
**School of Computer Science and Engineering**

**Faculty of Engineering**

**The University of New South Wales**

# Generating Realistic Wi-Fi Traffic for Honeypots

by

Ricard McGill Grace

Thesis submitted as a requirement for the degree of

Bachelor of Engineering in Software Engineering

Submitted: August 2020

Student ID: z5113357

Supervisor: Salil Kanhere

Co-Supervisor: David Liebowitz

# Abstract

The arms race in cyber security is speeding up, with more attacks becoming automated and thus deployed more widely. Many security technologies have kept up and are at least semi-automated. This includes security mechanisms such as firewalls or intrusion detection systems or antivirus applications with automatic sample submission or heuristic analysis to combat day 0 threats. Honeypots however have seen little headway in the development of automated generation techniques.

To combat this, we propose the use of temporal point process learning architectures to automate the synthesis of low-level Wi-Fi traffic. This will provide an easy and cost-effective method for producing high quality Wi-Fi Honeypots. We will cover the use of three temporal modelling techniques; Hawkes processes, intensity free distributions, and Markov chains followed by an analysis of each techniques' performance including potential benefits or shortcomings.

# Acknowledgements

This research has been supported by the Cyber Security Research Centre Limited whose activities are partially funded by the Australian Government's Cooperative Research Centres Program

# Abbreviations

**NN** Neural Network

**RNN** Recurrent Neural Network

**LSTM** Long-Short Term Memory

**TPP** Temporal Point Process

**HMM** Hidden Markov Model

**RHS** Right Hand Side

**LHS** Left Hand Side

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Overview

The following report details the prior research, experiments performed and subsequent result analysis in pursuit of the goal of generating an effective method for autonomously generating wireless honeypots.

Chapter 2 contains background details to the project, including a review of relevant literature, and background information on the project, including goals and aims.

Chapter 3 contains a full breakdown of each experiment performed. Each experiment contains background information/details of the setup, the results of performing the experiment and a brief analysis of what the results mean in the context of producing an effective model.

Chapter 4 holds a broader analysis of the effectiveness of the applied models in comparison to each other, challenges encountered during the project and possible future work to extend the project.

## 1.2 Problem Statement

Currently, a hot topic in cyber security is wireless security, especially with the abundance of Wi-Fi enabled devices, and lately the introduction of IoT systems, and the lack of security mechanisms built into these devices. This makes wireless devices a prime target for attacks, with regular users and their devices typically unaware of when an attack takes place. One method to detect these attacks is through the use of honeypots.

In the field of honeypots, the only effective method is to generate one is manually, which is both very time consuming and may produce something of sub-par quality, whether this be a lack of realism or the presence of sensitive information originally trying to be protected. Ideally, the process of honeypot creation should be automated to aid in the development and widespread deployment of these defences, aiding our ability to detect fraudulent behaviour amongst legitimate ones. By employing temporal modelling techniques, it may be possible to create an automated but effective system for the development of wireless honeypots, allowing their widespread use.

## 1.3 Project Aims

This project will involve investigating the use of various temporal modelling techniques as a method for generating convincing Wi-Fi data acceptable for use in a honeypot. This requires for a model to reliably produce sequences of low level network packets for an arbitrary length of time while containing similar properties to the packet stream it is attempting to mimic in an effort to prevent arousing the suspicion of attackers to a fake. The performance of these models will be compared to each other and to a Markov model as a baseline, due to its exceptional performance in modelling sequences for decades. Achieving this requires building, training and simulating event sequences using three of the reviewed modelling techniques, Hawkes processes, Intensity free temporal modelling and Markov models.

# Chapter 2

# Background

## 2.1 Literature Review

This section details the modelling techniques explored which are designed for learning sequences of events. While not all these techniques are featured in this project (only Hawkes processes, intensity free and Markov models are used), the rest are covered here for completeness.

### 2.1.1 Temporal Point Processes

#### Overview of TPPs

Point processes consider a sequence of points with respect to a second distribution (examples include $\mathbb{R}$, $\mathbb{N}$ and $\mathbb{R}^2$). Temporal point processes are a sub-category of point processes, involving an ordered sequence of events occurring over time [Ras18]. Examples include social media posts on an emerging topic, transactions on a securities exchange or earthquakes and their aftershocks. Instead of considering these processes as a sequence of events, they are typically considered as a sequence of inter-event times. Doing this allows for uniquely modelling temporal processes with a density function of

inter-arrivals, enabling further manipulation [Ras18].

A key theory in TPPs is evolutionarity [Ras18], future events are dependent on the events preceding them. While this is not true for some forms of TPPs, such as Poisson processes, where each event is entirely independent, many temporal processes follow this structure, forming chains of events. Dependent events are modelled with a conditional intensity function, an intensity function with variable intensity based on the current history of events. This allows for future events to be more (or less) probable given the event history [Ras18]. For example, sending a personally addressed email to someone will likely increase the chance of receiving a response back.

### Marks

A temporal process can be augmented with additional information related to the occurrence of an event. This information, called marks, can be helpful for building a more accurate model of the process. It allows for distinguishing between different kinds of events and finding deeper patterns in an event sequence [Ras18]. From a technical perspective, marks augment the conditional density function to be conditional on both the event history and mark history.

### Simulation

Once a model of a temporal process is built, the process can then be simulated, generating new events using the learned conditional density function. Simulation of a process usually occurs for one of two reasons: As a tool to evaluate the effectiveness of the learned model in capturing the original process and forecasting new points in the event sequence [Ras18]. Two methods exist for simulating a general temporal point process. Finding the inverse density function or Ogata's modified thinning method.

With the inverse method, the process is simulated until a given time $T$ is reached with new points being drawn from the inverse integrated conditional density function. An

issue with the inverse method is the density function may not be invertible, in which case iterative approximation techniques must be applied to evaluate it, which can be slow [Ras18].

Ogata's thinning method considers intervals of time and attempts to generate a new event in each [Ras18][LTP15]. The event is thrown away if it meets any of the following criterion:

- The event time is outside the current time interval
- The event time is outside the maximum simulation time frame
- the chance of the event occurring at the given time with respect to the conditional intensity function is too low

If the event is kept, the start time for future intervals is set to the new event's time. The process of considering intervals continues until the start of such interval is outside the simulation time frame.

### Types of Temporal Point Processes

Three general types of TPP's exist. Renewal, self-exciting and self-correcting.

Renewal processes consider each event in the process to be independent. The process has no history and all events are identically distributed. This means the conditional intensity of the process is a function of time from only the last event seen [Ras18]. This behaviour can be seen in processes such as Markov chains and Wold processes.

Self-exciting processes gain increased intensity with each new event arrival, decaying over time at a parameterisable rate. This can lead to quickly rising intensity levels resulting in clusters of events forming [Ras18].

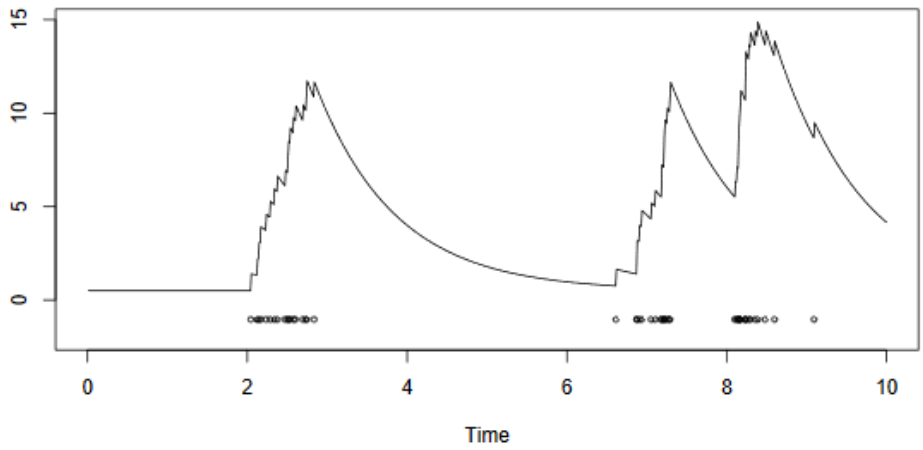



Figure 2.1: Example Self-Exciting Process [Ras18]. Top: intensity of the process. Bottom: event sequence.

Self-Correcting processes have their intensity reduced with each new event arrival, slowly increasing in intensity with time. This results in a more regular spread of events that will quickly try and correct back when deviating from the regular pattern [Ras18]

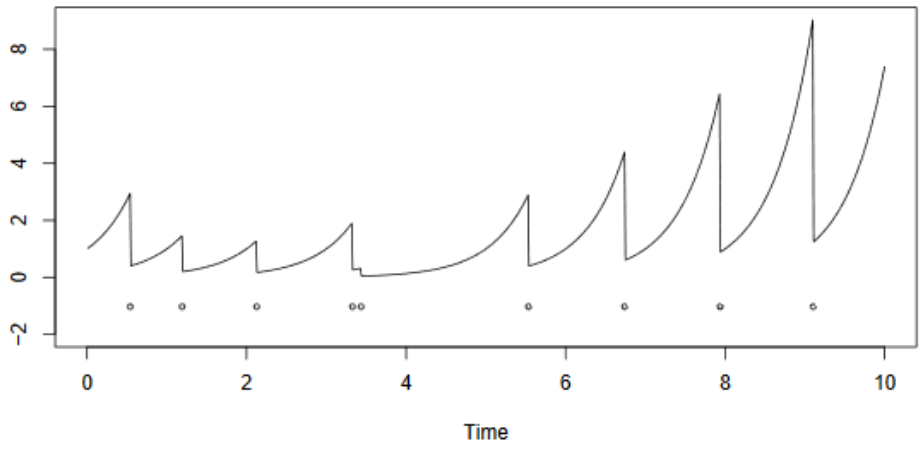



Figure 2.2: Example Self-Correcting Process [Ras18]. Top: intensity of the process. Bottom: event sequence.

