and simulation research because of their critical role in connecting the public with emergency responders. Since PSAPs are emergency call centers, analytical models such as Erlang equations are often used to model their operations. For instance, in the US, the National Emergency Number Association (NENA)

incorporates the Extended Erlang B equation into their recommendations for PSAP staffing policies [39]. However, the simplicity of analytical models requires that one makes significant and often unrealistic simplifying assumptions; such models can be unsuitable for complex systems, such as ESCSes. Computer simulations offer a practical solution for the analysis of complex systems, offering various advantages, including enhanced model reusability, simplified model creation, reduced need for oversimplification, and easy adaptability [53]. They also force investigators' mental models and assumptions to be made precise and explicit.

This thesis introduces an adaptable graph-based simulation framework and mathematical models that provide a comprehensive solution for ESCS simulation. This includes abstract models for the three distinctive aspects of an ESCS: (1) network structure, (2) internal behavior of its main components, and (3) interactions among these components. The versatility of this system facilitates the creation of ESCS simulations for various purposes, including day-to-day operational management, analysis and planning for crises or catastrophic events, and evaluating vulnerabilities to cyber-attacks.

We designed our simulation framework by re-architecting Braingrid, a specialized high-performance neural network simulator [47], into a general-purpose simulator for graph-based systems known as Graphitti [31]. ESCS components are depicted as a graph-based structure supporting Communicating Finite State Machines (CFSMs). Each vertex in this graph represents an FSM operating within a discrete event queuing model; whereas each edge denotes a communication link. Currently a sequential implementation, the architecture is designed to facilitate a future GPU implementation.

1.1 Contributions

The framework and models introduced in this study provide a comprehensive solution for simulating ESCSes, harnessing the advantages offered by computer simulations. Rather than a one-off simulation, we have developed a first-of-its kind, generalized framework for simulating graph-structured systems and applied it to ESCS operations. This framework uses configuration files to provide flexibility for simulating heterogeneous ESCS networks and to

adapt diverse mathematical models for stochastic processes. ESCS graphs are specified using GraphML, a standardized graph representation language [50]; whereas call arrival and other stochastic processes are specified using Extensible Markup Language (XML).

Modeling call arrivals is especially complex due to its multifaceted nature, which includes inter-day, intra-day, and seasonal variability [21]. A novel contribution of this thesis is the modeling of call arrivals as the realization of a cluster point process, capitalizing on the connection between emergency calls and the underlying emergency incidents. Moreover, this work delves into aspects of ESCSes frequently overlooked by other models, such as call abandonment, redial, and the emergency personnel dispatch process.

Lastly, to enable the simulation of large and complex ESCSes, the models were designed for a highly parallelized GPU implementation, although the current implementation was carried out on a serial CPU architecture.

1.2 Overview

Chapter 2 provides an in-depth exploration of ESCS, tracing their historical development alongside technological advancements and offering insights into the 911 response process. This chapter lays the foundational context necessary for understanding the subsequent chapters. Chapter 3, on the other hand, presents a thorough literature review, examining the landscape of ESCS modeling and simulation.

The methods are outlined in two separate chapters. In Chapter 4, we delve into the abstract models employed to capture different facets of an ESCS, the principal architectural decisions guiding the simulation framework, and their practical implementations. Meanwhile, Chapter 5 offers an examination of the mathematical models serving as the foundation for realizing the various stochastic processes integral to simulating ESCS.

Chapter 6 offers a comprehensive evaluation of the mathematical models' implementation and their ability to capture various aspects of an ESCS through the use of a demonstrative application. Lastly, in Chapter 7, we summarize the key findings from this study and provide a glimpse into potential avenues for future research.

Chapter 2

BACKGROUND: EMERGENCY SERVICES COMMUNICATION SYSTEMS (ESCSes)

An Emergency Services Communication System (ESCS) encompasses the collective organizational, electronic, and virtual components involved in the reception of emergency calls and the coordination of responses to emergency incidents, complemented by the relevant policies and procedures governing their operation. Consequently, such a system plays a pivotal role in enabling efficient communication during emergencies, ensuring prompt response and assistance. For example, in the United States and Canada, the emergency hotline number 911 is used to connect individuals in distress with the appropriate responders to handle critical situations. Similarly, the European Union introduced 112 as the single European emergency number to allow access to emergency services through a common number everywhere in the European Union, becoming available to all European Union member states by 2009 [29].

An ESCS can be thought of as two networks. First, a reliable telecommunication network serves as the vital link between the public and emergency services [30]. Then, a public safety network provides the wireless communication network used by emergency services organizations, such as police, fire, and emergency medical services [15].

ESCSes have evolved alongside technology advancements. For instance, the 911 system began using older circuit-switch telephony technology, with subsequent incorporations of newer technologies as stopgap measures [3]. As demands and expectations grew, these systems became limited, necessitating comprehensive overhauls and the implementation of more advanced communication infrastructure. Ongoing efforts such as Next Generation 911 (NG911) in the US [3], and similar initiatives in other countries like Ecuador [8] and the Next

Generation 112 (NG112) in the European Union [30] [27], continue to drive these efforts.

2.1 History of 911: Keeping up With Technology

The importance of communication during disasters and emergencies has been recognized since the early 1900s. The development of new telecommunication technology showed promise but often lacked proper planning and preparation, which led to numerous failures, as described by Farnham [16]. In the United States, before 911 became the designated emergency number, people experiencing emergencies would dial "0" to reach an operator, who often didn't have the tools or training to assist with emergency calls [22].

In their report on the history of 911, the Industry Council for Emergency Response Technologies (ICERT) outlines several critical developmental stages in the evolution of the 911 emergency system. These stages include Basic 911, Enhanced 911, Wireless Enhanced 911, VoIP Enhanced 911, and Next Generation 911 [22].

In 1957, the International Association of Fire Chiefs began advocating for a single telephone number dedicated to reporting fires. However, it was not until 1968 that AT&T, the primary telephone service provider in most of the US, officially designated 911 as the emergency number. Senator Rankin Fite completed the first 911-based emergency call in Halleyville, Alabama. During this period, AT&T emerged as the most suitable entity to provide 911 services. As a free market approach was not yet feasible, state legislatures and regulators established a regulated monopoly by entering into arrangements with incumbent local exchange (telephony) carriers (ILECs). This marked the inception of the Basic 911 service. Subsequently, the Public Safety Act of 1999 presented the official designation of 911 as the national emergency calling number [42].

Using the same wireline, analog, circuit-switched technology that was used in public switched telephone networks (PTSN), the infrastructure of the Basic 911 system established a connection between a central telephony office with a single public safety answering point (PSAP). However, the responsibility for handling and regulating emergency call services fall under the jurisdiction of state governments thus classifying 911 calls as a local service.

The initial implementation of the Basic 911 in the 1970s faced hurdles. As reported by Dayharsh et al. [12] in 1979, only 25% of political jurisdictions had a functional 911 system. This sluggish adoption was attributed to challenges in planning, high costs, limited interest from telephone companies, fragmented federal funding, and a lack of public awareness regarding the 911 service [12].

In the mid-1970s, the first significant improvement to the 911 system emerged in the form of Automatic Location Information (ALI) and Automatic Number Identification (ANI). This enhancement granted automatic access to critical information such as the caller's name, address, and phone number. This marked the transition to the Enhanced 911 (E911) system, which introduced various advances, including the capability to route calls to the appropriate Public Safety Answering Point (PSAP) after going through a Public Switched Telephone Network (PTSN) and a specialized selective router. Although this solution proved effective for landlines, it posed challenges for wireless telephone services.

As wireless telephone services gained widespread adoption in the early 1990s, the Federal Communications Commission (FCC) stepped in, mandating the issuance of the Wireless Enhanced 911 Rules. These regulations required wireless carriers to provide the location of 911 callers. By 2001, providers were mandated to deliver the latitude and longitude of callers but were given the freedom to choose the location technology, whether through a global positioning system (GPS) or network-based methods like cell-tower triangulation.

The advent of Voice-over-IP (VoIP) technology posed a fresh challenge for E911. The nomadic character of Internet-based VoIP services made pinpointing a caller's precise location challenging, resulting in many VoIP providers lacking 911 capabilities. Additionally, these providers often failed to inform their customers about the absence of 911 service. In response, the Federal Communications Commission (FCC) intervened by imposing 911 obligations on VoIP providers.

Furthermore, the New and Emerging Technologies 911 Improvement Act of 2008 granted VoIP service providers access to network elements and other capabilities that could seamlessly integrate with the E911 system.

2.2 Next Generation 911

In the early 2000s, it became evident that the reliance on reactive adjustments to the 911 system, prompted by emerging technologies, was no longer sustainable. The 911 system needed a redesign to overcome the limitations of its circuit-based technology and to accommodate newer services such as text, video, social networking, and web-based calling [3].

Next Generation 911 (NG911) represents the ongoing efforts to overhaul the system with more advanced telecommunication infrastructure. In 2011, the National Emergency Number Association (NENA) defined the first i3 architectural framework to standardize the structure and design of the elements that form the software services, databases, network elements and interfaces necessary for processing multi-media calls and data for NG911. That same year, Barnes et al. [4] published the "Technical Considerations for Next Generation 911" document, which provided comments about various aspects of the i3 architecture such as capabilities, applications, transport mechanisms, NG911 participants, interoperability, standards, location capabilities, and cybersecurity. This document was the cornerstone for a nationwide Next-Generation 911 implementation, with the latest revision of the standards published in 2021 [34].

The European Emergency Number Association (EENA), the European counterpart of NENA, is actively engaged in similar efforts. EENA had adapted NENA's standards to European PSAPs by maintaining a close alignment with the i3 architectural framework within their Long Term Definition (LTD) framework architecture [27]. The similarities between the NG112 LTD and i3 architectures reinforce our confidence that the models developed based on the current NG911 system can be applied to other ESCS networks. Furthermore, the architectural framework presented by Corral-De-Witt et al. [8] for the transition from E911 to NG911 in Ecuador also contains common elements to their US counterparts. This makes sense, since they all seek to incorporate the ability to handle new multimedia services into their ESCS and provide better and faster caller localization and routing.

One goal of NG911 is to service multimedia messages [22]. Similar proposals have emerged

in Europe for the design and implementation of a Next Generation platform capable of handling rich media emergency calls [30]. This would enable an ESCS to take advantage of car crash detection systems, already present in modern vehicles, and health monitoring devices in addition to text, images and videos. However, this raises operational questions that might be explored through modeling and simulation such as:

- How would the handling of multimedia messages affect the workload of call takers and dispatchers?
- Would this affect the wait and service time for regular voice calls?
- Would this have any effect on the throughput of regular voice calls?
- How prepared are call takers and what technology do they need to handle multimedia messages effectively?

The NG911 system introduces a crucial shift by using Geographic Information System (GIS) data for caller location and routing. This modern approach replaces traditional databases like the master street address guide (MSAG), automatic location identification (ALI), emergency service number (ESN), and selective routing database (SRDB) [22]. This thesis encapsulates this NG911 aspect in modeling the ESCS network, using the geographic locations and jurisdictional and service boundaries to determine the ESCS elements (vertices) and the communication links (edges) between them.

In summary, since its conception in 1968, the 911 system in the US has organically evolved into a *system of systems*, dependent on dated technology that now requires a paradigm change to take advantage of modern technology. Such paradigm change is being implemented in the Next Generation 911 (NG911) project [49]. Although the FCC offers recommendations to enhance cybersecurity, operations, and funding, the actionable decisions will ultimately hinge on the unique circumstances, priorities, and knowledge at the Public Safety Answering Point (PSAP) level [49].

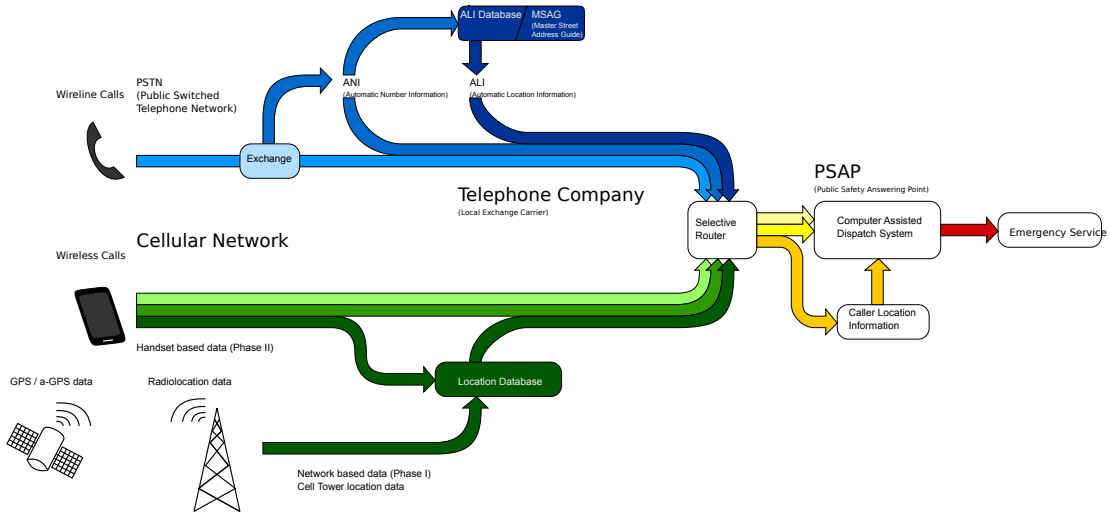
The current transition into Next Generation ESCSes requires substantial efforts in planning and research if it is to avoid past mistakes. Due to the local governance nature of 911, research must be done at various jurisdictional levels: local, state, and national. The flexibility and performance of the "Graphitti" simulator, together with the abstract and mathematical models discussed in this thesis, present the foundation for future simulation research at different scales. Although we based our models on the NG911 system, for practical purposes, these models should apply to any ESCS network because they share many core commonalities.

2.3 The 911 Response Process

The 911 response is a dynamic process with multiple key players. It all starts with the person in need, who makes the initial call for help that connects them to a call-taker at the nearest PSAP. The call-taker gathers essential details and inputs them into a system that pinpoints the caller's location and categorizes the type of emergency. From there, a dispatcher, who might also be the call-taker in some cases, sends the right emergency responders to the scene of the incident. Once the responders are on-site, they provide the necessary assistance. But it does not end there; the system keeps gathering valuable data. Responders file reports, cross-check their assessment with the call-taker's information, and record response times [35].

2.3.1 Call Initiation

Beyond connecting callers with the appropriate PSAP, telephony service providers are mandated to transmit the caller's location — a process that varies in form and precision depending on the calling technology: wireline phones, wireless cellular devices, or Voice over Internet Protocol (VoIP) services. For wirelines, this would typically be a physical address; for wireless, it could involve GPS in the handset or cell tower information, which could result in location inaccuracy of kilometers [3]. Figure 2.1 provides an illustration depicting the process of acquiring the caller's location within the 911 system.



Source: Evan Mason, "911 System," via Wikimedia Commons. Licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license

Figure 2.1: The 911 System call routing for wireline and wireless emergency calls.

2.3.2 Call Routing and Delivery

Call routing aims to link the caller with the appropriate PSAP. Previously, the emergency call was directed to the nearest PSAP based on the caller's location, retrieved from the local telecommunications network. This process is depicted in Figure 2.1. However, the evolution to Next-Generation 911 (NG911) is ushering in a revised routing mechanism reliant on Geographic Information System (GIS) data [7]. NG911 employs GIS databases housing the service boundaries of PSAPs, leading to the redirection of calls to the PSAP serving the caller's precise location.