### 2.1.2 Markov Chains and Markov Models

Markov chains are a statistical method of representing a sequence of events. Given a set of input states (previous events), it returns the distribution of output states (predicted events) based on what it has seen before (Figure 2.3). They are structurally simple, quick to implement and train in comparison to neural networks. Training usually consists of adding a new N-tuple of states to an internal database or incrementing the number of occurrences if already present [PK11]. Markov models are effective for modeling discrete sequencing problems, especially text, and are capable of producing reasonably coherent sequences. However, it can lack the ability to capture longer dependencies, limiting the quality of the data it can produce as it may create 'out of context' data.

Markov chains were in use before neural networks and are still in use today. It is important for newer technologies to advance upon their predecessors to further technological advancement. As a result, the Markov chain will be used within this project as a baseline model in contrast to newer, more advanced methods.

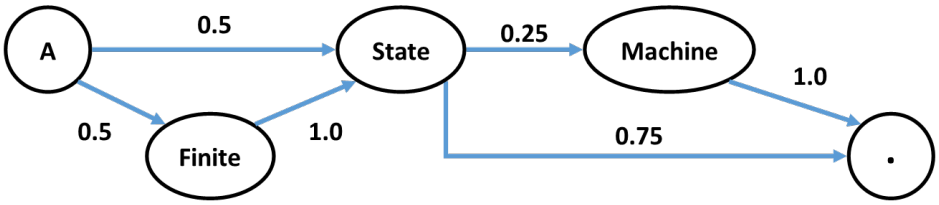



Figure 2.3: Example Finite Markov Chain

#### Architecture

Implementation of a Markov model involves a 'database' where training data will be stored and a look up method to retrieve the distribution of outcomes for a given input. The performance of the model is heavily dependant on the data structure used for the database, affecting space efficiency and lookup rate differently.

Array - Extremely high space requirements as a slot needs to be available for each

possible combination of input states as an array's size is fixed. In most cases, very space inefficient as many input states will never be seen. Very quick lookup O(c).

List - Only uses as much memory as states trained on as it expands as needed. Lookup is O(n) in the number of unique states trained on, which can be slow for larger data sets.

Sorted Array - Similar to a list implementation, only using memory as needed but can be searched significantly faster using binary search O(log n) [PK11].

Hash Table - High space requirements to create an efficient hash but utilises its space more efficiently than an array and can be expanded in size as needed (at a performance cost). Typically, very quick lookup O(c) with worst case O(n) in the number of states [PK11].

Tree - Only uses as much memory as states seen and expands as needed. Maximum depth of the tree is the length of the input. Thus, lookup performance is O(n) in the input length, much faster than a list. This method also allows for 'variable length n-grams' where certain branches of the tree are deeper than others [NW96]. It allows for dynamically adjusting the input length to minimise split distributions.

### 2.1.3 Hawkes Processes

Hawkes processes are a type of self-exciting temporal process using an intensity function to model the probability of spawning new events at each timestep.

The intensity function can be decomposed into parts, the baseline intensity, and the excitation function [RLMX17][LTP15]. The baseline is a constant level of excitement designed to trigger new chains of events. This is designed to simulate the arrival of new events from external to the process (not self-excited) [RLMX17]. The excitation function determines how additionally excited the processes gets due to previous events. This function can theoretically be anything, if for any positive time it has a positive (or 0) level of excitement (negative levels would depress the excitement, instead turning

it into a self-correcting process). A common choice of excitation function is the exponential function due to its simplicity, only requiring optimisation of two parameters (excitement and decay) [LTP15].

The process can also be viewed as an immigrant – offspring tree [LTP15], modelling the event chain caused by an immigrant event. This gives rise to the notion of a branching factor [LTP15] Larger branching factors signify processes that are more excitable, since on average they produce more points per immigrant (the branching factor is then the average number of new events generated by a parent event). This branching factor must be less than 1 to produce a reasonable process, otherwise the process will never terminate (each event produces at least 1 event resulting in an unbounded number of child events) [LTP15].

New Hawkes process based sequences can be generated using the thinning algorithm [RLMX17][LTP15] described in Section 2.1.1

### 2.1.4 Perceptrons, Neural Networks and Activation Functions

The perceptron is a simple transformation unit. Given numerical inputs, input scaling factors (weights) and a bias factor, it aggregates and produces an output using a step function [Nor05] (Figure 2.5). The perceptron will produce an output of 0 if the sum is $< 0$ and 1 if the sum is $\geq 0$ (Figure 2.4). The unit 'learns' by adjusting the input weights and bias slowly towards a correct answer [Nor05].

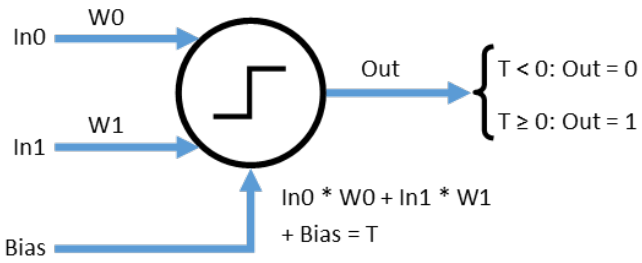



Figure 2.4: Perceptron Architecture

The function that transforms the input aggregate into the output is called the activation

function, in this case, the activation function is a step function [Nor05] (Figure 2.5). For more complex tasks, the perceptron can be extended with more inputs (increasing the dimensionality of the unit) or with different activation functions. Other commonly used activation functions include the sigmoid logistic curve and the hyperbolic tangent curve [SSL17] (Figure 2.5). These activation functions change the output of the unit to be a range between $0 \rightarrow 1$ and $-1 \rightarrow 1$ respectively; useful for more complex networks.

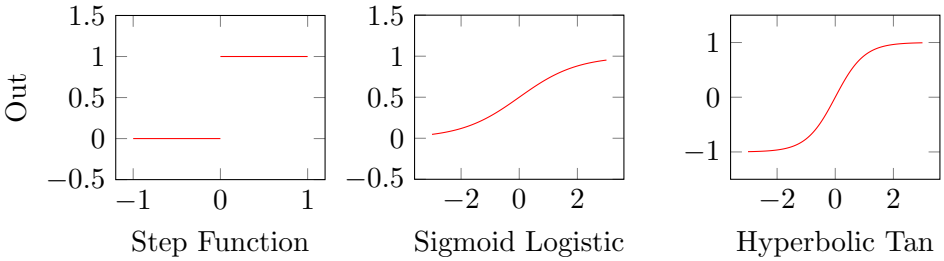



Figure 2.5: Typical Activation Functions

The perceptron is limited however, only being able to model linear functions, but by layering units (with the outputs of previous units becoming inputs for future ones) this can be overcome [Nor05]. The layering of units allows approximation of more complex functions and is the basis of neural networks. In more complex networks (like the RNN or LSTM), the perceptron may be altered or replaced, but the principle of using multiple input-output transformation units is universal [RIGE16].

### 2.1.5 Recurrent Neural Networks

Vanilla NNs do not understand the concept of a sequence, instead being ideal for classification or transformation of individual objects. RNNs are vanilla NNs but also pass state from previous executions to the next. Thus, the 'recurrence' of the network comes from applying the same vanilla NN to each input and chaining them together by passing hidden state from the previous execution to the next one [Che16][Gra13]. Chaining allows for keeping track of information over numerous iterations and for outputs to be calculated based on context from previous iterations (the chained state) [Che16].

### Training Through Backpropagation

The model learns through the updating of network weights using 'backpropagation through time'. It entails computing the error of the output and propagating it through the network backwards through network layers and iterations of the neural network. The method uses the gradient of the error function to compute the direction in which weights should to be moved to improve the model [Che16][Gra13]. A requirement of backpropagation is the activation function used within each node must be differentiable. Since the step function, used in typical perceptrons, is not continuously differentiable, another continuously differentiable function is used for RNNs (such as sigmoid or hyperbolic tangent (Figure 2.5)).

### Architecture

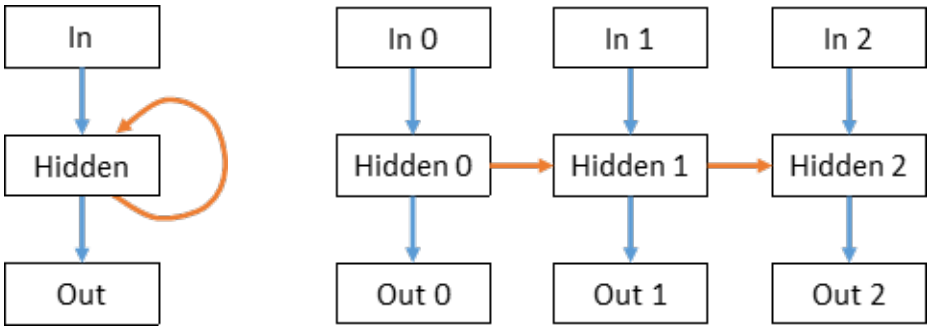
A typical RNN consists of 3 main layers; input, hidden and output [Che16][Gra13] (Figure 2.6). All nodes present in the RNN must use a differentiable function as their activation function to enable back propagation.