Published by Barnes and Rosen [3], Figure 2.2 outlines the routing sequence within the NG911 system. This routing framework is tailored for modern mobile calls and internet calling services, integrating wireline and older wireless services through a Gateway, thus standardizing their protocol. Furthermore, a Location-to-Service Translation (LoST) proto-

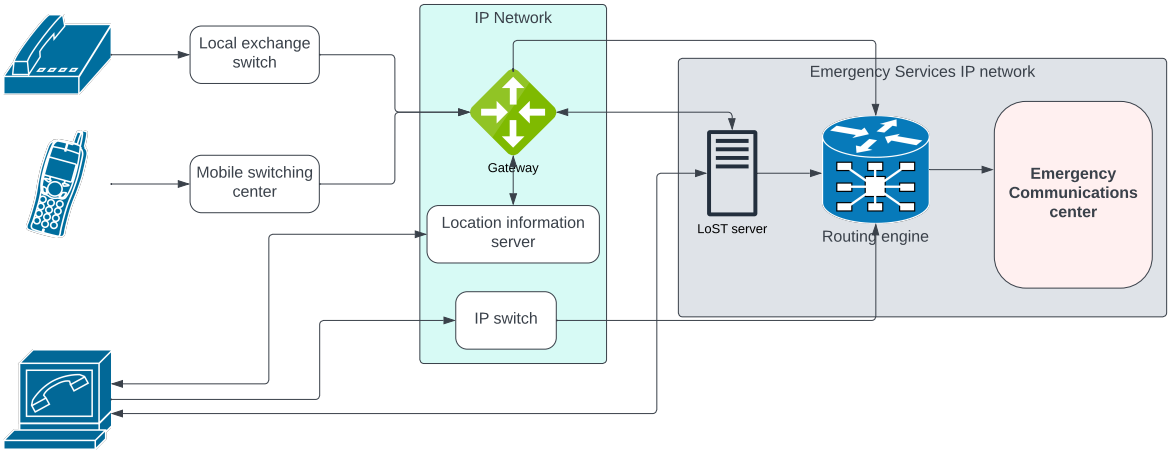


Figure 2.2: The Next Generation 911 (NG911) System call routing framework. Redrawn from Barnes and Rosen [3].

col server unifies location information, either civic addresses or geographic coordinates, into a Uniform Resource Identifier (URI) that is used to route the emergency call. Ultimately, all emergency calls go through the same LoST protocol and routing engine.

2.3.3 Call Processing and Dispatch

After an emergency call reaches a PSAP, call-takers follow a specific call protocol designed to assess the emergency, determine necessary services, and provide essential information for responders [35]. While this protocol might vary among PSAPs, the National Emergency Number Association (NENA) outlines minimum requirements, suggesting that call-takers should gather critical details such as the incident's address or precise location, a callback number, the nature of the call, the time it occurred, any identified hazards, and the caller's identity [33]. NENA also establishes standards for the time intervals between the call arrival at a PSAP and the answering time, as depicted in Figure 2.3. According to these guidelines, NENA recommends that 90% of calls should be answered within 15 seconds of arriving at a

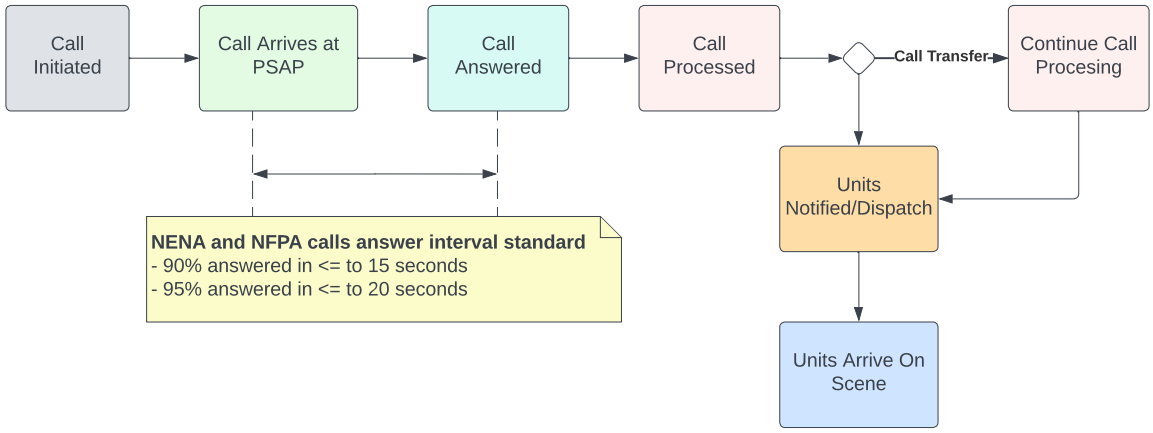


Figure 2.3: 911 call handling and dispatching flow chart. Both the National Emergency Number Association (NENA) and the National Fire Protection Association (NFPA) suggest the same call-answering interval standard. Redrawn from National Emergency Number Association (NENA) and others [33].

PSAP and 95% of calls within 20 seconds — a standard that is also adopted by the National Fire Protection Association (NFPA) [33].

The culmination of the call processing phase ideally results in dispatching a response unit to the emergency. Dispatch responsibilities can differ across jurisdictions, sometimes involving specialized dispatchers within certain PSAPs. Once the essential information is gathered, call-takers proceed to either dispatch the appropriate response unit or transfer the call to a designated dispatcher. This dispatching process encompasses critical decisions regarding the types and quantity of response units required. Furthermore, all pertinent information must be effectively communicated and logged into the Computer-Aided-Dispatch (CAD) system that assists responders in swiftly identifying the required type and speed of response [35].

2.3.4 Incident Response

The primary goal of an Emergency Services Communication System (ESCS) is to ensure a prompt and appropriate response to incidents. Depending on the emergency's nature, this response might involve one or multiple units from various services, such as police, fire, or emergency medical services (EMS).

The interval between the initial call for assistance and the arrival of a response unit at the scene is known as the *response time*. This time frame is critical within an ESCS and serves as a key metric for evaluating police performance, as noted in Stevens et al. [46] work on the role of response time in assessing police performance. Stratmann and Thomas [48] support this by linking a decrease in 911 response times to a significant reduction in homicide rates.

Once a response unit is dispatched, it remains occupied for the combined duration of the response time, the time it takes to travel to the site of the incident, and the period dedicated to delivering necessary services at the emergency scene. This delivery of emergency services marks the final stage of the emergency response process.

Chapter 3

LITERATURE REVIEW

Previous research on ESCSes has predominantly focused on data analysis and the development of predictive models aimed at identifying incidents and call patterns; whereas simulation remains relatively limited and unexplored. Neusteter et al. [35] analyzed 35 papers delving into 911 data within policing contexts, revealing two main methodological approaches. One approach relies on standardized metrics such as call volume, types, and response time, offering a comprehensive yet less detailed perspective. The second approach employs complex techniques to model caller behavior, track call-type patterns over time, and identify obstacles to prompt responses. Notably, none of the studies in this review took advantage of simulations.

The little work that uses simulation to study ESCSes primarily concentrates on vulnerabilities to cyberattacks such as DDoS [2, 32, 54]. The only exception is Gustavsson [20], who employed simulation within Operation Management (OM) of an Emergency Call Center. This research introduced a method for modeling the burst behavior in the call arrival process that, although different than the cluster-point process presented here, also leverages the link between emergency calls and the incidents that trigger them [21]. Gustavsson [20] studies important aspects of an ESCS such as the non-stationarity of call arrivals, stochastic call-taker behavior, and the relationship between the geographical distance between PSAPs to their service area and service time. However, their simulations were tailored to a Swedish emergency call center provider, requiring extensive work to implement for other emergency call centers. In addition, their work recognizes the importance of call abandonment but does not include it in their models.

Mirsky and Guri [32] researched the vulnerability of the 911 system to DDoS attacks

by anonymous unblockable calls originating from mobile phones. They conducted practical assessments by deploying bots designed for executing such DDoS attacks within a limited cellular network. They then simulated the potential impact of a DDoS assault on an E911 network, finding that fewer than 6000 bots could disrupt emergency services throughout a state for extended periods. Their simulation employed a lumped discrete event simulation of the busiest time of the day to mirror the potential overload of Selective Routers due to a botnet; however, it lacked depth in representing the comprehensive behavior of 911 entities such as PSAPs and responders.

Xue and Roy [54] also studied DDoS attacks, using a cyberphysical queuing-network model of the 911 system. Their simulation included models for a time-varying Poisson process for call arrival, an exponentially distributed service time, dispatching dynamics, responder travel time, and response time. However, their study omitted other important processes, such as call abandonment and redial, and focused on a small area of 10x15 miles (Charlotte, NC) with no discussion of model scalability.

Other research works explore the performance impacts of an Internet Protocol (IP)-based 911 network [14] and the increase in attack surface posed by the text-to-911 service [2]. Desai et al. [14] employed a testbed to investigate the performance impacts of implementing Transport Layer Security (TLS) within the Session Initiation Protocol (SIP) and Location to Service Translation (LoST) on IP-based emergency calls; revealing that the TLS handshake typically accounts for about 20 percent of the call setup delay under normal circumstances. Bal et al. [2] studied how text messages to 911 could be abused by attackers, introducing a modular Text Analysis Engine for early detection and mitigation of cyberattacks via text messages to 911.

Chapter 4

METHOD: ESCS DOMAIN ARCHITECTURE

Our framework is designed to simulate the operations of large heterogeneous ESCS networks, and to easily adapt diverse mathematical models for the various stochastic processes involved. It can be used for studies of cybersecurity threats, catastrophic events, and regular ESCS operation. We model three crucial aspects of an ESCS: (1) system *structure*, (2) its main components' internal *behavior*, and (3) the *interactions* among these components. These considerations are examined in the following subsections in order, from general to specific.

4.1 Structure: ESCS as Graph-Based Systems

Initiating a 911 call sets in motion a cascade of events that involve various layers of technology and emergency personnel. ESCSes are complex heterogeneous multi-network systems that have safety-critical, operational, and regulatory concerns. For any model to be useful, however, it must be simpler than the real system; thus, one must reduce it to the core elements. In light of this, the following three fundamental ESCS entities were identified and characterized: (1) Caller Regions (CR), (2) Public Safety Answering Points (PSAPs), and (3) First Responders.

Here, *Caller Regions* denote the geographic areas from where calls originate, *PSAPs* are emergency call centers responsible for answering emergency calls and dispatching first responders, and *Responders* indicate the headquarters from where first responders are dispatched (e.g. police and fire stations). These components are arranged in a network that underlies the GIS-based call routing and dispatching dynamics of an ESCS, as described in Subsection 2.3.2.

ESCSes are part of a family of complex systems with cyber-physical dependencies that are effectively represented using graphs. For instance, in the work by Jalving et al. [23], an algebraic graph abstraction was introduced to represent physical connectivity in complex optimization models, along with a computing graph abstraction to depict communication connectivity in computing architectures, similar to the approach in this thesis. Graphs are used to model network nodes (*vertices*) and their connections (*edges*) in a pairwise relationship. The particular model, in this thesis, is a *directed graph*, where vertices denote the aforementioned ESCS entities and edges represent the communication channels between them.

The topology of an ESCS network is determined by the geographic placement and service coverage of its components. For example, when a caller region falls within the service area of a PSAP, these entities are interconnected by bidirectional edges. Likewise, bidirectional edges are established between first responder locations and the corresponding PSAPs serving their geographic vicinity.

This modeling technique was applied to the NG911 network of the State of Washington's King County. The network topology was extracted from various GIS datasets by encoding the jurisdictional and neighboring relationships of the 911 components in a directed graph. To that end, the Washington State 911 Coordination Office provided GIS datasets that contained the jurisdictional and service boundaries for PSAPs, police, fire, and emergency medical services (EMS) of the State of Washington 911 network. Figure 4.1 presents a depiction of the service boundaries of the State of Washington's PSAPs, overlayed on an OpenStreet map.

4.1.1 Performance Considerations in Simulations of Large Complex Systems

Depending on the size of the modeled region, the ESCS network can be large, in the range of tens of thousands of nodes and connections, each expressing complex behavior. Therefore, performance is a key architectural concerns in our simulations, prompting the consideration of a future parallel implementation of the described models. Lacking parallelization, simulations

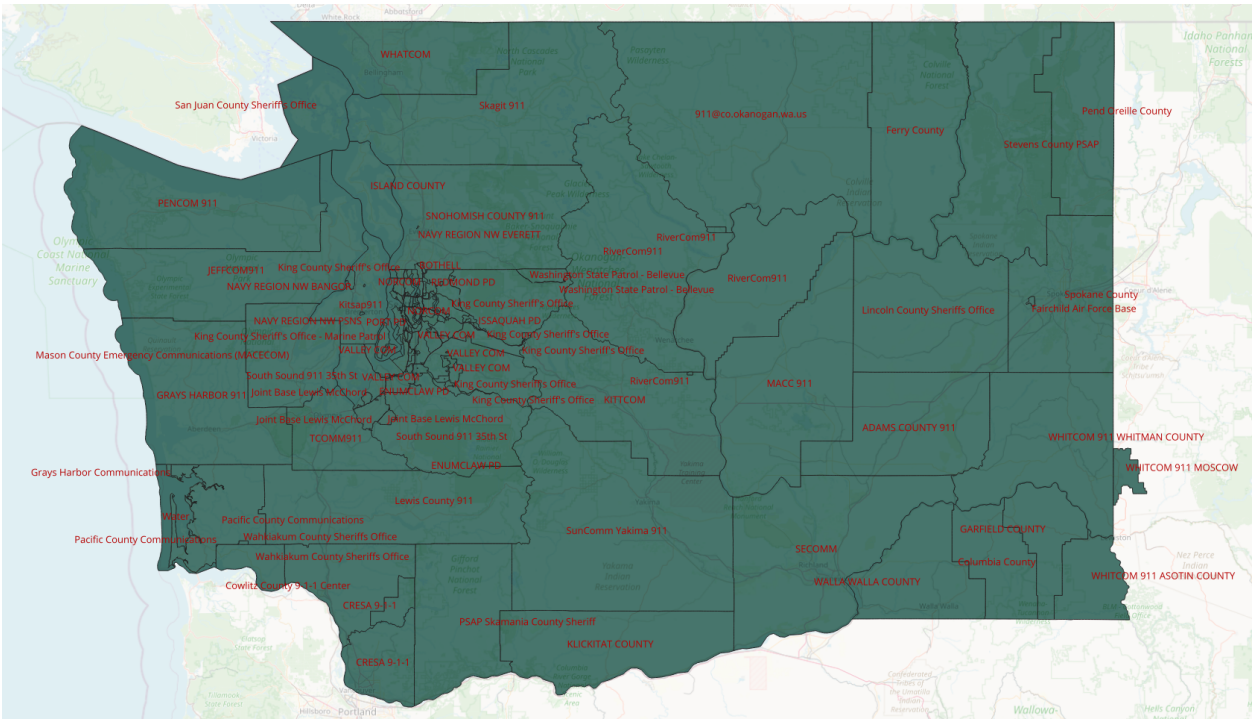


Figure 4.1: Service boundaries of the State of Washington's PSAPs

of such complex systems will result in significant execution time. Stiber et al. [47] found that the parallel GPU implementation of a biological neural network simulation, using modern GPU architectures and libraries (CUDA), achieved a 40-fold increase in performance.

The models described in this thesis were implemented using the Graphitti simulator to take advantage of its performance capabilities in simulating large graph-based systems. Furthermore, Graphitti is designed to simulate hybrid discrete/continuous graph-based systems and facilitate the migration and validation of large-scale (tens of thousands of vertices, millions of edges) and long-duration (billions of time steps) simulations to GPUs. These capabilities were demonstrated in simulations of development and tuning in biological neural networks at that scale [47].

Parallel programming can be complex. It involves the challenging task of understanding the sequencing of threads or processes running concurrently and one must exercise caution

to avoid introducing race conditions. Consequently, debugging can be an intimidating task. Message passing and synchronization mechanisms, such as mutex locks and semaphores, are often used to deal with some of the challenges introduced by parallelization. However, these could cause threads to block execution under certain circumstances. In Graphitti, the strategy for eliminating race conditions is to avoid using variables that could be accessed and manipulated by different threads. Abstractly, one can think of each vertex as a thread of execution, each having ownership over its data structures, and thus, not allowing their manipulation by other vertices.

To facilitate debugging, an initial serial CPU version of the models is implemented with parallelism considerations, serving as the foundation for the subsequent GPU implementation. Consequently, bugs found in the CPU implementation can be correspondingly rectified in the GPU version. The representation of entities possessing data structure ownership, alongside the mirrored CPU and GPU implementations, provides a framework for addressing the intricacies inherent to parallel computing.

4.2 Behavior: Discrete Events Queueing Model

In essence, PSAPs are call centers specialized for emergencies. Therefore, discrete Event Queuing Models extensively used in the management of call centers are suitable for modeling the processing of emergency calls by PSAPs. Furthermore, the same concept can be applied to emergency responder nodes, for modeling dispatching and response actions, forming a multi-queuing model.

Figure 4.2 illustrates a queuing model, generally used in the modeling of call centers. Conceptually, a call center contains k trunk lines with up to the same number of workstations ($w \leq k$) and agents ($n \leq w \leq k$) [17]. One of three scenarios occurs when a call arrives: (1) it is answered right away if there is an available agent, (2) the call is placed in a queue if there are no available agents, or (3) the caller receives a busy signal if there are no trunks available.

In this model, we think of an agent as a resource that is occupied while a caller is receiving

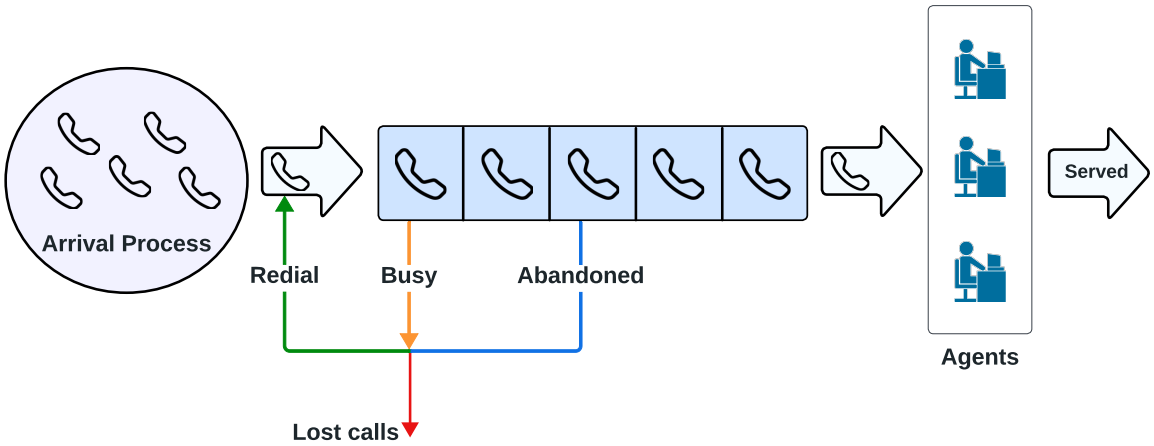


Figure 4.2: Call center as a queuing system. The diagram represents a system with 3 agents (n) and 8 trunk lines (k); therefore, the size of the waiting queue is 5 ($n - k$).

assistance and immediately released once it has been served. Calls are lost due to blocking when all trunks are busy or when a caller abandons the queue due to impatience, possibly redialing soon after. Consequently, arriving calls come from either those who made an initial call, those who got a busy signal, or those who abandoned the queue after waiting.

Stochastic processes such as call arrivals, service time, and customer impatience –which represent call abandonments– must be modeled through random variables, drawn from a suitable probability distribution. Existing research in this area provides possible candidates but their goodness of fit has to be evaluated based on data obtained from the real system [25]. The details of the modeling of stochastic processes are discussed in Chapter 5.

4.3 Interactions: Communicating Finite State Machines

The use of Communicating Finite State Machines (CFSM) proves to be well-suited for modeling the interactions between ESCS components. This framework was researched by Brand and Zafiropoulo [5] in the domain of distributed computing. They employed a common

representation of communicating processes, where each process functions as a finite-state machine and inter-process communication occurs through reliable full-duplex first-in first-out (FIFO) channels. This approach offers several advantages, such as the ability to analyze the execution of each process separately, rather than considering the system as a whole, thus partitioning the complex behavior into manageable subproblems. Notably, this model abstraction aligns with the potential for a future parallel GPU implementation.

A simulation begins with an initial configuration defined by specific parameters and proceeds for a predetermined number of time steps. The values of these parameters and simulation variables at each time step collectively constitute the "system state." For an ESCS graph, the system state is a comprehensive representation of every vertex and edge within the graph. During each time step, ESCS entities receive inputs and undergo state transitions, potentially resulting in the transmission of outputs to other vertices.

For performance purposes, Graphitti is designed to facilitate the implementation of simulation models on Graphic Processing Units (GPUs). These GPU implementations enable high-performance simulations of large complex systems but at the same time, their high parallelization presents challenges for inter-process communication. Modeling the interaction between ESCS entities as Finite State Machines that communicate with each other by transferring event messages through their connecting edges, provide a useful abstraction for highly parallelized systems such as future GPU implementations of these models. In Figure 4.3, the depiction illustrates how various types of ESCS vertices both receive messages through their incoming edges and transmit messages via their outgoing edges.

Each vertex is depicted as a Finite State Machine (FSM), with state transitions determined by arising events, such as call initiation, call answer, unit dispatch, call termination and end of emergency response. The internal behavior of each vertex aligns with a discrete event queuing model, as illustrated in Figure 4.4. Consequently, each vertex contains a First-In-First-Out (FIFO) queue that dictates the sequence in which events are processed and a collection of attributes, some of which are expressed as stochastic processes using probability distributions. The PSAP and responder vertices maintain lists of resources, representing

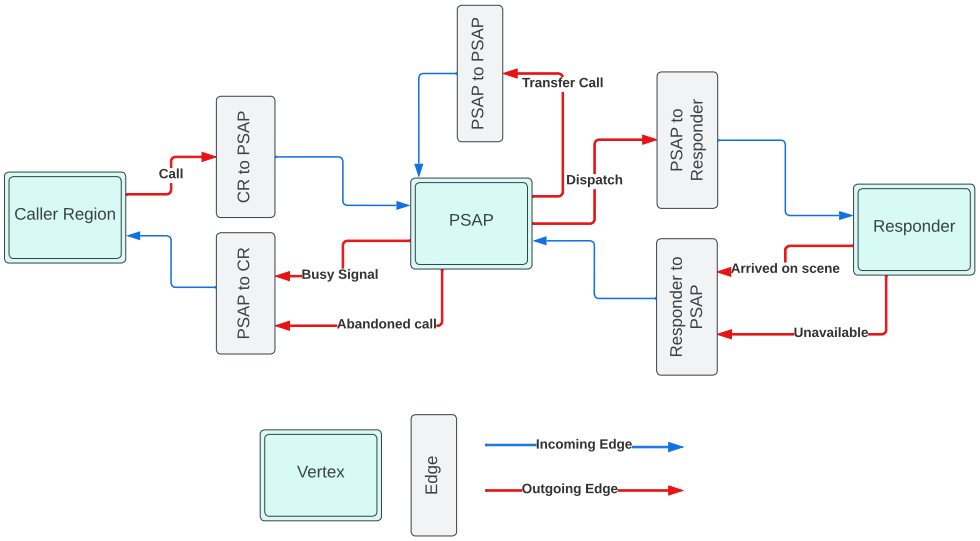


Figure 4.3: Illustration of vertices communication. Messages are exchanged via their incoming and outgoing edges.

servers and responder units, respectively. Moreover, each vertex records metrics for both serviced calls and emergency responses.

4.4 Graphitti Implementation

The successful implementation of the ESCS models requires that the simulator can load diverse graphs, such as those essential for representing ESCS networks, and accept external events as inputs. These features were not present in Graphitti thus requiring their implementation as part of this Thesis. It was decided that the graph and event inputs would be provided through external files because it optimizes flexibility, clearly separating the data from the algorithmic implementation of the models.

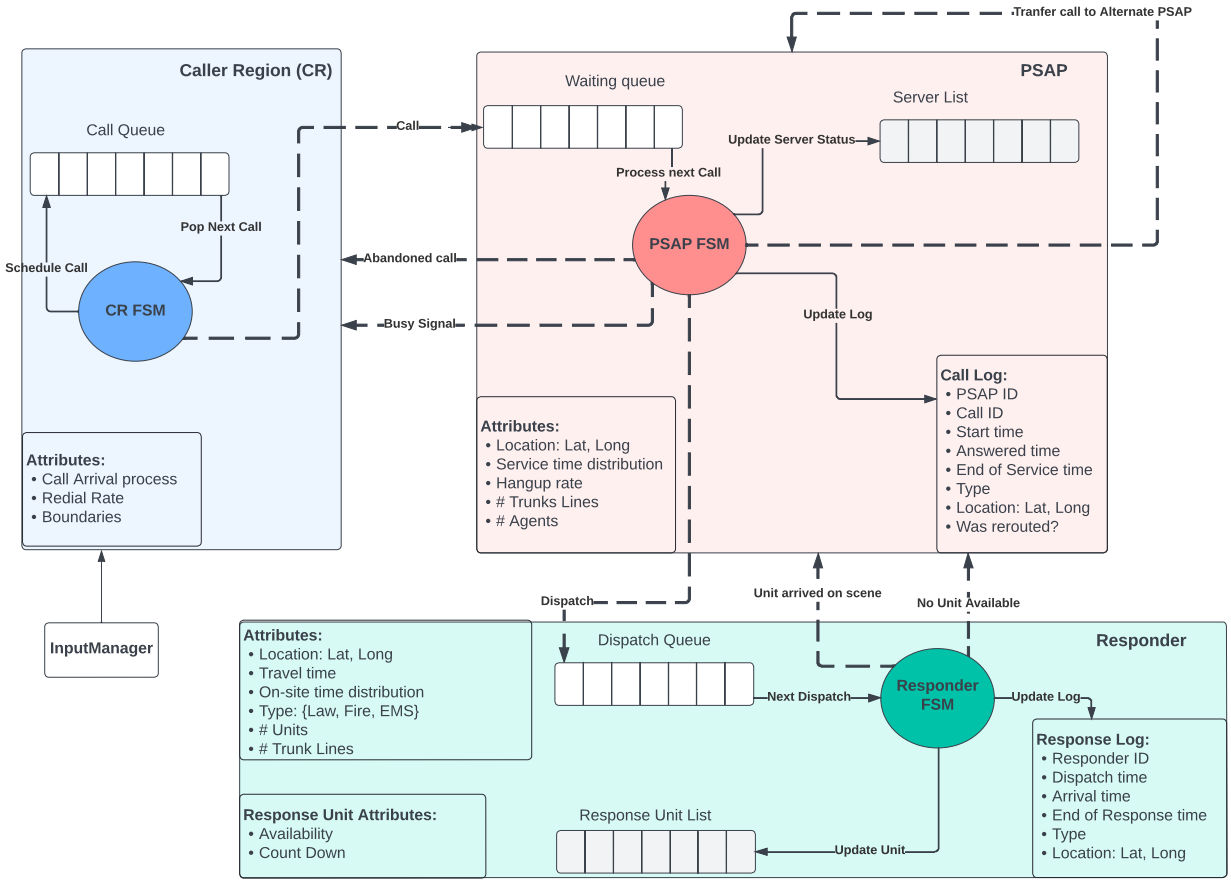


Figure 4.4: 911 System as Communicating Finite State Machines

4.4.1 GraphML: The Graph File Format

The chosen format for representing graphs and their attributes, as inputs for Graphitti, was GraphML. This decision was made after careful consideration, weighing the options of developing an in-house format and against other widely recognized formats such as GraphSON [24] and GEXF [18]. GraphML is a well-documented industry standard with extensive support from various graph libraries, including the C++ Boost Graph Library (BGL) [43]. The GraphML standard, initially proposed by Brandes et al. [6] in 2002, prioritizes simplicity, generality, extensibility, and robustness, aligning perfectly with our specific requirements. Furthermore, GraphML is XML-based, accommodating typed data, thus seamlessly integrating with Graphitti's use of the strongly typed C++ language and XML as the chosen format for configuration and output files.

In Listing 4.1, you can see an illustrative ESCS graph expressed in the GraphML format. This definition outlines a directed graph featuring two nodes and two edges. Each node has various attributes, namely objectID, name, type, y, x, servers, and trunks, each specifying their corresponding data types (int, double, and string).

Listing 4.1: Example of ESCS Graph Using GraphML Format

```
<?xml version='1.0' encoding='utf-8'?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
<key id="trunks" for="node" attr.name="trunks" attr.type="int"/>
<key id="servers" for="node" attr.name="servers" attr.type="int"/>
<key id="x" for="node" attr.name="x" attr.type="double"/>
<key id="y" for="node" attr.name="y" attr.type="double"/>
<key id="type" for="node" attr.name="type" attr.type="string"/>
<key id="name" for="node" attr.name="name" attr.type="string"/>
<key id="objectID" for="node" attr.name="objectID" attr.type="string"/>
<graph edgedefault="directed">
    <node id="0">
        <data key="objectID">PSAP120@kingcounty.gov</data>
        <data key="name">SEATTLE PD</data>
```

```
        <data key="type">PSAP</data>
        <data key="y">47.27281192700005</data>
        <data key="x">-122.31477654167</data>
        <data key="servers">6</data>
        <data key="trunks">16</data>
    </node>
    <node id="1">
        <data key="objectID">EMS104@kingcounty.gov</data>
        <data key="name">Seattle FD - Belltown</data>
        <data key="type">EMS</data>
        <data key="y">47.61620068714826</data>
        <data key="x">-122.34445404482534</data>
        <data key="servers">6</data>
        <data key="trunks">7</data>
    </node>
    ...
    <edge source="0" target="1"/>
    <edge source="1" target="0"/>
</graph></graphml>
```

4.4.2 The Input File Format (XML)

Graphitti characterizes system behavior as a discrete sequence of events. In the Biological Neural Network model, these events materialize as spikes, triggered when a neuron's membrane electrical charge attains a particular threshold (spiking neuron model). In contrast, in the NG911 model, the events take the form of calls or other messages exchanged among the various entities.

The system's behavioral logic is encoded within classes that extend the appropriate Graphitti's core component. However, there are situations in which we require the ability to introduce external events as inputs. For instance, this capability enables us to replicate past events from a 911 call log or simulate external neurological stimuli. To this end, the list of

events is provided to Graphitti as an XML input file.

Unlike GraphML, which offers a standardized format for graph representation, we did not find a pre-existing standard format for describing event inputs. Consequently, we developed an in-house XML-based format tailored to our requirements. This format comprises a root node labeled "simulator_inputs" and a "data" node, which acts as a container for a well-structured list of events categorized by "vertex" nodes. Additionally, essential attributes can be incorporated into the "data" node. For a practical example of an ESCS input file, you can refer to Listing 4.2.

Listing 4.2: Example of an ESCS Input File

```
<?xml version='1.0' encoding='UTF-8'?>
<simulator_inputs>
<data description="SPD_calls" clock_tick_size="1" clock_tick_unit="sec">
    <vertex id="194" name="SEATTLE␣PD␣Caller␣region">
        <event time="0" duration="0" x="-122.38496236371942" y="
           47.570236838209546" type="EMS"/>
        <event time="34" duration="230" x="-122.37482094435583" y="
           47.64839548276973" type="EMS"/>
        <event time="37" duration="169" x="-122.4036487601129" y="
           47.55833788618255" type="Fire"/>
        <event time="42" duration="327" x="-122.38534886929502" y="
           47.515324716436346" type="Fire"/>
...
    <vertex/>
...
<data/>
<simulator_inputs/>
```

4.4.3 Adding Support for GraphML and Inputs Files

Following Graphitti's architectural approach, which uses a *ParameterManager* class for loading the simulation parameters from a user-defined configuration file, two new *Manager* classes

were integrated:

1. GraphManager: Loads the GraphML file

2. InputManager: Loads the simulation Inputs file

Figure 4.5 provides a component diagram of Graphitti's architecture, illustrating the roles of the *GraphManager* and *InputManager* components. The *Connections* and *Layout* components, in addition to using the parameters from *ParameterManager*, rely on the graph contained within *GraphManager* to generate Graphitti's internal representation of the network. The role of the *InputManager* is to load the list of events into internal queues, one per vertex, to be retrieved by the respective vertex during a simulation. This design enables the handling of a wide range of graph structures and input sources, whether they are from real-world systems or synthetically generated. As a result, one can conduct experiments by adjusting the network's structure or the input arrival rate without making changes to the simulator's internal code.