The input layer contains one input for each element in the sequence to compute. The length of the input is the number of recurrences in the model and thus the length of 'context' available to the model. Longer input sequences tend to yield better results and reduce the accumulation of error, but the increase in components leads to greater computation complexity.

The hidden layer can be a single layer or multiple layers of nodes, depending on the complexity of problem to solve. This layer is where the main computation/transformation of inputs is performed. LSTM cells can replace the nodes in the hidden layer to improve memory retention and improve the quality of results (See LSTM for more details). Output from the hidden layer both goes to the final layer and hidden layer in the next iteration.

The output layer usually performs some aggregation of data from the hidden layer

(the output layer is typically smaller than the hidden layer) to output a vector of values [Gra13]. In language modeling this vector would contain the probability of each character occurring next in a sequence.




(a) RNN Wrapped (b) RNN Unwrapped

Figure 2.6: Architecture of a RNN

#### Shortcomings of RNNs

Standard RNNs in theory can compute sequences of infinite length with infinite accuracy, the only requirement is an infinite amount of resources [Gra13]. Practically, RNN's struggle with 'remembering' past inputs for significant periods of time [Gra13]. Each time hidden state is chained from one iteration to the next, a weight update is applied to it. With enough weight updates, information chained from many iterations ago is lost due to the Vanishing Gradient Problem (Explained in 2.1.6). This is a big source of error in standard RNNs since, the more it forgets about the past, the quicker error accumulates in future iterations, causing the model to hallucinate incorrect results.

### 2.1.6 LSTM

LSTM cells are memory cells designed for use in neural networks with the sole purpose of improving the memory capabilities of a network. The usual nodes in a network (such as an RNN) can be replaced with layers of LSTM cells or have extra layers of LSTM cells added to it. Development of LSTMs was in response to the vanishing gradient problem plaguing RNN's performing longer sequencing tasks [GSK+16].

### Vanishing Gradient Problem

As values are passed through a network, passing through multiple nodes in the process (such as the hidden values in an RNN), the weight scaling factor is applied to these values on multiple occasions. With weights other than 1, the values quickly explode towards infinity or diminish towards 0. As a result, the state passed between nodes to try and 'remember' past events is skewed to the point of being useless [Che16]. This is the vanishing gradient problem.

A bad solution is to fix all the weights in a network, setting them to 1, so they do not diminish in any direction. However, constant weight updates are how a network learns. Fixing the weights of a network renders it ineffective for long sequence manipulation.

Development of LSTM was the solution, managing the explosion of weights through careful self regulation of input and output gates [Che16]. This means error across cells is constant and memory is improved during processing of longer sequences (specifically the processes saving, accessing and forgetting past information).

As a result, LSTM enabled RNNs are regularly the go to method for sequence manipulation only to now be contested by the introduction of the transformer.

### Architecture

A typical LSTM cell consists of 6 main parts [Che16][Gra13](Figure 2.7):

Memory Cell - Where information inside the cell is stored. Self explanatory.

Cell Input - Receives weighted inputs and passes them through a hyperbolic tangent activation function before merging with data already in the memory cell.

Cell Output - Receives data stored in the memory cell, passes it through a hyperbolic tangent activation function and serves as the output of the memory cell. The output is then weighted for use in the next layer.

Input Gate - Uses information from the previous layer to determine how to augment the data from the 'Cell Input' before reaching the memory cell. Its purpose is to control information flow into the 'Memory Cell'. This passes through a sigmoid activation function before augmenting the input.

Output Gate - Similar to the input gate except it augments the 'Cell Output' before it exits the cell. This controls information flow out of the LSTM.

Forget Gate - This gate determines the memory retention of the cell. It interfaces with the 'Memory Cell' (after passing through a sigmoid activation function) to make it forget information as desired.

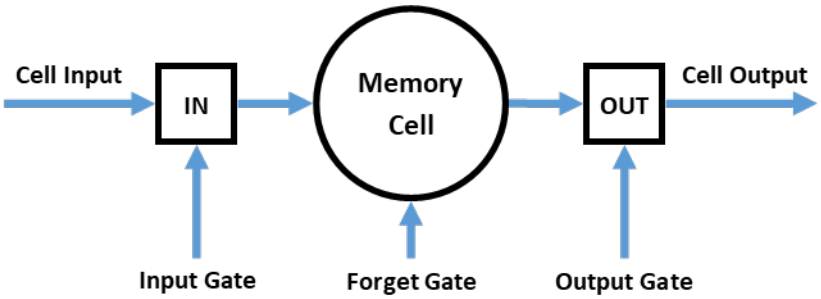



Figure 2.7: LSTM Internal Architecture

### 2.1.7 Intensity Free Temporal Modelling

Typical temporal modelling techniques, such as Hawkes processes, focus on modelling the conditional intensity. Doing this requires a trade off between simplicity and accuracy [SBG20]. Simpler models will typically have a closed form for computing the log-likelihood of learned parameters, but at a cost of reduced accuracy/modelling capacity. Conversely, more complex models will be more accurate in capturing the source data but lack the ability to compute the log-likelihood in closed form, requiring iterative root-finding algorithms instead [SBG20]. Creating an intensity free model aims to avoid this trade off, by directly computing the conditional distribution and is hence 'intensity free'.

To achieve this, the intensity free model proposed in [SBG20] uses a mixture of log-

normal distributions. There are three principles behind this:

- Interevent times of event sequences are always positive, so the positive nature of log-normals works well to model this without adding any extra hard constrains.
- A mixture of log-normal distributions is flexible with the potential to fit a probability density arbitrarily well (with enough components)
- Log-normal distributions and a mixture of are simple to compute, giving the model a closed form

The log-normal mixture model is also capable of being conditioned on the prior history of events (through the use of an RNN), sequence metadata (different to marks as they apply to the entire sequence, such as user id) and a learnable sequence embedding (for modelling sequences from varying distribution [SBG20]. This information is concatenated into a 'context vector' and used as a part of the process of determining the parameters of the mixture distribution. Each log-normal distribution in the mixture has three parameters:

- $w$ - The weight of this component in the mixture
- $\mu$ - The mean of this log-normal component
- $s$ - The standard deviation of this log-normal component

These parameters are computed using different affine transformations of the context vector, the parameters to the transformation being learned by the model. Once learned, new points can be simulated simply by sampling the mixture distribution.

### 2.1.8 Honeypots

Honeypots are passive security measures designed to monitor and notify when they are interacted with. They do not take action against intruders on their own, instead

requiring intervention from a third party to combat the issue (usually this is a system administrator) . This is very different from other active security measures, such as antivirus, which are capable of search and destroy. Honeypots work by being specialised tools of deception, acting as a silent trap, thus looking like a convincing target while not raising suspicion is key for their success. And by extension, the process of creation and the content inside a honeypot will change based on its intended purpose [Whi17]. The result of this is a security device that can be more effective than any other at identifying the intended targets of attacks.

When developing a honeypot, a key decision to make is the level of interactivity desired. Interactivity is the level of engagement a honeypot can demand from an attacker; something more realistic will tend to draw more attention. Obviously, higher interaction honeypots are better, but they tend to be more costly to produce. Thus a balance must be struck between cost and function, usually dictated by the goal for the honeypot

#### Low Interaction

Low interaction honeypots are low effort, low quality creations. They are quick and cheap to produce as their level of realism is lacking. Typically, this can consist of a file filled with random characters, posing as a passwords file. It is unlikely to fool anyone, but is good enough to detect simple automated data harvesters [Whi17]. Because of this, low interaction honeypots are less likely to trigger false security alerts as real users will disregard or quickly move away from these types of files [Whi17].

#### High Interaction

Capturing the attention of smarter attackers, whether that be humans or advanced data harvesters (e.g. one that scans for keywords), requires a more sophisticated honeypot. This means the honeypot must be developed with structure and coherent information (full sentences, paragraphs, a table, etc) [Whi17]. Additionally, placement of this kind of security measure is important as it will only protect when it is interacted with, and

it will only be interacted with when it does not look out of place. This include tailoring information inside the honeypot to be relevant to where it is placed (e.g. information about company inventory is unlikely to be found amongst employee records).

The benefits of a higher interaction honeypot over a low effort one include, maintaining the attention of attackers for longer, providing more information on the target of the attack and looking realistic [Whi17].

#### Automation

The infeasibility of creating high interaction honeypots is their main downside, they are costly to produce, typically requiring many human hours [Whi17], and lack the ability of automated production due to limited availability of automated tools. Current automated tools either generate random text like low interaction honeypots or use real files as input and stitches them together to form a higher interaction honeypot. An issue with the latter being real files possibly contain sensitive information, subsequently causing the honeypot to contain sensitive information, partially defeating the purpose of a honeypot.

## 2.2 Data Format

All data used for training models within the project consists of four main fields:

| | |
|---|---|
| Time | The timestamp on the packet in nanoseconds from the start of the packet capture. |
| Size | Size of the packet in bytes. |
| Client ID | A unique identifier for each client in the packet capture. All IP addresses are replaced with a unique id for privacy and ease of training for the learning models. |
| From DS | Whether the packet was from or to the given data station |

The format of data output from each model is of a similar nature except in cases

where the interarrival times of events are being modelled. In this case, the time field will consist of a non-negative time in nanoseconds representing the time between the previous event and the next one.

| ID | Time (ns) | Size (Bytes) | Client ID | From DS |
|---|---|---|---|---|
| 0 | 128447771 | 76 | 1 | 0 |
| 1 | 165731668 | 62 | 2 | 1 |
| 2 | 268232822 | 206 | 2 | 1 |
| 3 | 268234729 | 68 | 3 | 1 |
| 4 | 268868684 | 68 | 3 | 1 |

Table 2.1: Sample data in input format

The full dataset from (Table 2.1) can be found on Google Drive HERE.