The creation of synthetic graphs and input events has been externalized from Graphitti, and their management is now handled through scripting. The *Python* programming language is employed in this work to generate experimental synthetic data. For example, in Section 5.1, Python is used to model call arrivals as a cluster point process. Similarly, retrieving King County's NG911 graph network, as detailed in Section 4.1, is also scripted. Although this approach effectively demonstrates the framework's capabilities, it's essential to note that these scripts are highly specialized for the current problem and offer limited reusability.

GraphManager Class

The GraphManager class leverages the Boost Graph Library (BGL) for loading GraphML files. It is designed to handle properties associated with graphs, edges, and vertices through the use of the following C++ structures: *GraphProperty*, *EdgeProperty*, and *VertexProperty*. Before loading a graph, users must inform the GraphManager which member variable within

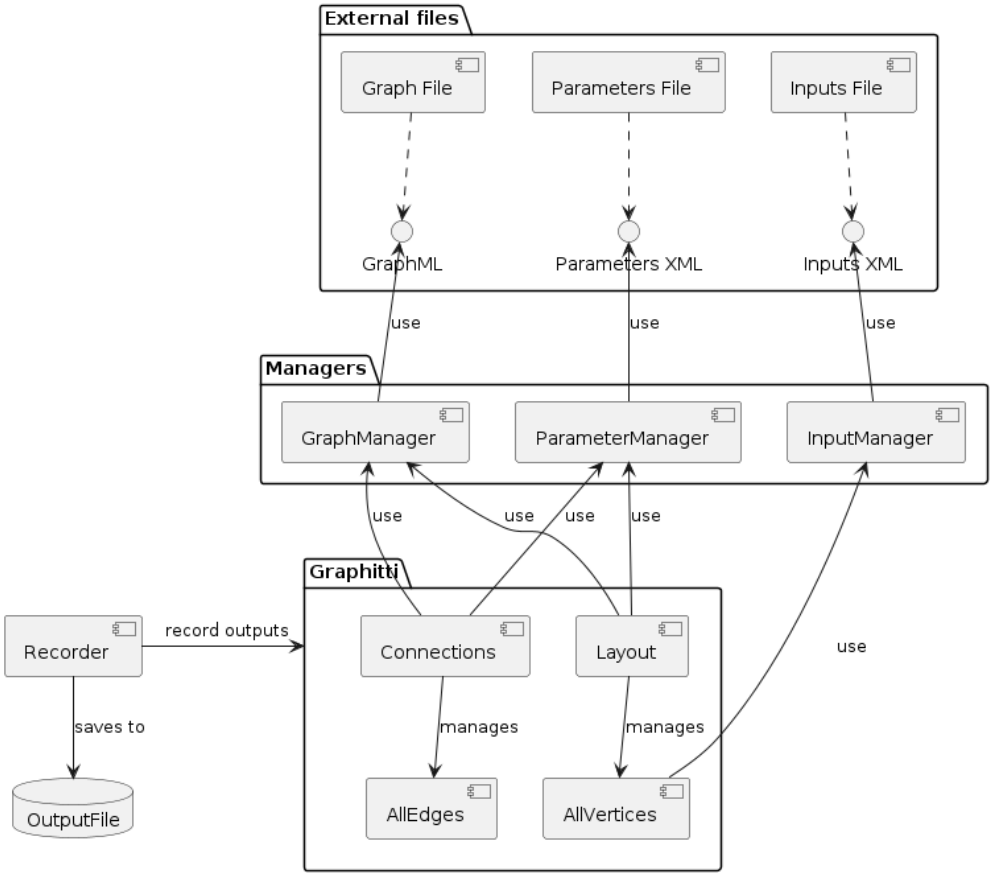


Figure 4.5: Graphitti's Component Diagram with support for GraphML and inputs file

a data structure should store each of the node attributes from the GraphML file. This is done using the *registerProperty* method, which requires the attribute's name as a *string* and a pointer to the data member within the data structure responsible for storing the attribute's value. These attributes are referred to as "properties" in the interface of the GraphManager class which is designed to support graphs with various types of properties. Consequently, the data type of the data member must match the type specified in the GraphML file. Listing 4.3 is a code snippet of how the Layout911 class, an NG911 Layout specialization, registers the vertex properties with the singleton GraphManager object.

Listing 4.3: Code snippet: Registration of vertex properties

```
void Layout911::registerGraphProperties()
{
    // The base class registers properties that are common to all vertices
    Layout::registerGraphProperties();

    // We must register the properties before loading the graph
    GraphManager &gm = GraphManager::getInstance();
    gm.registerProperty("objectID", &VertexProperty::objectID);
    gm.registerProperty("name", &VertexProperty::name);
    gm.registerProperty("type", &VertexProperty::type);
    gm.registerProperty("y", &VertexProperty::y);
    gm.registerProperty("x", &VertexProperty::x);
    gm.registerProperty("servers", &VertexProperty::servers);
    gm.registerProperty("trunks", &VertexProperty::trunks);
}
```

After all the properties have been successfully registered, the process of loading the GraphML file becomes a matter of invoking the *readGraph* method. Following this step, consumer code can seamlessly traverse the vertices and edges to construct Graphitti's internal network representation, as illustrated in Listing 4.4. This code snippet illustrates how Layout911 retrieves the "x" and "y" coordinates of the vertices from the GraphManager, subsequently copying these values into its internal data structures.

Listing 4.4: Code snippet: Iterating over the graph's vertices

```
GraphManager::VertexIterator vi, vi_end;
GraphManager &gm = GraphManager::getInstance();
for (boost::tie(vi, vi_end) = gm.vertices(); vi != vi_end; ++vi) {
    assert(*vi < numVertices_);
    xloc_[*vi] = gm[*vi].x;
    yloc_[*vi] = gm[*vi].y;
}
```

InputManager Class

The design of the *InputManager* class closely follows the principles of *GraphManager*. It supports any user-defined event input property as long as it aligns with the input file format described in Subsection 4.4.2. As with the GraphManager, the event properties must be registered using its *registerPropery* method, which is an exact homolog of the GraphManager's method of the same name. A practical example of this step is given in Listing 4.5, which is a code snippet extracted from the *All911Vertices* class — an NG911 specialization of *AllVertices*.

Listing 4.5: Code snippet: Registration of event properties

```
inputManager_.registerProperty("vertex_id", &Call::vertexId);
inputManager_.registerProperty("time", &Call::time);
inputManager_.registerProperty("duration", &Call::duration);
inputManager_.registerProperty("x", &Call::x);
inputManager_.registerProperty("y", &Call::y);
inputManager_.registerProperty("patience", &Call::patience);
inputManager_.registerProperty("type", &Call::type);
```

Internally, the *InputManager* maintains event queues for each vertex, in a First-In-First-Out (FIFO) fashion, that are subsequently accessible to the respective vertices during the inter-epoch interval of a simulation. As each new epoch begins, all scheduled events for that epoch are transferred into the internal queue associated with the relevant vertex.

Queue: CircularBuffer Class

The queueing of events is inherent to the "discrete event queueing" model used in this work. As a result, we designed a *CircularBuffer* class that adheres to a basic FIFO queuing policy. Opting to create a bespoke data structure rather than relying on an existing one, like the CircularBuffer class in the Boost library, offers the benefit of streamlining the transfer of data to GPU memory because we are in control of its internal representation. This strategic

decision positions Graphitti favorably for potential future GPU implementations of these models.

4.4.4 Implementation of Vertex Communication and State Transition

One advantage of the Communicating FSM abstraction is its capability to consider the execution sequence of each vertex in isolation. Interactions between vertices occur through message exchanges across their connecting edges. Consequently, the system's behavior at each time step can be divided into two distinct phases:

1. a Communication Phase, when messages are received by the vertices, and

2. a State Transition Phase, during which the vertices undergo state transitions due to internal or external events

4.4.5 The Communication Phase

During the communication phase, vertices must pull, from their incoming edges, the messages that were sent by other vertices during the state transition phase. In the case of the NG911 models, these messages include emergency calls, dispatch instructions, and availability status. The flow diagram in Figure 4.6 describes a sequential implementation of the communication phase where vertices pull incoming messages by looping over all incoming edges.

In a parallel implementation, this "pulling strategy" prevents multiple processes from simultaneously accessing and modifying data structures, eliminating the need for locking. The process that initiates a message, places it in its outgoing edge during the state transition phase. Subsequently, the receiving process is responsible for retrieving and processing messages from its incoming edges. In this graph-based model, the sender and receiver processes correspond to the source and destination vertices of an edge. This approach offers the added benefit of enabling the use of a parallel reduction algorithm when managing a substantial number of incoming edges.

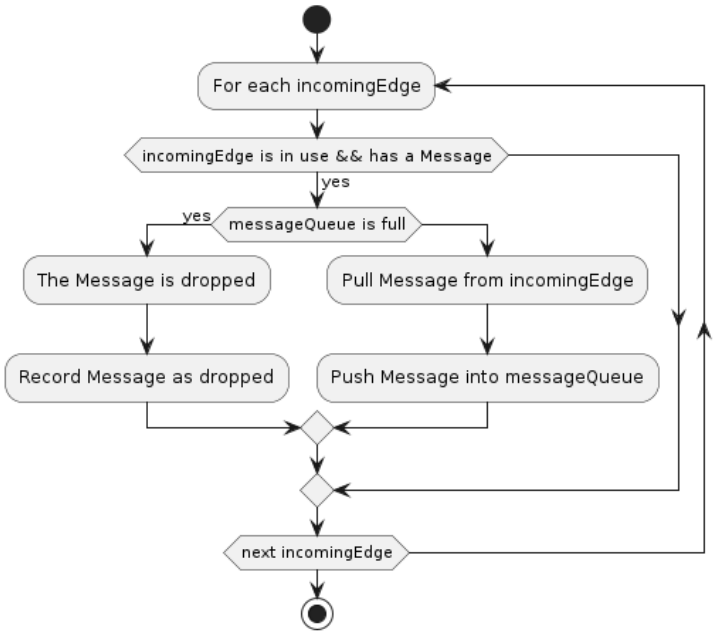


Figure 4.6: Flow Diagram of the Communication Phase

Regarding message handling in the current implementation, if there is available space, the destination vertex stores the message in its designated waiting queue. However, if space is unavailable, the message is flagged and recorded as a dropped message. Decisions about how to manage dropped messages are delegated to the source vertex.

4.4.6 The State Transition Phase

During the state transition phase, vertices react to internal or external events, potentially undergoing state transitions. The state transitions in each of the three main ESCS entities identified in Section 4.1, and contextualized as FSMs in Section 4.3, are dictated by a discrete-event queuing model as described in Section 4.2. This section provides details of the implemented algorithm that characterizes each entity's FSM.

Caller Region State Transition

A Caller Region denotes the geographical zone from which calls initiate, with a rate of occurrence defined by the inputs file as described in Subsection 4.4.2. The call arrival process in the present implementation is modeled as a cluster point process, described in Section 5.1.

Graphitti Simulations are divided into epochs, which are composed of many time steps. Figure 4.7 presents the state transition algorithm of caller regions for one time step. In between epochs, all calls scheduled for the next epoch are loaded into the Caller Region's queue. Then, at each time step, a caller region executes two actions: (1) checks for any dropped calls and decides if redialing is necessary, and (2) sends the next call to the corresponding PSAP through an outgoing edge.

The current implementation requires that no more than one call occurs at a Caller Region per time step. This is enforced by subdividing the geographic area that encompasses a PSAP's service boundaries into multiple regions or adjusting the duration of a time step, the latter being an adjustable parameter in the "Inputs File" as presented in Subsection 4.4.2.

Public Safety Answering Point (PSAP) State Transition

The primary function of a PSAP is to respond to emergency calls originating from its Caller Regions and coordinate the dispatch of suitable emergency responders. The algorithm outlined in Figure 4.8 delineates the steps taken by a PSAP vertex during each time step: (1) identify servers that have concluded their service for a call, and (2) allocate new calls to servers that are currently available.

In this context, a "server" represents an individual, at the PSAP, responsible for handling calls. When a server finishes a call, it becomes available to attend to new ones. However, before assuming their next call, servers are tasked with fulfilling their dispatch responsibilities, which include transmitting a dispatch message through an outgoing edge to the nearest suitable responder. Furthermore, they are required to document essential call metrics, including the start time, response time, conclusion time, and any instances of call abandonment.

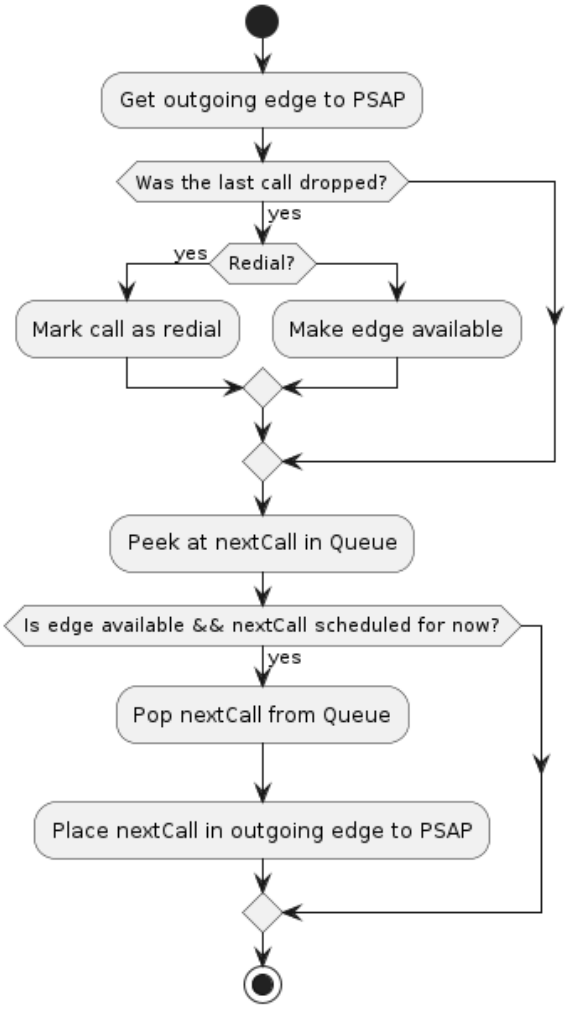


Figure 4.7: Caller Region Vertex Flow Diagram

All available servers are assembled into a list and iteratively processed for call assignments. This process continues until there are either no more available servers or no more calls remaining in the waiting queue. To account for possible call abandonments, a check is performed at the loop outset. If a caller's wait time surpasses their patience threshold — the duration they are willing to wait in the queue — their call is classified as abandoned and recorded as such.

Upon getting a call assignment, a server is temporarily occupied for the call's duration. The duration of the busy time is governed via a countdown variable, reverting to an available state once this countdown expires. Lastly, the count of occupied servers is updated and later used to calculate the available space in the queue for the next time step.

Emergency Responder State Transition

A responder vertex designates the location from which responder units are dispatched. The algorithmic framework, depicted in Figure 4.9, shares a likeness with that of the PSAP vertex, rooted in their common discrete event queuing model. Consequently, both algorithms comprise two analogous phases. Initially, units that have completed their response to an emergency incident document their response metrics and are reinstated as available units. Subsequently, emergency incidents are assigned to available units until either all units are busy or no more incidents remain in the dispatching queue.

The time during which a response unit remains busy is calculated as the sum of the driving time to the incident location and the time spent at the scene (on-site time). Like the servers in a PSAP, this duration is monitored using a countdown variable. The count of occupied units is also updated as it influences the estimation of the waiting queue's available capacity for the next time step.