## 2.3 Implementation and Training of Models

All model implementation and training will be performed in Python 3.7. The three models selected to be a part of this project are Hawkes processes, intensity free modelling and Markov models. Hawkes processes will be modelled using the Tick library for python [BBGP17]. Intensity free modelling will be performed by an intensity free implementation available on Github [SBG20]. Markov models will be trained using a hmmlearn, a hidden Markov model implementation available HERE. Training of intensity free models requires the use of a GPU. Access to a GPU was obtained through a GPU cluster provided by CSE and online GPU runtime provided by Google Colab. Both tick and hmmlearn do not require any additional hardware feature to run, running directly on the CPU.

## 2.4 Performance Metrics

Since the idea is to generate data suitable for a honeypot, it is important for any synthesised sequences to be reasonably similar to the source sequence they are trying

to mimic. To achieve this, a series of metrics will be used to assess how good a model is at mimicking, these are as follows:

- Comparing event interarrival distributions. The interarrival distribution of a TPP governs the characteristics of the resulting event sequence. If this is dissimilar to the source sequence, then a completely different sequence will be produced. The similarity of interarrival distributions will be compared in four ways:
  - General shape of the distribution. Ensure general features such as approximate intensity and rate of intensity change is similar at each point in time.
  - Comparison of major characteristics of the distribution. These include very defining features that if not present would immediately distinguish a real from a fake. An example of such a characteristic is explained in Section 3.3
  - Analysing the distribution at different time scales. While two distributions can appear similar at a larger timescale (say 10 seconds), zooming in on a distribution (10ms) may reveal different underlying distributions. This is a side effect of the level of data aggregation performed at different timescales to make it more manageable and possible to graph.
  - Comparing the mean interarrival time. The mean is a very quick way to identify if two distributions have the potential to be similar. Different means result in event sequences each containing the same number of events but span over different lengths of time. E.g. with means 1 sec and 2 sec, event sequences with 1000 events will average 1000 sec and 2000 sec in length respectively.
- Comparing event timelines resulting from an interarrival distribution. This is important when the ordering of interarrivals is not entirely random. This means that certain interarrivals can be more likely to occur based on the history of events. E.g. a short interarrival may likely result in another short interarrival. This ordering of interarrivals cannot be modelled with a interarrival distribution alone, requiring a comparison at an event timeline level.

- In specific cases, utilising statistical plots such as Q-Q plots and residual plots to determine goodness of fit.

# Chapter 3

# Research Experiments and Methodologies

This chapter contains five sections, each relating a different type of model/distribution explored. Each section gives an overview of background information into the specific experiment, the outcomes of performing it (including graphs of the synthetic distributions formed using the model trained during the experiment) and an analysis into the broader meaning of the results. The sections are generally in the order they were investigated over the course of this project.

## 3.1 Hawkes Processes Experiments

All the following Hawkes process modelling experiments involve multivariate models, where the data is split into two processes, messages to and from the client, to model interactions between the two processes. Attempts to model the behaviour of sending a message in reply to one. Only focuses on the communication between a single client and the router. Size of packets is not considered

### 3.1.1 Modelling Directly

Two experiments were performed using raw data as input; no transformations were applied apart from the multivariate split as described at the beginning of this section. These experiments trialled different ranges of intensity decay values to see what gives more reasonable results. Both experiments used an exponential kernel to model intensity. (Figure 3.1) is a graph containing the distribution of points from a portion of the source data, used in comparison to the simulated datasets generated in each experiment below.

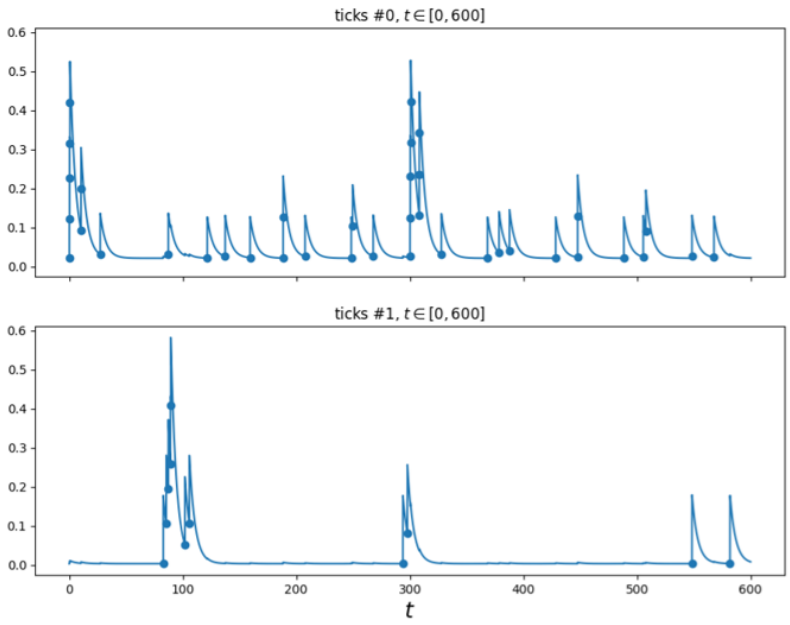



Figure 3.1: Portion of the source event sequence. Up to 600 seconds.

#### Restriction of learning parameters

In the first trial, the range of decay values the Hawkes learner could use were restricted to where the real decay value likely existed.

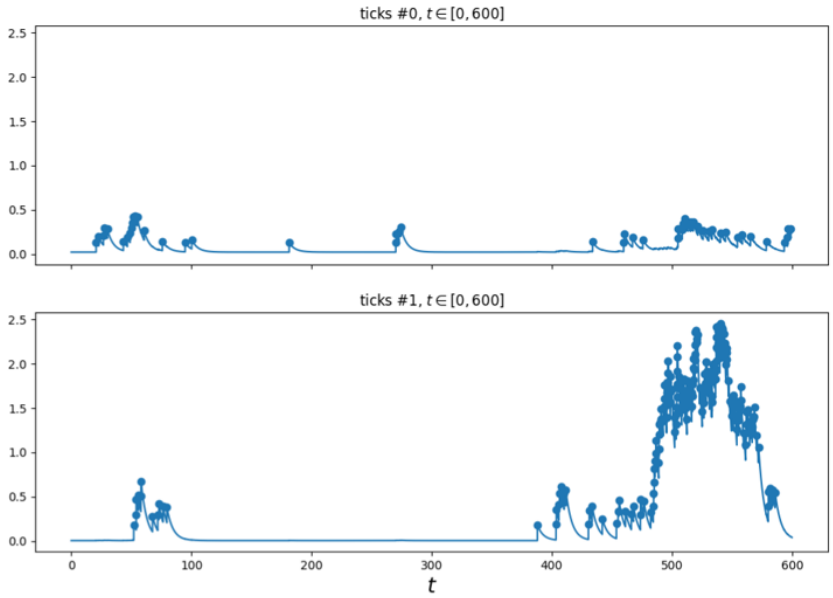



Figure 3.2: Portion of the simulated event sequence by a Hawkes process with decays restricted. Up to 600 seconds.

The graph above, (Figure 3.2), is a portion of the event sequence from a simulation of the learned model. Each point is a new event and the current event intensity is represented by the height of the line. From the source graph (Figure 3.1), the points are mainly spread out with minimal clustering occurring. This distribution is significantly different from the simulated event sequence generated using the learned model. The simulated data contains a significantly higher degree of clustering and number of points over the original (very clearly seen at the bottom right of (Figure 3.2)). This means the learned model's excitement is too high to model the source data correctly, resulting in over excitement, generating large clusters of points over a short period of time. However, it should be noted that the preferred decay value was the largest possible in the range of allowed values.

### Relaxed restriction

The second trial uses relaxed restrictions on decay values. It is possible the previous trial suffered from not being allowed the freedom of values to choose from and thus

caused a suboptimal model to be produced. By relaxing the restrictions on decay values the learner may use it may allow for a more effective model to be produced.

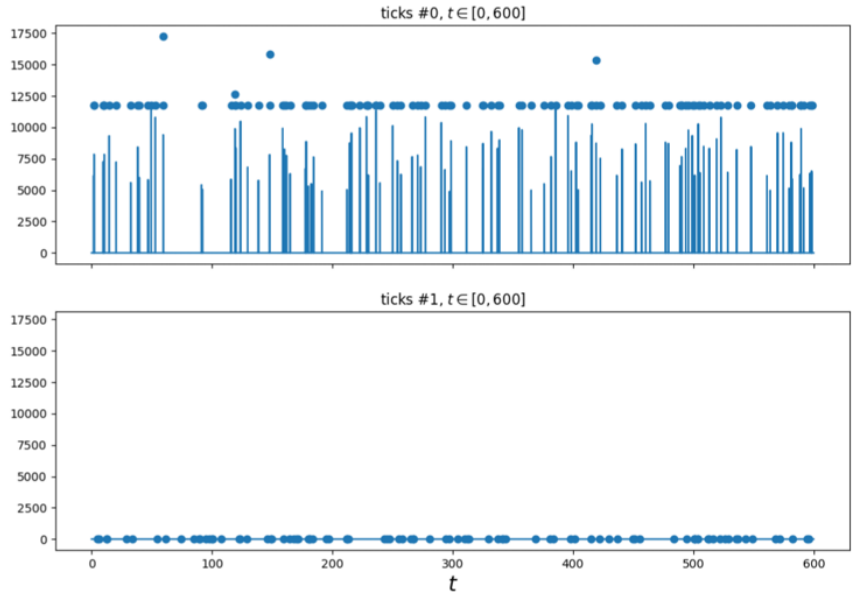



Figure 3.3: Portion of the simulated event sequence by a Hawkes process with relaxed decay range. Up to 600 seconds.

When restrictions are relaxed, a significantly different model is preferred, one with exceedingly high decay and excitement values. By analysing the simulated event sequence (Figure 3.3) we can see spikes in intensity occurring, which dissipate as quickly as they appear, each containing only a single point. This is basically random noise as the event intensity decays too quickly to spawn any dependent events and thus has no memory, effectively forming a Poisson process. In addition to this, the simulated sequence contains more points than what appears in the source sequence (Figure 3.1). In this case, it is possible the learner has overfitted the baseline intensity as almost all the generated points are only produced by the baseline.

### Result Analysis

While restricting the decay range for the learner improved results, (visible clusters would form as opposed to randomly distributed points), it clearly did not capture the

source data. The points generated were too clustered and too many. In addition to this, it should not be required to restrict the range of decay values so harshly for the learner to produce an acceptable model as this would essentially be manually fitting the data. Thus, providing the raw data directly to the Hawkes learner was ineffective at properly capturing and synthesising the source distribution

### 3.1.2 Sequence Clustering

The second approach with Hawkes processes was to model the data as many smaller sequences instead of one longer sequence. By doing this, it would reduce the amount of 'silence' present (where no events occurred) in the data. An example of the clustering can be found in (Figure 3.4). The hypothesis is that the entire data stream may not be representable by a Hawkes process, but shorter periods of activity might. The event sequence is split into multiple sub-sequences when a period of silence $T$ seconds long is encountered (the exact cutoff is determined per dataset based do on what looks good). This clustering approach splits the problem into two sub-problems, modelling when new sub-sequences occur and modelling the events within a sub-sequence.


Figure 3.4: An example of splitting into clusters at large segments of silence.

#### Modelling the Occurrence of Sub-Sequences

After splitting the sequence into sub-sequences, taking the time of the first point from each sub-sequence gives a new sequence of when each cluster starts. This new sequence is a single dimension (there is no to or from the client) thus a univariate Hawkes process with exponential kernel was used to attempt to model it.

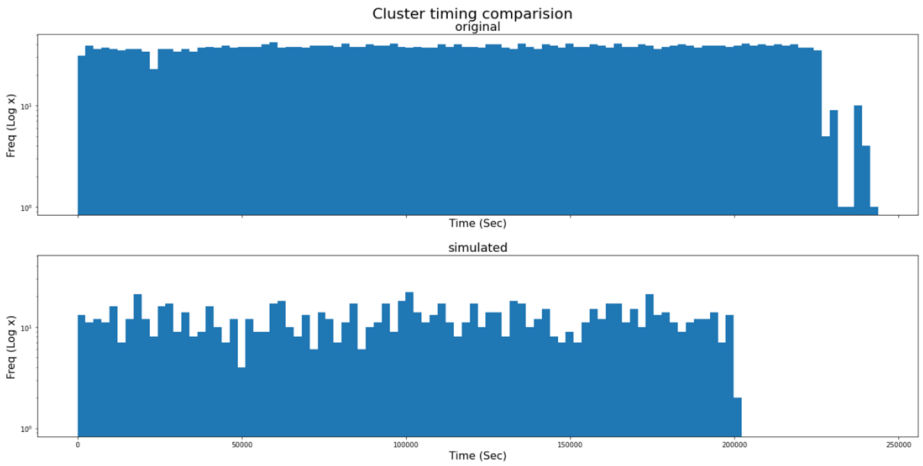



Figure 3.5: Comparison of event sequences for the occurrence of clusters.

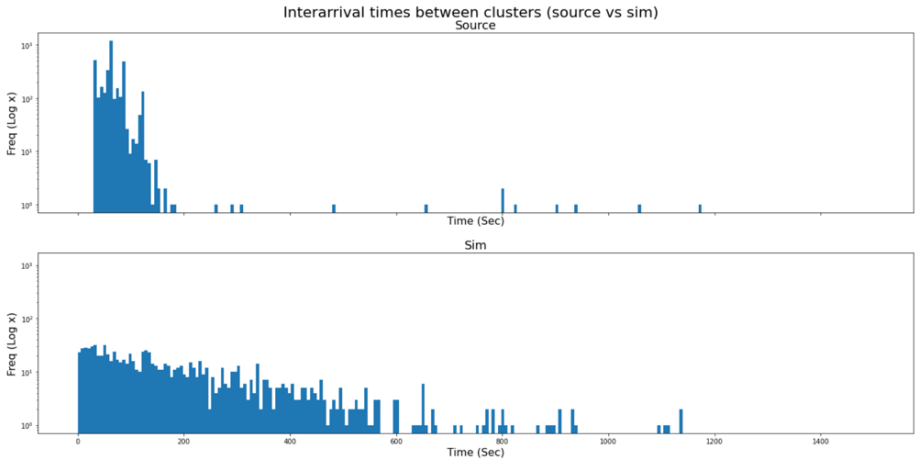



Figure 3.6: Comparison of cluster interarrival distributions

(Figure 3.5) contains two histograms (log scale) of when new sub-sequences occur, with the top being for the source data and the bottom the simulation from the learned model. It is immediately apparent that the simulated event sequence contains significantly fewer events than the source, approximately three times less. The reason for the poor event density can be seen in (Figure 3.6), which contains the interarrival times between clusters. The interarrivals from the source event sequence (top) are more heavily concentrated at smaller time values, most between 0 – 100 seconds and stop almost completely before 200 seconds. Where as the interarrivals from the simulated sequence (bottom) span over a larger range of time, trailing off closer to 600 seconds.

This behaviour is likely caused by a lack of excitement or too high decay value, either prevent the event intensity from being high enough for long enough to produce higher densities of events over a shorter time span. Thus, larger interarrivals lead to a lower event density than required and a sub optimal model for capturing the dynamics of the source data.

### Sub-Sequence Modelling: Using an Exponential Kernel

The second part of the problem involves modelling events within each sub-sequence, taking all such sub-sequences as input when training the model. In the first trial, an exponential kernel was used to model intensity decay.

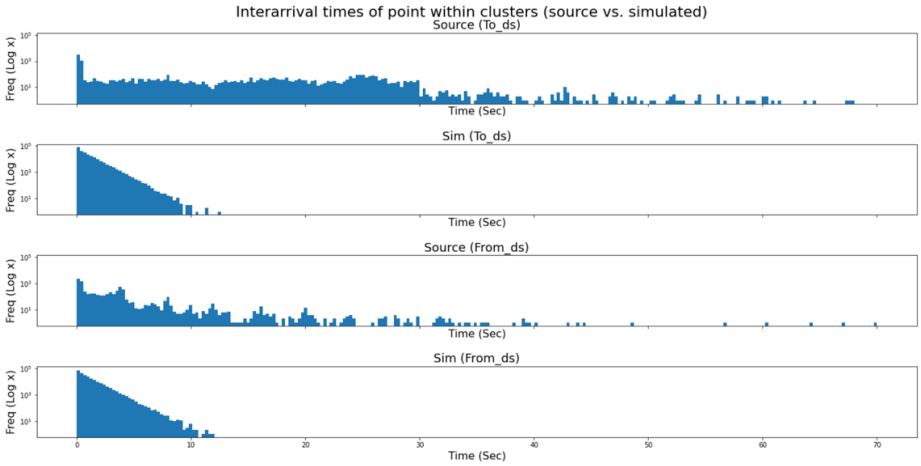



Figure 3.7: Comparison of sub-sequence interarrival distributions using an exponential kernel

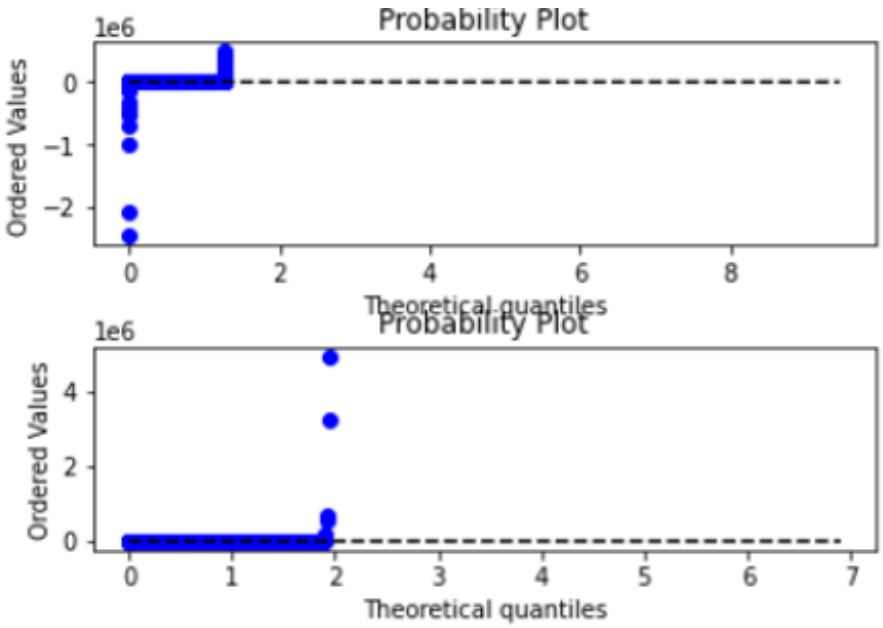



Figure 3.8: Subsequence residual plot

(Figure 3.7) demonstrates the interarrival times between each event. Analysing the two simulated distributions (graphs 2 and 4) reveals the model has learned a consistently decreasing distribution. This distribution is vastly different from the source distribution (graphs 1 and 3), instead containing an initial spike in small interarrivals followed by a sharp drop off, thereafter only a few larger interarrival times exist. Also note how the simulated distributions cover a shorter period than the source distributions, stopping at approximately 11 seconds as opposed to 30 seconds in the source. An explanation of this erratic shape is that most interarrival times are short, potentially coming from a separate distribution than the longer interarrival. This would lead the learner to averaging the two distributions to try and fit a single exponential kernel, leading to a completely inaccurate model as seen here.