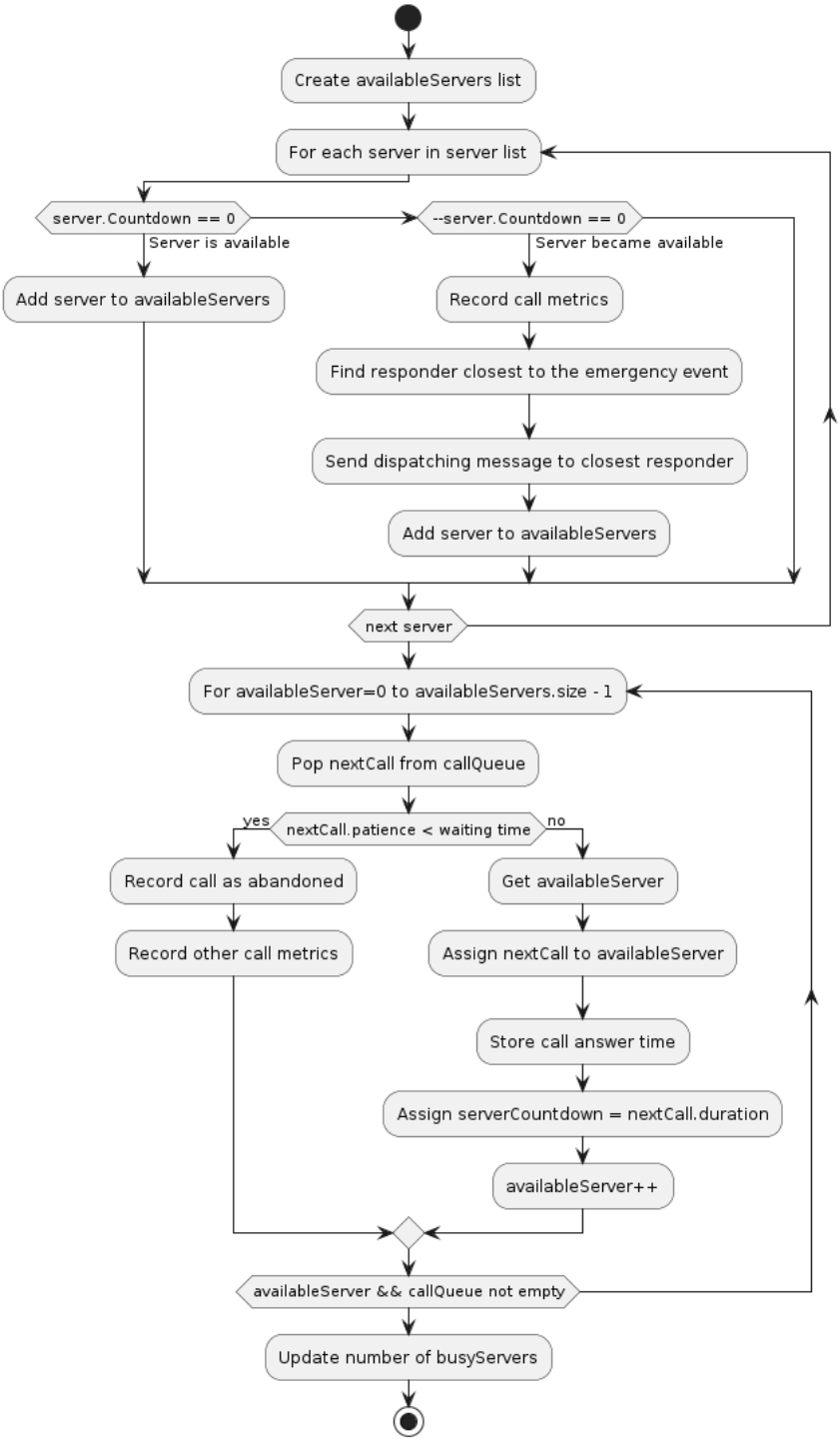


Figure 4.8: Public Safety Answering Point (PSAP) Vertex Flow Diagram

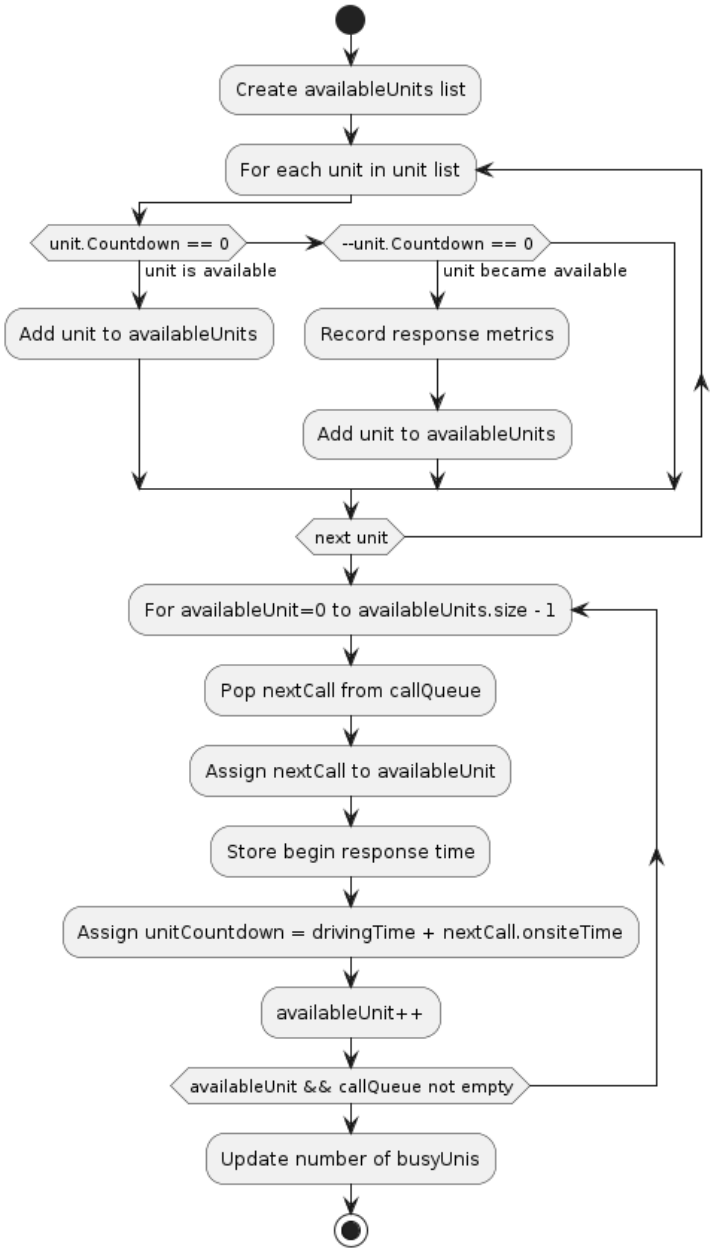


Figure 4.9: Responder Vertex Flow Diagram

Chapter 5

METHOD: ESCS MATHEMATICAL MODELS

5.1 Call Arrival as Realization of a Spatio-Temporal Cluster Point Process

A *spatio-temporal cluster point process* is a random phenomenon that describes the arrangement of events within a given area and along time, exhibiting a tendency to form clusters or groups instead of being randomly distributed. This process is characterized by three key elements: (1) a primary process, (2) a secondary process, and (3) the systematic aggregation of secondary points. In simpler terms, a cluster point process involves aggregating clusters of events associated with the points of a primary process, often referred to as the parent process [40].

Cluster point processes, exemplified by the Neyman-Scott process [36], have demonstrated remarkable effectiveness in modeling natural phenomena. Their versatility extends to diverse fields including cosmology [36], rainfall analysis [9], seismic events [1], and neural spike trains [19]. These examples underscore the ability of cluster point processes to capture complex real-world phenomena accurately. In this work, we use a spatial-temporal cluster point process for modeling the relationship between emergency calls and the emergency incidents that generate them. Here, the primary process represents the occurrence of emergency incidents and the secondary process represents the resulting emergency calls. By adopting this approach, we can effectively capture the spatial and temporal clustering of call arrivals in proximity to emergency incidents.

5.1.1 The primary process

Any point process that can effectively capture the temporal and spatial characteristics of emergency events can serve as the primary process. It is essential to highlight that the dis-

tribution of points in time and in space operates independently, allowing for both stochastic and deterministic occurrences as well as regular and irregular patterns. For simplicity, we adopt a Poisson distribution to model the primary point process in time and the points are assumed to be uniformly distributed in space.

5.1.2 The Secondary Process

The emergency calls are simulated as a Poisson point process inside a circle, centered at the corresponding primary event that generated the calls. For each cluster, we need to model (1) the number of calls, (2) the time between each call and the emergency event, and (3) the location of the emergency calls.

5.1.3 Number of emergency calls

Each cluster's circle is defined by a radius $r > 0$ and an intensity $i > 0$. The intensity indicates the number of points per unit area; thus the number of secondary points (n) is

$$n = \pi \cdot r^2 \cdot i$$

Our model captures the varying magnitudes of emergency events through the use of *prototypes* that define the radius and intensity of the secondary process. In practice, the specific values and the rate of occurrence of the prototypes would be determined by the incidents and call patterns observed at the PSAPs being studied. The parameters of the secondary process are then drawn from a normal distribution that aligns with the selected prototype.

Given a prototype radius's mean (μ_r) and standard deviation (σ_r), the cluster radius is defined as

$$R \sim \mathcal{N}(\mu_r, \sigma_r^2)$$

where R follows a normal distribution. Likewise, from a prototype intensity's mean (μ_i) and standard deviation (σ_i), a normally distributed intensity is defined as

$$I \sim \mathcal{N}(\mu_i, \sigma_i^2)$$

5.1.4 Time interval between each call

We address the distribution of calls in time as follows. If T is a random variable representing the interval between each emergency call, then

$$T \sim \text{Exp}(\sigma_t)$$

where T follows an exponential distribution with a rate parameter of σ_t. Subsequently, the cumulative sum of T $(t_1, t_1 + t_2, ..., t_1 + t_2 + t_3 + ... + t_n)$ is added to the timestamp of the emergency incident to compute the times of occurrence of each of the n emergency calls.

5.1.5 Location of emergency calls

For the spatial domain, the emergency calls are uniformly distributed within a circle. To achieve this, polar coordinates consisting of an angle (θ) and a radius (ρ) are drawn from uniform distributions. The distribution of angle values (θ) is proportional to 2π; whereas, the area of a circle is proportional to the square of its radius ($Area = \pi \cdot r^2$). Therefore, if U and V are two independent uniform random variables on the interval (0, 1), the polar coordinates of points uniformly located on a circle of radius r are given by

$$(\rho, \theta) = (r \cdot \sqrt{U}, 2\pi \cdot V)$$

The polar to Cartesian coordinates conversion can then be done via the following trigonometric functions:

$$x = \rho \cdot \cos(\theta)$$

$$y = \rho \cdot \sin(\theta)$$

The complete algorithm for synthesizing the secondary events for the cluster point process is presented in algorithm 1.

5.2 Call Abandonment

A customer who calls when all call takers are busy is placed in a waiting queue (see Section 4.2); those who run out of patience before their call gets answered, hang up. This

Algorithm 1 Generating secondary events

for all prim_event **in** prim_events **do**
    Select μ_r, σ_r, μ_i, and σ_i from a prototype
    Sample R from a normal distribution $\mathcal{N}(\mu_r, \sigma_r^2)$
    Sample I from a normal distribution $\mathcal{N}(\mu_i, \sigma_i^2)$
    Calculate $n = \pi \cdot R^2 \cdot I$
    for all n secondary points **do**
        Sample T from exponential distribution with rate σ_t
        Sample U from a uniform distribution between 0 and 1
        Sample V from a uniform distribution between 0 and 1
        Calculate $(\rho, \theta) = (r\sqrt{U}, 2\pi \cdot V)$
        Calculate $(x, y) = (\rho \cdot \cos(\theta), \rho \cdot \sin(\theta))$
        Calculate $t = \text{prim\_event}(t) + \mathit{cumulative\_sum}(T)$
    end for
end for
return $\mathit{sec\_events}(t, x, y)$

process is known as abandonment. Palm [38] developed Palm/Erlang-A that incorporates call abandonment into the Erlang-C equation. In the Erlang-A model, each call arrival is associated with an exponentially distributed *patience time* with mean θ^{-1} and gets an *offered waiting time* in the queue. If the waiting time exceeds the patience time, the call is considered abandoned [28].

The importance of incorporating call abandonment into a model is illustrated by Gans et al. [17] and Mandelbaum and Zeltyn [28]. Gans et al. [17] used the Erlang A queuing model to demonstrate that, during heavy traffic, even a small fraction of abandoned calls can greatly affect system performance. Likewise, Mandelbaum and Zeltyn [28] showed that in a heavily loaded call center, 3.1% abandonment reduces the *average speed of answer* from 8.8 seconds to 3.7 seconds. This reduction occurs because abandonment effectively decreases the workload precisely when it is most needed.

In our models, we adopt an exponentially distributed patience time. However, estimating the abandonment rate θ presents the difficulty of not being directly observable. The patience time is only measurable for customers who abandon the queue before being served, whereas the waiting time represents only a lower bound of the patience time for those who receive service. Here, we implement the practical method suggested by Mandelbaum and Zeltyn [28] for estimating patience time, which is based on the relationship between the average wait in the queue ($E[W]$) and the fraction of customers that abandon it ($P\{Ab\}$). From this, θ is estimated as

$$\theta = \frac{P\{Ab\}}{E[W]} = \frac{AbandonmentFraction}{AverageWait}$$

5.3 Redial

If all call takers and trunks (phone lines) are busy when a customer initiates a call, they receive a busy signal and the call is dropped. NENA uses the Extended Erlang-B equation for the estimation of Offered Call Volume when determining the staffing needs of PSAPs. The advantage of the Extended Erlang-B equation is that it allows for the consideration of

call redials, which NENA estimates to be at least 85% of the time [39]. It is also expected that the redialing occurs immediately because this is the case of a person experiencing an emergency, in desperate need for their call to go through.

In this thesis, the redialing decision is represented by a discrete random variable (Rd) that follows a Bernoulli distribution with a probability of success of $P = 0.85$.

$$Rd \sim Bernoulli(0.85)$$

based on NENA's staffing policy document [39].

The redialing probability could be fitted to the data of a given region if one obtains call data that uniquely identifies the callers. However, obtaining call data with unique identifiers from the 911 system was unfeasible for this work due to technical obstacles, privacy considerations, and legal compliance issues.

5.4 Service Time

In Emergency Services Communication Systems (ESCS), the service time represents the interval between the initial call answer and the dispatch of a responder to the emergency incident. Here, the service time is assumed to follow an exponential distribution and is predetermined at the outset of the simulation. The average service time used in the simulations, implemented for this thesis, was derived from call data acquired from the Seattle Police Department PSAP of September 2020. This dataset encompassed 41,217 records, showcasing an average service time of 205 seconds with a standard deviation of 222 seconds.

5.5 Responder Dispatch and Response Time

The dispatch of responders starts with selecting the most suitable one for a given emergency event. This involves consideration of responder availability, expertise, and proximity to the event. In the modeling framework developed for this thesis, responder units are dispatched from different stations, with priority given to the unit stationed closest to the emergency incident.

To enhance simulation realism, emergency calls, and therefore responses, are categorized into one of three types:

1. Law: Incidents requiring a police response.

2. EMS: Incidents requiring an Emergency Medical Service response.

3. Fire: Incidents requiring a fire unit response.

The implementation methodology adopted in this thesis uses incident-type ratios derived from the 2022 report of the North East King County Regional Public Safety Communication Agency (NORCOM). According to their reporting, incidents can be classified as 64% Law-related and 36% distributed between EMS and Fire [37].

The time during which a response unit remains occupied, following a successful dispatch, includes both the driving time to the incident location and the on-site time dedicated to addressing the emergency. The driving time is determined by calculating the time it takes for a responder to cover the Euclidean distance between the dispatching center and the incident location. Although this approach represents a simplified assumption, it serves as an initial approximation in the models, providing a foundation for potential refinements in future research.

The duration of on-site time is assumed to be exponentially distributed, with an average of 20 minutes considered reasonable based on prior research studies. Notably, a study in five regions of the Western Cape reported an overall average on-scene time of 27.55 minutes [51], and Emergency Medical personnel in Mississippi reported an average on-scene time of 14.67 minutes [11]. Moreover, a study by Vincent-Lambert and Mottershaw [52] highlighted that medical providers should ideally spend no more than 20 minutes at the emergency scene.

5.6 Dealing with Outliers in Exponential Distributions

Drawing random variables from probability distribution introduces the potential of including undesirable outliers. In normal distributions, those outliers can be removed by excluding

values that fall outside of the range of three standard deviations above and below the median. However, there is no standard method for calculating outliers in exponential distributions.

The method used in this thesis for removing outliers consists of calculating lower and upper *fences*, where values falling outside their defined range are excluded [44]. The *lower fence* of an exponential distribution is effectively 0 thus only the *upper fence* needs to be computed. Tukey's boxplots present a practical method for estimating the *upper fence*, which is calculated as 1.5 times the interquartile range (IQR) above the third quartile ($Q3$). For an exponential distribution with a rate parameter (λ), those values can be estimated as follows:

$$UF = Q3 + 1.5 * IQR$$

$$Q3 = \ln(4)/\lambda$$

$$IQR = \ln(3)/\lambda$$

The application of Tukey's boxplot method, as outlined in this work, offers a robust approach to mitigate the introduction of outliers when generating random variables from exponential distributions. This ensures that simulations maintain a realistic and representative distribution of data in stochastic processes.

Chapter 6

RESULTS

This chapter is focused on verifying the software implementation and evaluating the effectiveness of the models developed in this thesis for capturing the fundamental behavior of an Emergency Services Communication System (ESCS). It includes the process of assessing that the implementation matches the described mathematical models (Chapter 4 and Chapter 5) and an illustrative application that describes the calibration of parameters using real data, presents metrics for analyzing simulation results, and demonstrates a behavior that aligns with the real world (Section 6.2).

6.1 Models Verification

Verification is the process of checking that the software implementation meets the product specifications. In this thesis, the specifications are given by the definition and design of the ESCS models. The verification process consisted of two activities. First, unit tests were used to verify the behavior of individual components, independently from other parts, such as the implementation of a *GraphManager*, *Inputmanager*, and *CircularQueue*. Second, a 911 simulation small enough to be traced by hand was set up and executed, and the output was confirmed to be as expected.

The unit tests were developed using the GoogleTest framework which is incorporated into Graphitti's development process, where changes to the codebase must pass a set of unit tests before being incorporated into the main branch of the repository.

Similarly to unit tests, Graphitti also incorporates regression tests where the outputs of multiple test simulations are verified against their *known-good outputs* before changes can be incorporated into the codebase. Accordingly, a test simulation of a small ESCS was developed

Type	# Nodes	Description
PSAP	1	4 Servers and 5 Trunks (Phone Lines)
Responder	9	3 EMS, 4 Law, and 2 Fire nodes
Caller Region	1	With grid that matches the Seattle PD service boundaries

Table 6.1: Small 911 simulation network, used to verify the ESCS models implementation.

to serve as a regression test for the NG911 models. The simulation setup is presented in Table 6.1, consisting of 1 PSAP, 1 Caller Region, and 9 Responder nodes.

A list of 20 calls was created to test the correct implementation of the models described in this thesis. The complete list of calls with the values for time of occurrence, duration, location (x,y), type, patience time, and on-site time is presented in Appendix A. The simulation was set to run for 30 minutes of simulation time to guarantee that all 20 calls were recorded given that calls are only recorded after being processed, either successfully served, dropped, or abandoned. The values for the calls' time of occurrence and duration were selected to ensure that the simulation yields 1 abandoned call, and 1 dropped call (which is immediately redialed) for a PSAP with 4 servers and 5 trunks (Table 6.1). Whereas, patience and on-site time were both drawn from exponential distributions with parameters as described in Section 6.2. In addition, the x and y locations of calls and responder nodes were selected to ensure that each node receives at least one dispatch order during the simulation.

The hand-traced expected results for this small 911 simulation and its comparison to the simulation output is presented in Appendix A. After tracing by hand and confirming the correctness of the simulation output, this simulation became a regression test for Graphitti thus providing a solid foundation for future developers.

Type	# Nodes	Description
PSAP	1	6 Servers and 16 Trunks
Responder	73	34 EMS, 5 Law, and 34 Fire
Caller Region	1	Seattle PD Service boundaries

Table 6.2: Seattle PD ESCS network, used to demonstrate the ESCS models applicability.

6.2 Illustrative Application

To demonstrate the applicability of our graph-based simulation framework, we developed a graph representation of the ESCS network of King County, Washington, extracted from a Geographic Information System (GIS) dataset (provided by the Washington State 911 Coordination Office) comprised of five layers that map the boundaries of PSAPs, police, fire, and EMS services. For a complete simulation, we focused on the Seattle Police Department (PD) ESCS network composed of 1 PSAP, 34 Fire Stations (also providing EMS services), and 5 Police stations (shown in Table 6.2). The Seattle PD PSAP, now known as Community Assisted Response and Engagement (CARE), is the largest of the 12 PSAPs in King County, serving about 10% of all 911 calls.

A month-long dataset composed of 41,217 calls with a call rate of 57.25 calls/hr was used to parameterize the discrete event queuing model for CARE. The dataset revealed a maximum hourly call rate of 137, an abandonment rate of 9.42%, a average wait time of 4.7 seconds, and call durations ranging from 4 to 2016 seconds (204 seconds average). Using Erlang equations, estimations suggest a need for 6 call takers, considering a 10-second caller waiting tolerance (Erlang C), and 16 trunk lines to manage a 1% probability of blocking during peak times with an additional 10 seconds of service time for post-processing.

Table 6.3 presents the values of the parameters used for the simulation runs of this illustrative application. We conducted simulations using one month of synthetic call data generated through the cluster point process algorithm discussed in Section 5.1. These sim-

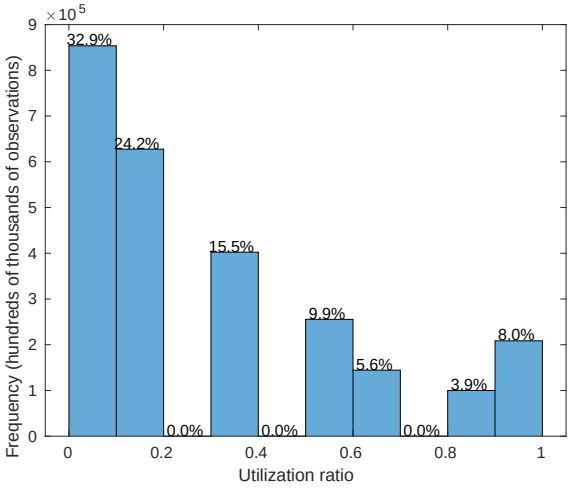
Parameter	Value
Simulation Length	1 month
# Servers	6
# Trunks	16
Redial Probability	0.85
Average Patience Time	49.36 sec
Min Service Time	4 sec
Average Service Time	204 sec
Average On-Scene Time	20 min
Call Arrival Rates	45.6 to 84.2 calls/hr in 10% increments

Table 6.3: Parameters used for the illustrative application simulation runs. The cluster point process model described in Section 5.1 is used to generate the various call arrival rates.

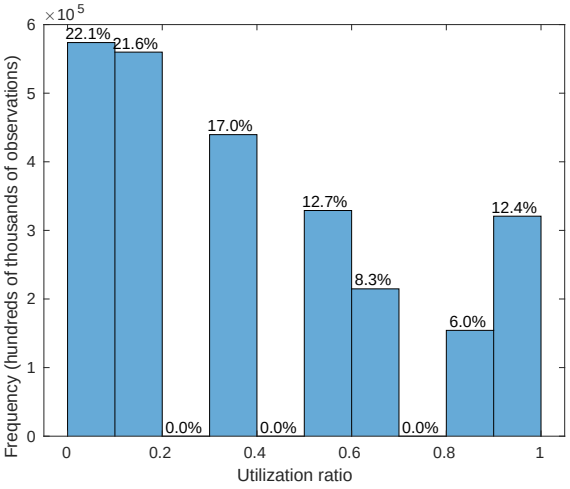
ulations used call arrival rates ranging from 45.6 to 84.2 calls/hr in 10% increments, which were simulated by varying the rate of the primary process of a cluster point process. Other parameters were kept consistent throughout the simulations. The redial probability used was 85% with an average patience time of 49.36 seconds (see Section 5.2). Additionally, the average on-scene duration was set at 20 minutes (see Section 5.5), whereas the minimum and average service times were set to 4 and 204 seconds, respectively.

The selected metrics for analyzing simulation results were customer waiting time and system utilization. System utilization is defined as the fraction of call-takers or responder units busy at any given time. The abandonment rate, being a by-product of system saturation or over-utilization, is also considered in our analysis.

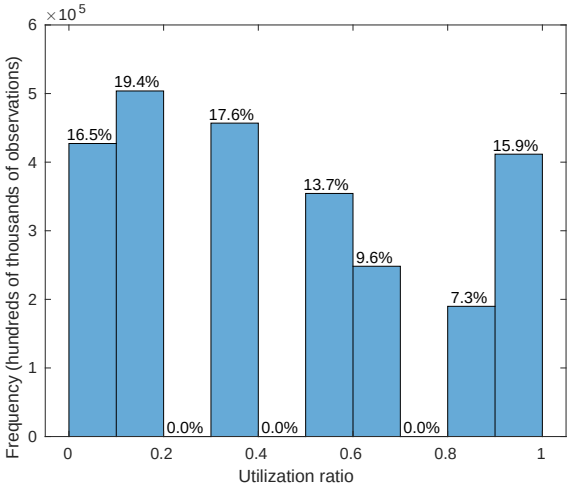
Utilization histograms in Figure 6.1 are presented for the different arrival rates. At 45.6 calls/hr, the system surpasses 80% utilization merely 11.9% of the time, coinciding with an average waiting time of 6.8 seconds. As the arrival rate grows to 84.2 calls/hr, elevating



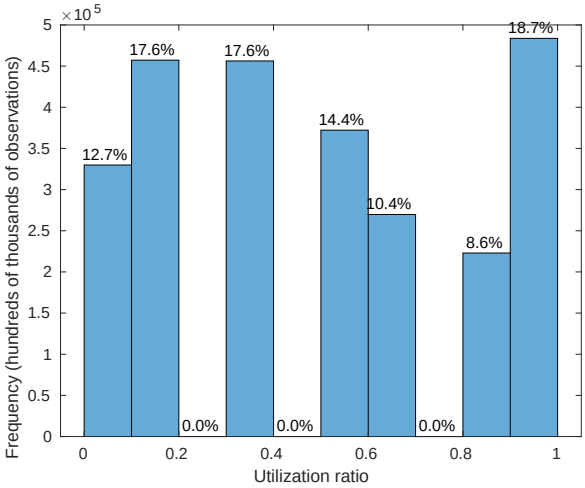
(a) Arrival rate = 45.6 calls/hr



(b) Arrival rate = 63 calls/hr



(c) Arrival rate = 76 calls/hr



(d) Arrival rate = 84.2 calls/hr

Figure 6.1: Call arrival rate impact on system utilization

# Servers	Avg. Wait (sec.)	Abandonment (%)
6	9.27	27
7	7.8	22
8	6.6	18
9	5.5	15
10	4.6	12

Table 6.4: Impact of the number of servers on wait time and abandonment rate. All simulations used 76 calls/hr and 11 for the call arrival rate and number of trunks, respectively.

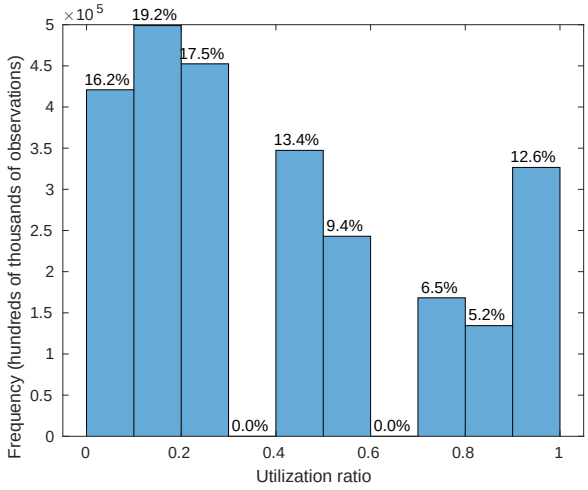
utilization to 27% above 80%, the average waiting time increases moderately to 9.4 seconds. However, at this threshold the abandonment rate triples from 9 to 27.3%.

When the call arrival rate reached 76 calls/hr the system started showing signs of saturation. At this arrival rate, the average wait time was only 9.27 seconds but the abandonment rate had already climbed to 27%, showing the impact of the abandonment process on keeping the waiting time relatively stable.

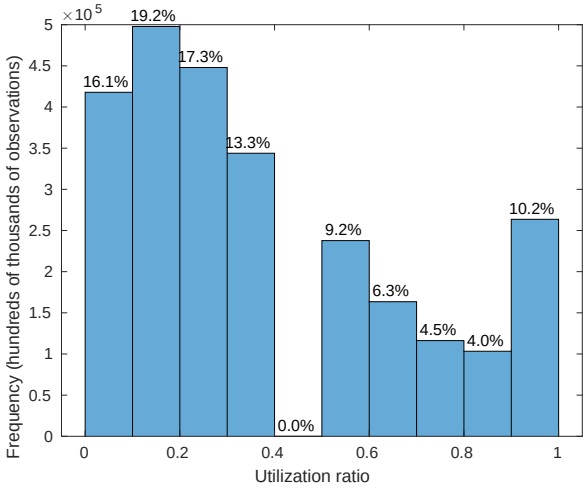
Considering that a PSAP with 6 servers and 16 trunks displayed a high abandonment rate of 27%, under the 76 calls/hr arrival rate. We subsequently searched for the number of servers needed to bring the abandonment rate to a more reasonable level. Therefore, we ran simulations with 7, 8, 9, and 10 servers, keeping all other parameters unvaried.

Table 6.4 presents the average wait time (in seconds) and the abandonment rate for simulations ran with 6, 7, 8, 9, and 10 servers, keeping all other parameters the same. With only adding 3 more servers, the abandonment rate improved by 45% (from 27 to 15%). Pairing the abandonment rate improvement with the system utilization histogram presented in Figure 6.2, one notices that the 9-servers simulation also displays the best system utilization above 80% (11.5%).