This is reinforced by the residual plot present in (Figure 3.8). If the simulated distribution modelled the source well, the plot should generally follow the dotted line (points randomly distributed above and below), very clearly this is not the case.

### Sub-Sequence Modelling: Using a Non-Parametric Estimation Kernel

To try and make up for the shortcomings of the exponential kernel and improve the chances of modelling a more erratic source distribution, the second trial used an EM kernel. It should provide increased flexibility to model the interarrivals properly, capturing the initial spike and subsequent drop off.

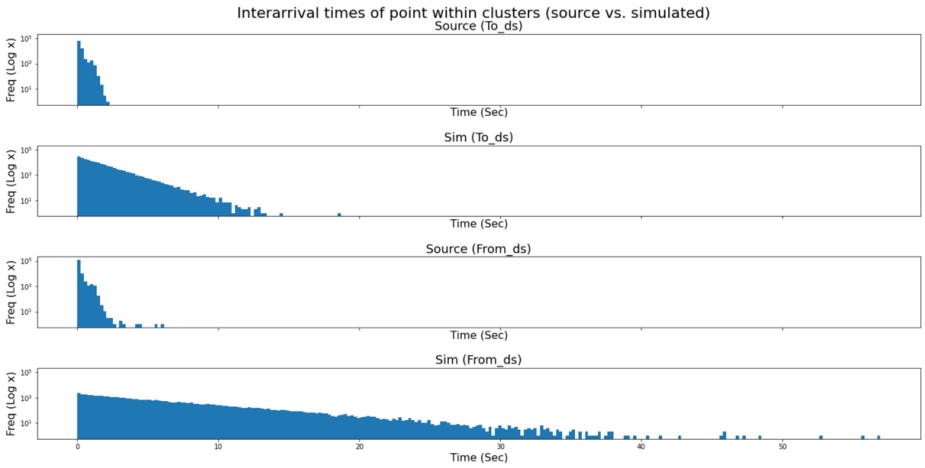



Figure 3.9: Comparison of sub-sequence interarrival distributions using an EM kernel

Despite initial predictions, the EM kernel produces a worse variant of the exponential kernel, as seen in (Figure 3.9). It has a similar shape but instead covers a significantly larger range of interarrival times not present in the source. Once again, it appears to be averaging the initial spike in the distribution and the larger interarrival times. It partially captures the density of the initial spike, containing a similar number of events, but does not drop off quick enough.

### Result Analysis

The results from the previous experiments suggest at least one of the following theories for why modelling failed to succeed in capturing the dynamics of the source.

- There is the possibility the distribution does not follow the self-exciting nature expected by a Hawkes process. This is further explored in Section 4.1.

- The source distribution is a composite of multiple distribution, thus fitting a single distribution to it is meaningless. This is further explored in Section 3.3.

## 3.2 Intensity Free Experiments

The following models only consider a single client and are univariate (messages to and from a client are in the same process, unlike Hawkes processes in Section 3.1 which had them split). In order to make up for the missing dimensionality, packet direction information has instead been added to the process as marks. This was done since multivariate processes are unsupported by the intensity free learning module and a univariate process with marks can give the same effect as multivariate one.

### 3.2.1 Modelling Directly with Intensity Free

Any experiments performed with the intensity free model were on the raw data (no clustering). This is a result of the way the model works, spitting the data into arbitrarily sized clusters degrades performance.

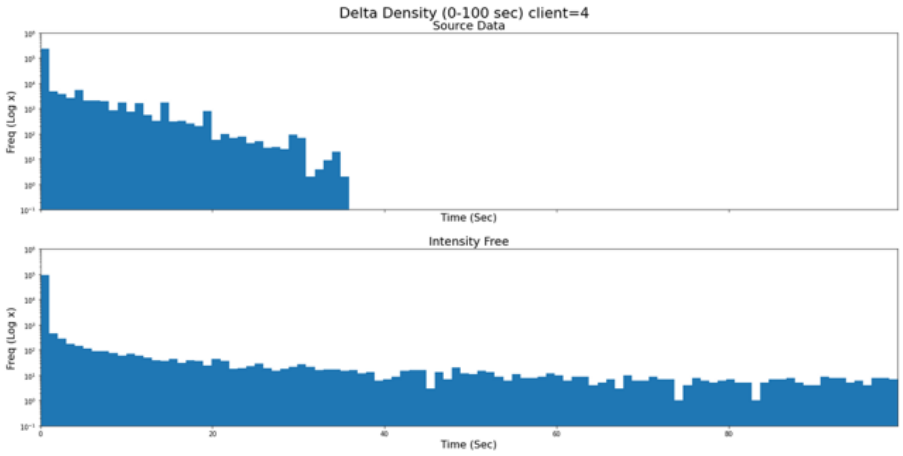



Figure 3.10: Comparison of interarrival distributions using the intensity free model without truncation of points

The first issue encountered with the learned distribution is the large amount of unre-

alistic interarrival times generated. Up to 40% of all simulated event interarrivals were greater than 40 seconds, the point in the source distribution where interarrivals do not exist beyond, as in (Figure 3.10). Many of these interarrivals were unrealistically large, an example of which is $6.3 \times 10^{17}$ years, approximately a billion times longer than the age of the universe. A solution to this is to truncate the erroneous data and only focus on the smaller more reasonable points.

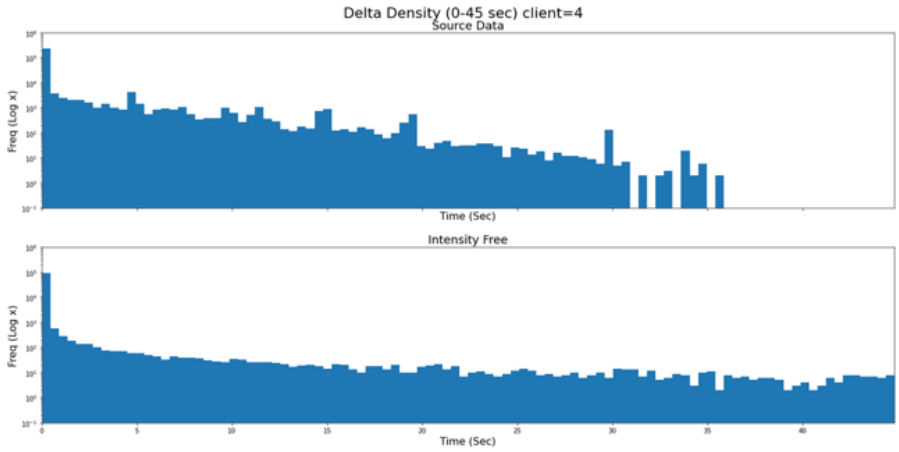



Figure 3.11: Comparison of interarrival distributions using the intensity free model with truncation of points

(Figure 3.11) contains the same distribution as (Figure 3.10) only truncated at $T = 45$ sec and zoomed in. On a large scale, the truncated distribution captures the initial spike in small interarrivals and quickly drops, but not in the same manner as the source. It drops too far, producing less than 100 points for each time bar whereas the source distribution continuous producing over 100 points per bar up until 15 seconds. To further reinforce this, the means of the two distributions are wildly different. The source distribution has a mean interarrival of 0.8975 seconds compared to the simulated distribution of 15.2305 seconds. This suggests the intensity free approach is generating interarrivals approximately 17 times larger than what is required. Zooming in on both distributions (20 ms timeframe) reveals another hidden issue.