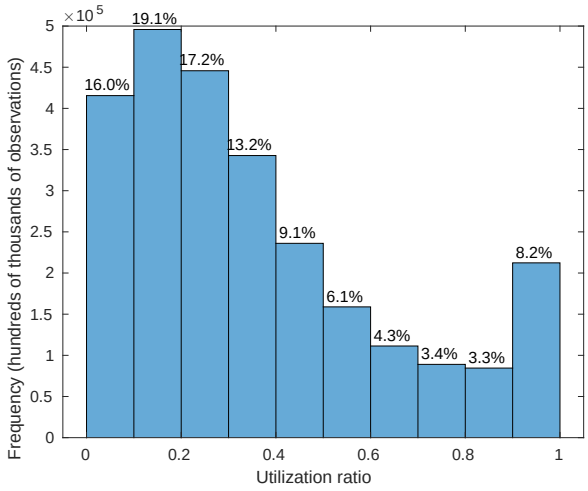
Note that the number of trunks for all the simulations was kept at 16. Therefore, the



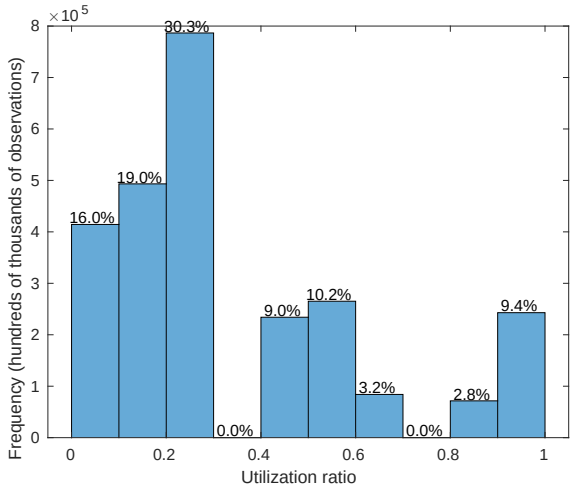
(a) Servers = 7



(b) Servers = 8

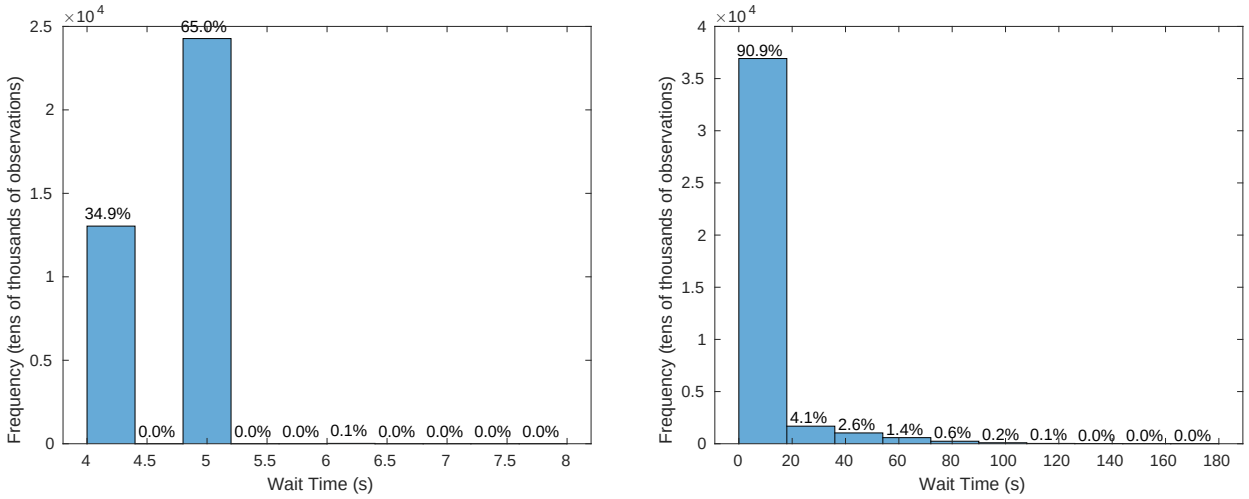


(c) Servers = 9



(d) Servers = 10

Figure 6.2: Impact of the number of servers on system utilization. All simulations used 76 calls/hr and 11 for the call arrival rate and number of trunks, respectively.



(a) Real world

(b) Simulation

Figure 6.3: Wait time histograms of the real and simulated Seattle PD PSAP.

waiting queue shortens as we add more servers to the system. The impact of the size of the waiting queue on utilization is that as the system reaches saturation levels calls are more likely to be dropped but because of the high redial probability (0.85), these dropped calls come back into the system most of the time. We must also consider that the cluster point process, used to generate the call arrival process, creates clusters of calls closed in time that might momentarily saturate the system. Consequently, the system utilization is kept high once the system reaches saturation levels until the arrival rates goes down, which explains the spikes above 90% utilization in the histograms of both Figure 6.1 and Figure 6.2.

Certain metrics stored in simulations may not be feasible or practical to measure in real-world systems, such as system utilization. Although these metrics enable a more comprehensive performance evaluation, their absence in real system data complicates the assessment of how accurately the models reflect real-world behavior. One consistently recorded metric in ESCSes is the *wait time,* providing a basis for comparing real-world and simulated Seattle PD PSAP behavior. Figure 6.3 illustrates histograms of wait times for both the real

and simulated Seattle PD PSAP outputs. In the real system, wait times were consistently between 4 and 5 seconds, with none exceeding 8 seconds, while in the simulated system, wait times reached up to 180 seconds, with 86.3% being less than 6 seconds and 90.9% less than 20 seconds. This disparity in wait time distributions likely stems from differences in staffing policies: the Seattle PD PSAP staffing is designed to maintain very low wait times, whereas the simulation uses minimum staffing based on the one-month average call arrival rate estimated using the Erlang C equation. The challenge of determining an appropriate staffing policy underscores the need to obtain comprehensive data, including information on caller and responder dynamics as well as staffing patterns, from the 911 PSAPs.

This illustrative application is meant to showcase the capabilities of the presented models for simulation and analysis of ESCSes. As such, the analysis of the responder nodes in the network are beyond the scope of this work; although, the same analysis of utilization, wait time, and abandonment rate can be applied.

6.2.1 Runtime scalability

The runtime performance of the one-month Seattle PD PSAP simulation was measured using a 14-cores Intel i7-12700H 2.3 GHz processor. On average, it took 47 seconds to execute a one-month simulation for a single PSAP network. Extrapolating this runtime to the 78 PSAPs comprising the Washington State ESCS network, would require approximately 61 minutes to complete a one-month simulation of the statewide 911 network.

A future GPU parallel implementation of these ESCS models has the potential to significantly reduce the simulation's runtime. However, as noted by Shi et al. [41], the size of large-scale graphs might present a challenge for GPU processing due to memory limitations. We argue that ESCS networks are known and not expected to reach the size of billions of vertices and edges, such as the Twitter and Youtube datasets used by Shi et al. [41]. Considering that the size of ESCS networks is not expected to exceed GPU memory, the improvement in runtime scalability of the GPU would enable researchers to execute longer simulations involving larger networks in a matter of minutes rather than hours.

Chapter 7

CONCLUSION

The primary objective of this thesis was to develop simulation models serving as the basis for Emergency Services Communication Systems (ESCSes) simulations. Achieving this goal involved identifying and examining the components of the 911 system, designing an architectural framework, and choosing suitable mathematical models to represent its diverse stochastic processes. As a result, the most significant contribution of this thesis is the architectural framework and model abstractions that define various aspects of ESCSes.

7.1 Architectural Framework

This thesis introduced a detailed architectural framework that models the components of an ESCS as a graph-based system. This graph forms the foundation for a set of Communicating Finite State Machines (CFSM) that operate within a discrete event queuing system.

The architectural framework that emerges from the model's implementation enhances Graphitti's capabilities by enabling the use of arbitrary graphs and the exploration of diverse call arrival processes through external files. Additionally, it simplifies the analysis of each individual vertex behavior, laying the foundation for a future parallel implementation where each vertex can be treated as a separate thread of execution.

7.2 Mathematical Models

Another novel contribution of this thesis was the modeling of call arrivals as the realization of a cluster point process that captures burstiness in calls and their proximity to the emergency events. Beyond this novel contribution, this work also investigated aspects of ESCS that are frequently overlooked by other models, notably addressing the issues of call abandonments

and redial.

7.3 Retrospective

Although the architectural framework and mathematical models presented here effectively capture the essential aspects of an ESCS, they include necessary simplifications. This is primarily due to project scope limitations and data acquisition challenges. Acquiring call and incident data from 911 introduced complexities, including technical obstacles, privacy considerations, and legal compliance issues. Consequently, simulating a comprehensive real-world 911 network, like King County, was unfeasible within the designated time frame.

7.4 Future Work

In the context of future research, several key areas merit exploration and development to enhance the ESCS simulation models presented in this thesis.

Firstly, there is a significant opportunity to implement a parallel GPU version of the existing CPU model. This adaptation can substantially improve the performance and scalability of ESCS simulations, allowing for more complex and extensive analyses.

Another promising avenue for research is the exploration of a non-homogeneous Poisson process as the primary process within a cluster point process, such as a cyclic non-homogeneous Poisson process [26]. Law and Kelton [25] present another approach to modeling a non-stationary Poisson process, known as thinning, that might be worth exploring. Modeling the primary process as a non-stationary Poisson process may provide a more accurate representation of the dynamic nature of emergency call arrivals in ESCS scenarios. Another implication of a non-stationary Poisson process is that the PSAP staffing policies would also need to change to have more servers during busier times, approach that is recommended by Russell and Mazeau [39].

To enhance the realism of ESCS simulation, the development of a more precise mathematical model for simulating responder travel time and on-site time is a critical future endeavor. Similarly, the implementation of mobility for emergency response units is another

promising direction. More refined models would contribute to a more faithful representation of real-world scenarios, thus offering a more adaptable and realistic representation of ESCS operations.

The current simulation setup involves saving results to an output file for subsequent analysis, which is effective when assessing various parameter values. However, certain scenarios demand real-time monitoring of simulation states through a Graphical User Interface (GUI). For instance, a crucial step in cybersecurity tabletop exercises is assessing the impact of a mitigation strategy during a cyber-attack simulation. The future implementation of a live monitoring framework within the simulation can significantly enhance its utility in tabletop exercises, ensuring real-time assessment, and bolstering its effectiveness.

Lastly, a framework could be developed for generating synthetic, parameterized graphs. This framework would facilitate further research into various graph characteristics at different scales and might aid in exploring various graph families.

Further improvements and refinements in the presented models would enhance the development of more realistic ESCS simulations. Nonetheless, this work lays a robust foundation for future research in this area.

BIBLIOGRAPHY

[1] S. Anwar, M. Yaseen, and S. A. Mahmood. Higher order gibbs point process modeling of 2005-kashmir earthquakes. *Modeling Earth Systems and Environment*, 9(1):1335–1347, oct 2022. doi: 10.1007/s40808-022-01554-9.

[2] B. K. Bal, W. L. Shi, S.-H. S. Huang, and O. Gnawali. Towards a content-based defense against text ddos in 9–1-1 emergency systems. In *2018 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–6. IEEE, 2018.

[3] R. Barnes and B. Rosen. 911 for the 21st century. *IEEE Spectrum*, 51(4):58–64, 2014. ISSN 1939-9340. doi: 10.1109/MSPEC.2014.6776307. Conference Name: IEEE Spectrum.

[4] R. Barnes, A. Cooper, and H. Tschofenig. Technical considerations for next-generation 911. 2011.

[5] D. Brand and P. Zafiropoulo. On communicating finite-state machines. *Association for Computing Machinery (ACM)*, 30(2):323–342, 1983. ISSN 0004-5411. Place: New York, NY Publisher: Association for Computing Machinery.

[6] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall. GraphML progress report structural layer proposal. In P. Mutzel, M. Jünger, and S. Leipert, editors, *Graph Drawing*, Lecture Notes in Computer Science, pages 501–512. Springer, 2002. ISBN 978-3-540-45848-7. doi: 10.1007/3-540-45848-4_59.

[7] F. C. Commission. Legal and Regulatory Framework for Next Generation 911 Services. Report to Congress and Recommendations 112-96, Federal Communications Commission, Feb. 2013.

[8] D. Corral-De-Witt, E. V. Carrera, J. A. Matamoros-Vargas, S. Munoz-Romero, J. L. Rojo-Alvarez, and K. Tepe. From e-911 to NG-911: Overview and challenges in ecuador. *IEEE Access*, 6:42578–42591, 2018. doi: 10.1109/access.2018.2858751. Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[9] P. S. P. Cowpertwait, C. G. Kilsby, and P. E. O'Connell. A space-time neyman-scott model of rainfall: Empirical analysis of extremes. *Water Resources Research*, 38(8): 6–1–6–14, aug 2002. doi: 10.1029/2001wr000709.

[10] Dajer, Antonio J., Lopez, Fred A., Baker, Todd, Toscano, Joseph D., Scalea, Thomas M., Todd, Knox H., Rao, Rama B., Rund, Douglas, Kman, Nicholas E., and Slovis, Corey M. Disaster Preparedness 10 Years After 9/11 The Experts Weigh In. *Emergency Medicine*, Sept. 2011.

[11] G. David and T. Brachet. Retention, learning by doing, and performance in emergency medical services. *Health Services Research*, 44(3):902–925, June 2009. ISSN 1475-6773. doi: 10.1111/j.1475-6773.2009.00953.x.

[12] T. I. Dayharsh, T. J. Yung, D. K. Hunter, and S. C. Ivy. Update on the national emergency number 911. *IEEE Transactions on Vehicular Technology*, 28(4):292–297, 1979. ISSN 19399359. doi: 10.1109/T-VT.1979.23804.

[13] B. Dearstyne. The FDNY on 9/11: Information and decision making in crisis. *Government Information Quarterly*, 24(1):29–46, Jan. 2007. ISSN 0740-624X. doi: 10.1016/j.giq.2006.03.004. URL `https://www.sciencedirect.com/science/article/pii/S0740624X06000505`.

[14] U. Desai, S. Alagesan, A. Goulart, and W. Magnussen. Performance of secure sip and lost signaling in a next generation 9–1–1 testbed. In *2012 IEEE International Workshop Technical Committee on Communications Quality and Reliability (CQR)*, pages 1–6. IEEE, 2012.

[15] T. Doumi, M. F. Dolan, S. Tatesh, A. Casati, G. Tsirtsis, K. Anchan, and D. Flore. LTE for public safety networks. *IEEE Communications Magazine*, 51(2):106–112, 2013. ISSN 01636804. doi: 10.1109/MCOM.2013.6461193.

[16] J. W. Farnham. Disaster and emergency communications prior to computers/internet: a review. *Critical Care*, 10(1):207, 2005. ISSN 1364-8535. doi: 10.1186/cc3944. URL `https://doi.org/10.1186/cc3944`.

[17] N. Gans, G. Koole, and A. Mandelbaum. Telephone call centers: Tutorial, review, and research prospects. *Manufacturing & Service Operations Management*, 5(2):79–141, 2003.

[18] GEXF Working Group. GEXF file format, 2009. URL `https://gexf.net/`.

[19] L. Gómez, R. Budelli, R. Saa, M. Stiber, and J. P. Segundo. Pooled spike trains of correlated presynaptic inputs as realizations of cluster point processes. *bc*, 92(2):110–127, Feb. 2005.

[20] K. Gustavsson. *Stochastic Modeling and Management of an Emergency Call Center*. PhD thesis, Mid Sweden University, 2018.

[21] K. Gustavsson, P. L'Ecuyer, and L. Olsson. Modeling bursts in the arrival process to an emergency call center. 2018.

[22] Industry Council for Emergency Response Technologies. History of 911: And what it means for the future of emergency communications. Technical report, 2016. URL `https://perma.cc/YL97-9J9C`.

[23] J. Jalving, Y. Cao, and V. M. Zavala. Graph-based modeling and simulation of complex systems. *Computers & Chemical Engineering*, 125:134–154, 2019. ISSN 00981354. doi: 10.1016/j.compchemeng.2019.03.009. URL `https://linkinghub.elsevier.com/retrieve/pii/S0098135418312687`.

[24] D. LaRocque. GraphSON format, 2014. URL https://github.com/thinkaurelius/faunus/wiki/GraphSON-Format.

[25] A. M. Law and W. D. Kelton. *Simulation modeling and analysis*. McGraw-Hill series in industrial engineering and management science. McGraw-Hill, 3rd ed edition, 2000. ISBN 978-0-07-059292-6.

[26] S. Lee, J. R. Wilson, and M. M. Crawford. Modeling and simulation of a nonhomogeneous poisson process having cyclic behavior. *Communications in Statistics - Simulation and Computation*, 20(2-3):777–809, Jan. 1991. ISSN 0361-0918. doi: 10.1080/03610919108812984. URL https://doi.org/10.1080/03610919108812984.

[27] F. Liberal, J. O. Fajardo, C. Lumbreras, and W. Kampichler. European NG112 crossroads: Toward a new emergency communications framework. *IEEE Communications Magazine*, 55(1):132–138, 2017. ISSN 1558-1896. doi: 10.1109/MCOM.2017.1600301CM. Conference Name: IEEE Communications Magazine.

[28] A. Mandelbaum and S. Zeltyn. *Service Engineering in Action: The Palm/Erlang-A Queue, with Applications to Call Centers*, pages 17–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-29860-1. doi: 10.1007/978-3-540-29860-1_2. URL https://doi.org/10.1007/978-3-540-29860-1_2.

[29] G. Marin and D.-P. Pop. The european emergency number 112. *Journal of Information Systems & Operations Management*, 3(1):173–184, 2009. URL https://ideas.repec.org//a/rau/jisomg/v3y2009i1p173-184.html. Publisher: Romanian-American University.

[30] E. K. Markakis, A. Lykourgiotis, I. Politis, A. Dagiuklas, Y. Rebahi, and E. Pallis. EMYNOS: Next generation emergency communication. *IEEE Communications Magazine*, 55(1):139–145, 2017. doi: 10.1109/mcom.2017.1600284cm. Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[31] J. Martinez, christopherdokeefe, M. Stiber, J. Brown, V. Gandhi, M. Sorvik, T. Salvatore, D. Kamath, P. Pal (PoojaPal2021), K. Dukart, V. Verma, R. Sarcevic, J. Kim, AlexNeary, xiang_coding, C. McIntosh, S. KUMAR, rashwini21, uw-theo, L. Zee, S. Wong ("Phoenix"), Andrzej-Dawiec, BenYang2002, and Surendran. UWB-Biocomputing/Graphitti: It's 2017 in 2023, June 2023. URL `https://doi.org/10.5281/zenodo.8087930`.

[32] Y. Mirsky and M. Guri. Ddos attacks on 9-1-1 emergency services. *IEEE Transactions on Dependable and Secure Computing*, 18(6):2767–2786, 2021. ISSN 1545-5971.

[33] National Emergency Number Association (NENA) and others. NENA standard for 9-1-1 call processing, 2020.

[34] National Emergency Number Association (NENA) 911 Core Services Committee and i3 Architecture Working Group. NENA i3 standard for next generation 9-1-1. Technical report, National Emergency Number Association, 2021.

[35] S. R. Neusteter, M. Mapolski, M. Khogali, and M. O'toole. The 911 call processing system: A review of the literature as it relates to policing. *Vera Institute Of Justice*, pages 240–254, 2019. doi: 10.4324/9781003027508-21.

[36] J. Neyman and E. L. Scott. Statistical approach to problems of cosmology. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(1):1–43, 1958. ISSN 00359246. URL `http://www.jstor.org/stable/2983905`.

[37] North East King County Regional Public Safety Communications (NORCOM) Agency. NORCOM 2022 Annual Report. Technical report, NORCOM, 2022. URL `https://www.norcom.org/wp-content/uploads/2023/04/NORCOM-2022-Annual-Report.pdf`.

[38] R. C. A. Palm. *Research on telephone traffic carried by full availability groups*. Tele, 1957.

[39] R. Russell and D. Mazeau. PSAP staffing guidelines report. Technical Report NENA-REF-001-2003, National Emergency Number Association, 2003. URL https://www.nena.org/resource/resmgr/standards/NENA-REF-001-2003_PSAP_Staff.pdf.

[40] R. F. Serfozo. Chapter 1 point processes. In *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, pages 1–93. Elsevier, 1990. doi: https://doi.org/10.1016/S0927-0507(05)80165-3. URL https://www.sciencedirect.com/science/article/pii/S0927050705801653.

[41] X. Shi, Z. Zheng, Y. Zhou, H. Jin, L. He, B. Liu, and Q.-S. Hua. Graph Processing on GPUs: A Survey. *ACM Computing Surveys*, 50(6):81:1–81:35, Jan. 2018. ISSN 0360-0300. doi: 10.1145/3128571. URL https://dl.acm.org/doi/10.1145/3128571.

[42] J. Shimkus. H.R.438 - 106th Congress (1999-2000): Wireless Communications and Public Safety Act of 1999, Feb. 1999. URL https://www.congress.gov/bill/106th-congress/house-bill/438. Archive Location: 1999-02-02.

[43] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. *Boost Graph Library, The: User Guide and Reference Manual*. Addison-Wesley Professional., 1st edition, 2001. ISBN 978-0-13-265183-7. URL https://www.informit.com/store/boost-graph-library-user-guide-and-reference-manual-9780132651837.

[44] C. H. Sim, F. F. Gan, and T. C. Chang. Outlier Labeling With Boxplot Procedures. *Journal of the American Statistical Association*, 100(470):642–652, June 2005. ISSN 0162-1459. doi: 10.1198/016214504000001466. URL https://doi.org/10.1198/016214504000001466.

[45] R. Simon and S. Teperman. The world trade center attack: Lessons for disaster management. *Critical Care*, 5(6):318–320, 2001. ISSN 1364-8535. doi: 10.1186/cc1060. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC137379/.

[46] J. M. Stevens, T. C. Webster, and B. Stipak. Response Time: Role in Assessing Police Performance. *Public Productivity Review*, 4(3):210–230, 1980. ISSN 0361-6681. doi: 10.2307/3379854. URL https://www.jstor.org/stable/3379854. Publisher: Taylor & Francis, Ltd.

[47] M. Stiber, F. Kawasaki, D. Davis, H. Asuncion, J. Lee, and D. Boyer. BrainGrid+Workbench: High-performance/high-quality neural simulation. In *Proc. International Joint Conference on Neural Networks*, Anchorage, Alaska, May 2017.

[48] T. Stratmann and D. C. Thomas. Dial 911 for murder: The impact of emergency response time on homicides. *SSRN Electronic Journal*, 2016. doi: 10.2139/SSRN.2877078. URL https://papers.ssrn.com/abstract=2877078. Publisher: Elsevier BV.

[49] Task Force on Optimal PSAP Architecture (TFOPA). Task force on optimal psap architecture adopted final report. Technical Report DA-16-179, Federal Communications Commission, Jan. 2016. URL https://www.fcc.gov/document/fcc-releases-tfopa-final-report/report.

[50] The GraphML Team. The GraphML file format, 2019. URL http://graphml.graphdrawing.org/index.html.

[51] M. Vanderschuren and D. McKune. Emergency care facility access in rural areas within the golden hour?: Western Cape case study. *International Journal of Health Geographics*, 14(1):5, Jan. 2015. ISSN 1476-072X. doi: 10.1186/1476-072X-14-5. URL https://doi.org/10.1186/1476-072X-14-5.

[52] C. Vincent-Lambert and T. Mottershaw. Views of emergency care providers about factors that extend on-scene time intervals. *African Journal of Emergency Medicine*, 8(1):1, Mar. 2018. doi: 10.1016/j.afjem.2017.08.003. URL https://www.ncbi.nlm.

nih.gov/pmc/articles/PMC6223598/. Publisher: African Federation for Emergency Medicine.

[53] G. Wainer. *Discrete-Event Modeling and Simulation: A Practitioner's Approach*. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, 2017. ISBN 9781420053371. URL https://books.google.com/books?id=E0T5dmTrG7EC.

[54] M. Xue and S. Roy. Cyber-physical queueing-network model for risk management in next-generation emergency response systems, 2021.

Appendix A

VERIFICATION OF SMALL 911 SIMULATION OUTPUT

Table A.1 contains the list of 20 calls used in the small 911 test simulation described in Chapter 6. The calls' time of occurrence, duration, and location (x, y) were tailored to test the correct implementation of the models described in this thesis. Whereas, patience and on-site time were both drawn from exponential distributions as described in Section 6.2. The expected results from this simulation, traced by hand, are presented in Table A.3 and can be compared against the actual simulation results in Listing A.1.

To compare the expected and actual simulation results, the PSAP call log was extracted into Table A.2. Here, one can verify that the begin, answer, and end times for all the calls are correct. The answer and end times for the call occurring at 73 seconds of simulation time are zero because the call is abandoned due to the wait time being longer than the customer's patience time. To verify that the correct responder is dispatched, one can verify that the end time for a given call is present in the list of beginning times for the corresponding responder vertex.

time	duration	x	y	type	patience	on_site_time
34	230	-122.374820944356	47.6483954827697	EMS	61	3142
37	169	-122.403648760113	47.5583378861826	Fire	3	2032
42	327	-122.385348869295	47.5153247164363	Fire	8	782
47	165	-122.275688766409	47.6790423255801	EMS	9	627
73	262	-122.367015875811	47.5193248411992	EMS	50	1890
130	242	-122.327331103854	47.6570871634272	Law	150	1105
324	209	-122.428429392232	47.5929827626697	Fire	5	603
388	45	-122.377464667327	47.7111396737191	Law	66	182
401	110	-122.450311894902	47.7041426158929	EMS	45	53
435	54	-122.384978533577	47.5859768754596	EMS	139	770
490	259	-122.33562990965	47.6488009024455	Law	77	2142
541	350	-122.375038778789	47.5274803656548	Law	86	510
671	389	-122.438467646611	47.5944297368486	EMS	60	637
900	81	-122.294911773769	47.6088629766148	Fire	57	90
960	53	-122.296485522688	47.6177632023237	Law	113	824
1009	638	-122.355965347492	47.6773160725642	Law	38	81
1106	230	-122.447215747839	47.6093549620811	EMS	16	3378
1110	64	-122.317137239523	47.5453578234151	Fire	8	139
1143	241	-122.393782737017	47.4843154029632	Fire	47	633
1155	180	-122.432330936241	47.6829697374505	Fire	20	967

Table A.1: Small 911 simulation input calls. The time, duration, patience, and on-site time are all in seconds.

Begin Time	Answer Time	End Time
37	38	207
73	0	0
47	48	213
34	35	265
42	43	370
388	389	434
130	208	450
435	436	490
401	402	512
324	325	534
490	491	750
541	542	892
900	901	982
960	961	1014
671	672	1061
1110	1111	1175
1106	1107	1337
1155	1175	1355
1143	1144	1385
1009	1010	1648

Table A.2: Seattle PD PSAP call log for the small 911 simulation output, used to verify the ESCS model implementation.

		Servers – end time						Call Log		
Call	**Time**	**1**	**2**	**3**	**4**	**Queue**	**Abandoned?**	**Answered**	**End Time**	**Responder**
34	35	265				[]				
37	38		207			[]				
42	43			370		[]				
47	48				213	[]				
73	74					[73]				
	125					[]	Yes	0	0	
130	131					[130]				
	207		Finish			[130]		38	207	#8 (Fire)
	208		450			[]				
	213				Finish	[]		48	213	#3 (EMS)
	265	Finish				[]		35	265	#1 (EMS)
324	325				534	[]				
	370			Finish		[]		43	370	#8 (Fire)
388	389	434				[]				
401	402			512		[]				
	434	Finish				[]		389	434	#4 (Law)
435	436	490				[]				
	450		Finish			[]		208	450	#5 (Law)
	490	Finish				[]		436	490	#2 (EMS)

Call	**Time**	**1**	**2**	**3**	**4**	**Queue**	**Abandoned?**	**Answered**	**End Time**	**Responder**
490	491		750							
	512			Finish		[]		402	512	#1 (EMS)
	534				Finish	[]		325	534	#9 (FIRE)
541	542	892								
671	672			1061						
	750		Finish					491	750	#5 (Law)
	892	Finish						542	892	#7 (Law)
900	901				982					
960	961		1014							
	982				Finish	[]		901	982	#8 (Fire)
1009	1010	1648								
	1014		Finish			[]		961	1014	#6 (Law)
	1061			Finish		[]		672	1061	#2 (EMS)
1106	1107				1337					
1110	1111		1175							
1143	1144			1385						
1155	1156					[1155]				
	1175		Finish			[1155]		1111	1175	#8 (Fire)
			1355			[]				
	1337				Finish	[]		1107	1337	#2 (EMS)

Call	**Time**	**1**	**2**	**3**	**4**	**Queue**	**Abandoned?**	**Answered**	**End Time**	**Responder**
	1355		Finish			[]		1175	1355	#9 (FIRE)
	1385			Finish		[]		1144	1385	#8 (Fire)
	1648	Finish				[]		1010	1648	#4 (Law)

Table A.3: Hand-traced expected results of the small 911 simulation used to verify the ESCS model implementation.

Listing A.1: Small 911 simulation output

```
<?xml version="1.0" standalone="no"?>
<!-- State output file for the 911 systems modeling-->
<SimState>
<Matrix name="xloc" type="complete" rows="1" columns="11" multiplier="1.0"
   >
   -122.329 -122.355 -122.38 -122.284 -122.335 -122.337 -122.317 -122.362
      -122.309 -122.355 0
</Matrix>
<Matrix name="yloc" type="complete" rows="1" columns="11" multiplier="1.0"
   >
   47.6043 47.6821 47.5608 47.6686 47.7029 47.6161 47.6152 47.536 47.5719
      47.6821 0
</Matrix>
<Matrix name="vertexTypesPreEvent" type="complete" rows="1" columns="11"
   multiplier="1.0">
   4 5 5 5 7 7 7 7 6 6 3
</Matrix>
<Matrix name="vertexTypesPostEvent" type="complete" rows="1" columns="11"
   multiplier="1.0">
   4 5 5 5 7 7 7 7 6 6 3
</Matrix>
   <Matrix name="numTrunks">
   5 10 10 10 10 10 5 10 10 10 0
   </Matrix>
   <Matrix name="numServers">
   4 5 3 3 3 3 3 4 5 3 0
   </Matrix>
   <Matrix name="droppedCalls">
   1 0 0 0 0 0 0 0 0 0 0
   </Matrix>
   <Matrix name="receivedCalls">
   21 2 3 1 2 2 1 1 5 2 0
```

```
</Matrix>
<Matrix name="BeginTimeHistory">
   <vertex id="0">
   37 73 47 34 42 388 130 435 401 324 490 541 900 960 671 1110 1106
      1155 1143 1009
   </vertex>
   <vertex id="1">
   265 512
   </vertex>
   <vertex id="2">
   490 1061 1337
   </vertex>
   <vertex id="3">
   213
   </vertex>
   <vertex id="4">
   434
   </vertex>
   <vertex id="5">
   450 750
   </vertex>
   <vertex id="6">
   1014
   </vertex>
   <vertex id="7">
   892
   </vertex>
   <vertex id="8">
   207 370 982 1175 1385
   </vertex>
   <vertex id="9">
   534 1355
   </vertex>
```

```
</Matrix>
<Matrix name="AnswerTimeHistory">
   <vertex id="0">
   38 0 48 35 43 389 208 436 402 325 491 542 901 961 672 1111 1107 1175
        1144 1010
   </vertex>
   <vertex id="1">
   266 513
   </vertex>
   <vertex id="2">
   491 1062 1338
   </vertex>
   <vertex id="3">
   214
   </vertex>
   <vertex id="4">
   435
   </vertex>
   <vertex id="5">
   451 751
   </vertex>
   <vertex id="6">
   1015
   </vertex>
   <vertex id="7">
   893
   </vertex>
   <vertex id="8">
   208 371 983 1176 1386
   </vertex>
   <vertex id="9">
   535 1356
   </vertex>
```

```
</Matrix>
<Matrix name="EndTimeHistory">
   <vertex id="0">
   207 0 213 265 370 434 450 490 512 534 750 892 982 1014 1061 1175
      1337 1355 1385 1648
   </vertex>
   <vertex id="1">
   496 623
   </vertex>
   <vertex id="2">
   545 1451 1568
   </vertex>
   <vertex id="3">
   379
   </vertex>
   <vertex id="4">
   480
   </vertex>
   <vertex id="5">
   693 1010
   </vertex>
   <vertex id="6">
   1068
   </vertex>
   <vertex id="7">
   1243
   </vertex>
   <vertex id="8">
   377 698 1064 1240 1627
   </vertex>
   <vertex id="9">
   744 1536
   </vertex>
```

```
</Matrix>
<Matrix name="WasAbandonedHistory">
   <vertex id="0">
   0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
   </vertex>
   <vertex id="1">
   0 0
   </vertex>
   <vertex id="2">
   0 0 0
   </vertex>
   <vertex id="3">
   0
   </vertex>
   <vertex id="4">
   0
   </vertex>
   <vertex id="5">
   0 0
   </vertex>
   <vertex id="6">
   0
   </vertex>
   <vertex id="7">
   0
   </vertex>
   <vertex id="8">
   0 0 0 0 0
   </vertex>
   <vertex id="9">
   0 0
   </vertex>
</Matrix>
```

```
...
<Matrix name="Tsim" type="complete" rows="1" columns="1" multiplier="1.0">
   900
</Matrix>
<Matrix name="simulationEndTime" type="complete" rows="1" columns="1"
   multiplier="1.0">
   1800
</Matrix>
</SimState>
```