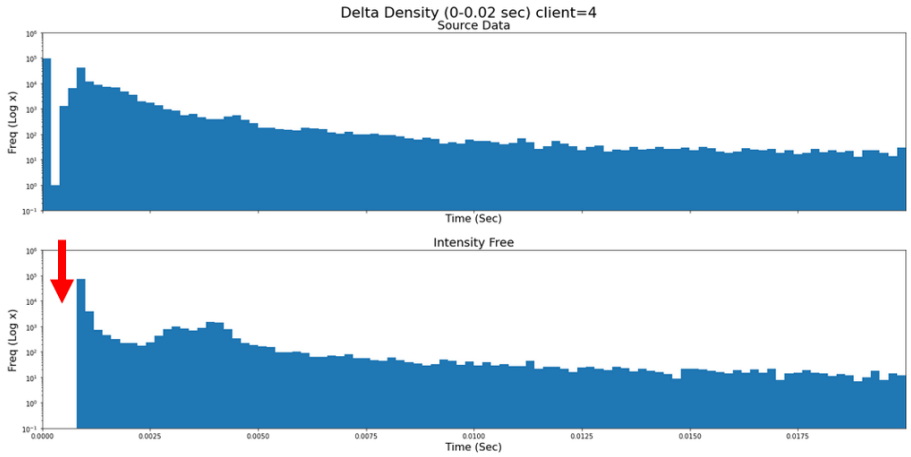



Figure 3.12: Comparison of interarrival distributions, zoomed in to 20ms, showing a gap in the learned distribution

(Figure 3.12) demonstrates the simulated distribution missing a region of time (signified by the arrow) present in the simulated distribution. This is a clear indication that the model is operating sub-optimally at capturing the dynamics of the source process. (Figure 3.12) also reveals a split in interarrivals in the source distribution (top left of the graph, the second bar is very low) possibly due to two separate distributions. This is discussed in Section 3.3.

## 3.3 Composite Distribution

### 3.3.1 Evidence for a Composite Distribution

Further investigation of the source interarrival distribution reveals a distinct split in the distribution. The top half of (Figure 3.13) shows the beginning 20 ms, where after an initial spike there is a significant drop off (almost to zero) before the rest of the distribution continues. Zooming in further on this drop off, the bottom half of (Figure 3.13), to a timescale of 2 ms, in fact reveals a total absence of interarrivals for a period of approximately 550$\mu$s. The distribution is discontinuous.

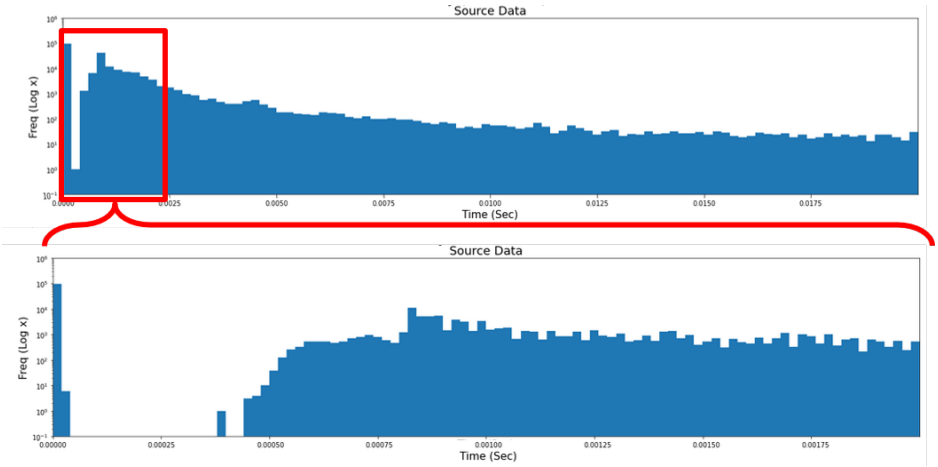



Figure 3.13: Split in interarrivals present in the source distribution. 20ms and 2ms time frames respectively.

This split in the distribution appears in multiple datasets and is very consistently located, always present at 250$\mu$s in the distribution. In addition to its consistent location, the size of the 'silence' is also very consistent, approximately 550$\mu$s long with a standard deviation of 106$\mu$s. Additional statistics on the length of the split can be found in (Appendix 1: A.1) and additional distributions containing the split in (Appendix 1: A.2).

Another significant observation is that a non-trivial portion of each dataset appears on the left-hand side of the split. The initial spike in interarrivals, contains anywhere from 10% to 40% of the total distribution within a period of approximately 20$\mu$s.

### 3.3.2 Evidence Analysis: The Significance

Following the evidence above, there is reason to believe the distribution we have tried to model so far is a composite of two separate distributions, the left and right sides of the split. This may explain the trouble encountered so far; applying a single distribution to a composite one will not capture it effectively, let alone trying to apply a continuous distribution to a discontinuous one. The split in the distribution may be caused by an underlying characteristic of networks such as the latency of the connection or some sort of separation between control and data packets. Potential reasons for the split will be

analysed further in Section 4.1.

To model a composite distribution, we need to create a composite model, thus we will split the problem into three parts; modelling the left and right sides of the split then use a third model to merge the two back together to form a single distribution. Splitting the distribution This is explored in Section 3.5.

## 3.4 Hidden Markov model experiments

The event sequence input into the HMM is slightly different than the other two models. Instead of taking the timeline of events directly, the HMM takes a timeline of interarrival times. This is because the HMM does not perform the conversion internally, instead requiring that the distribution to learn (the interarrival distribution) be inputted directly to the HMM. Additionally, the event sequence is not separated based on packet direction like in previous experiments, instead all being part of the one event sequence

The HMM uses a single hidden layer, taking the interarrival sequence and splitting it up into states internally which are also used internally to model the source sequence. Each state has its own learned distribution which is used by the model to sample from and return a new sequence of interarrival times.

### 3.4.1 Direct Application

Before investigating the use of HMMs for modelling the split distribution, first we will try and apply it directly to the source distribution. This will act as a baseline comparison for both the Hawkes process model and intensity free models that learned directly from the source distribution.

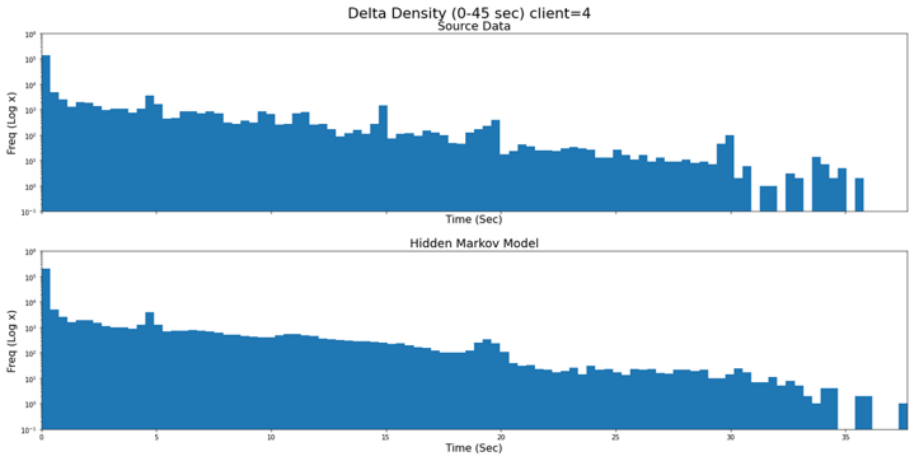



Figure 3.14: Comparison of interarrival distributions using the hidden Markov model

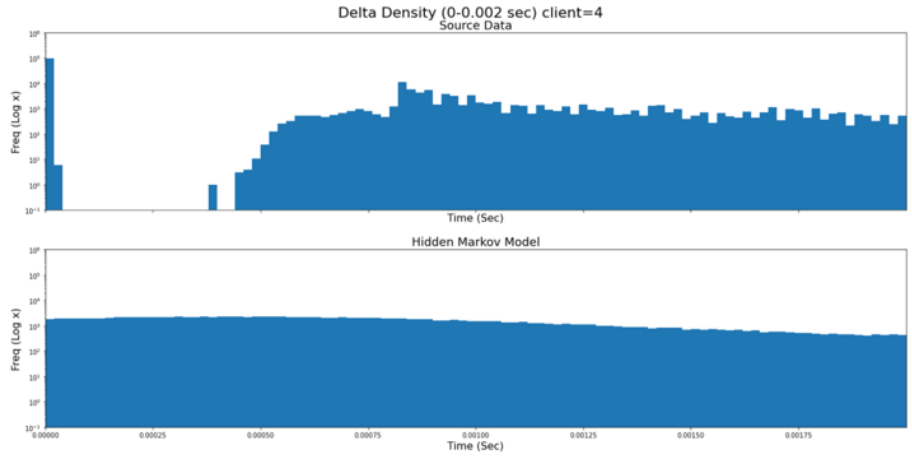



Figure 3.15: Comparison of interarrival distributions around the split location using the hidden Markov model

Applying the HMM directly yields a good fitting model, shown in (Figure 3.14), containing many good features. First, it captures the initial spike in intensity and sub-sequence drop off. Second, the interarrival distribution after the peak does not majorly over or underestimate the intensity of the source distribution, staying on par with it and terminating at approximately the same time (35 seconds). Third, the HMM model has added some minor variations to its distribution in similar locations and with similar shape to what is found in the source. This is a big benefit over previous Hawkes and intensity free models which would generate smoother, more uniform distributions that can be identified as synthetic easier. Clearly, these variations are not as aggressive

as what is found in the source distribution (they are smoothed out and generally less intense), but this is to be expected given the number of states this model was trained with (25 states). Increasing the number of states in the model would improve this at the cost of computational complexity.

Like previous models tested, the HMM model also struggles to model the split in the distribution. Zooming in to the split, in (Figure 3.15), highlights how the model has completely ignored the split, generating points throughout it. Like the Hawkes EM model, this is probably because the model was not complex enough (did not have enough states) to model such a minute timeframe of 20$\mu$m, instead averaging over a longer timeframe and engulfing the split.

## 3.5 Creating a Composite Distribution

To create a complete model of the interarrival distribution we need a three part model. The parts are as follows:

- Modelling the distribution on the LHS of the split
- Modelling the distribution on the RHS of the split
- Modelling which of the two distributions to pull from at each time step to form a complete distribution

The first two parts, modelling the LHS and RHS requires splitting the source data into two categories based on the event interarrival time. Models involving the LHS will receive events with an interarrival time less than where the split occurs (i.e. all the interarrivals on the left half of the source interarrival distribution). Similarly, for models involving the RHS, they only receive events with interarrivals greater than the split. Since the split always occurs at $T = 556.4\mu s$, this will be the cut-off point for determining the two sides of the distribution. In this way, each side will only receive data from their respective sides of the split and thus only model the distribution on

that side forming two disjoint distributions which can be combined using the third part of the composite model.

### 3.5.1 Modelling the LHS Distribution

To effectively analyse the left part of the distribution, we must first zoom in to a period of 25$\mu$m. The top graph in (Figure 3.16) shows the source distribution at this timescale, revealing a discrete curved distribution. The curve appears to be like a lognormal distribution, one with the same mean and standard deviation has been plotted in the lower half of (Figure 3.16) for comparison.

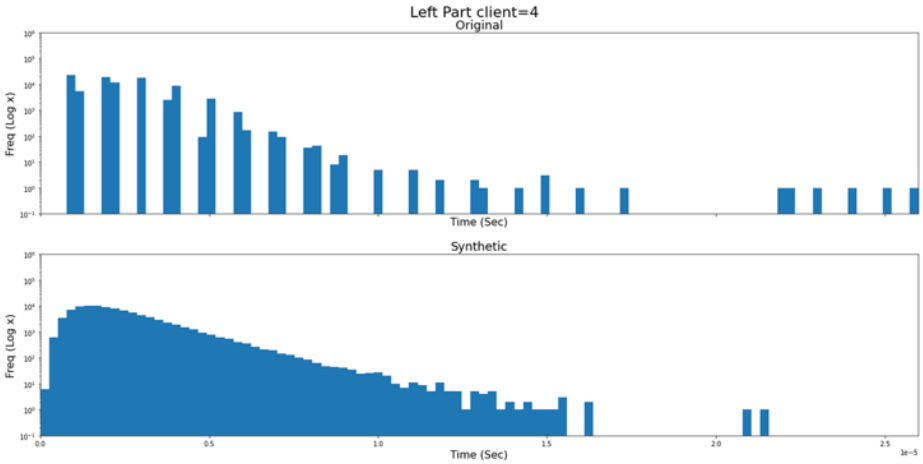



Figure 3.16: Comparison of interarrival distributions on the LHS of the split using a lognormal model

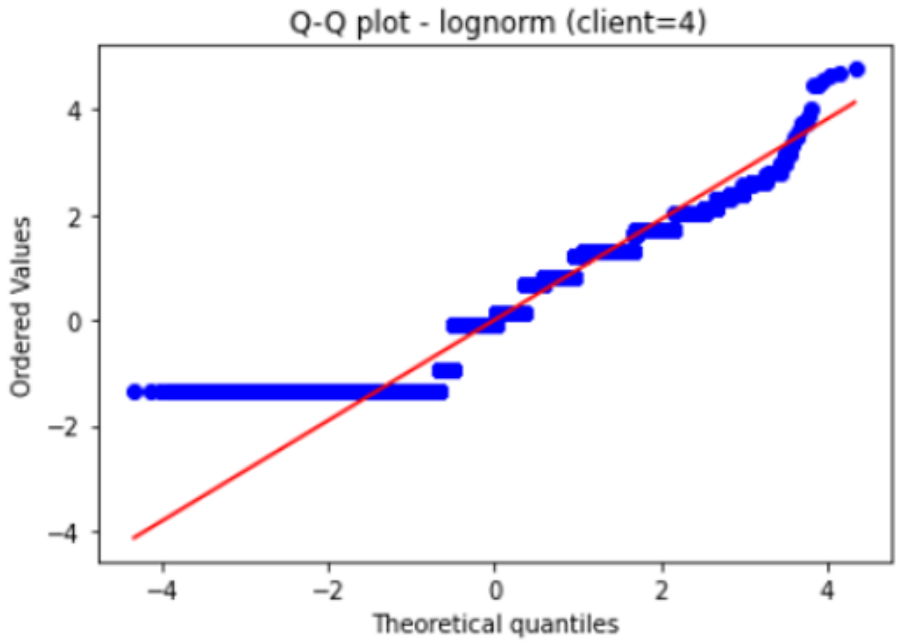



Figure 3.17: Q-Q plot of the source distribution vs a lognormal distribution with the same mean and standard deviation

The goodness of fit of the lognormal distribution can be determined using a Q-Q plot, as in (Figure 3.17) and an $r^2$ correlation value. The Q-Q plot visually shows the closeness of fit to a lognormal. Most points in the Q-Q plot roughly follow the red line, indicating a close match to a lognormal distribution. This is further reinforced by an $r^2$ correlation value of 0.9529, indicating that over 95% of the variance in the source distribution can be explained by a lognormal distribution. This conveys that there is a strong relationship between the two. Thus, a lognormal (or slight variation of) is a good approximation to the observed distribution.

An additional feature to note is the discrete nature of the distribution found in the top half of (Figure 3.16) and the bars in the Q-Q plot in (Figure 3.17). The distribution is not a continuous one, instead formed by many regularly spaced bursts of points.

### 3.5.2 Modelling the RHS Distribution

The right half of the distribution is the more complex half and is basically the same shape as in previous experiments, detailed in Section 3.1, Section 3.2 and Section 3.4. The only difference is a slightly smaller initial spike in interarrivals due to the LHS of the distribution being removed. A good starting point is to model it using the three methods previously explored, Hawkes processes, intensity free modeling and HMMs. This will help to determine if the split in the distribution is the cause for the modelling issues experienced previously.

#### Hawkes Process

When modelling the right distribution as a Hawkes process an EM kernel was used to provide increased flexibility in capturing key features of the distribution, such as the initial spike in interarrivals.

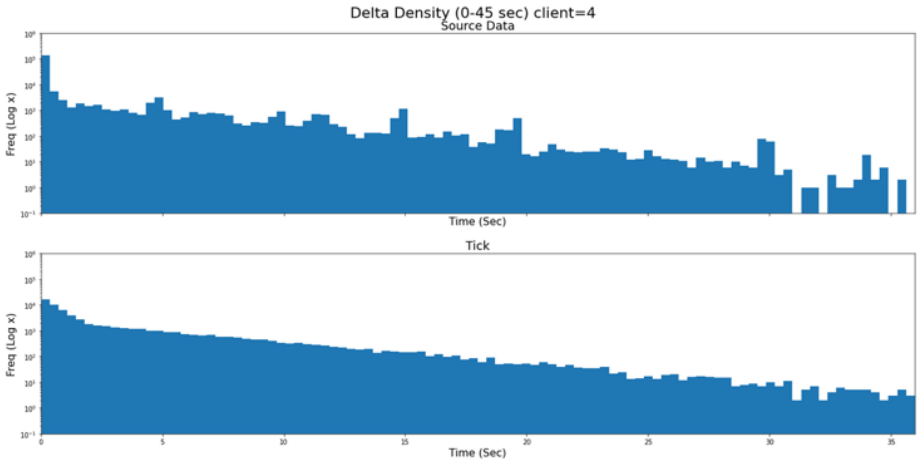



Figure 3.18: Comparison of interarrival distributions on the RHS of the split, using a Hawkes EM model

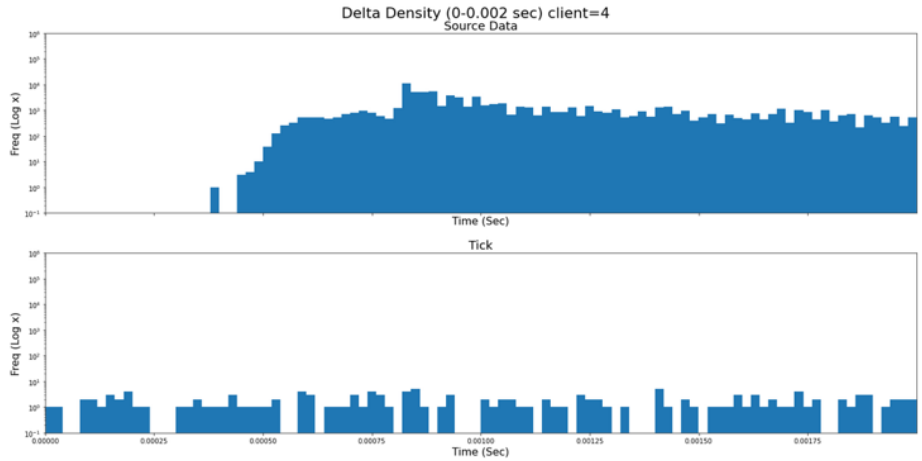



Figure 3.19: Comparison of interarrival distributions on the RHS of the split, at the split location, using a Hawkes EM model

From (Figure 3.18), on a very coarse level the model is capturing the shape of the data but is missing many distinct features that easily tell it apart from the original. Firstly, the initial spike is not modelled well, only producing thousands of points when tens of thousands are required; but it is clear an attempt was made to try and learn it, signified by the slight increase in intensity in that general timeframe. Secondly, there is a lack of 'natural' variation in the model, it does not attempt to create any small curves or bumps present in the source distribution, instead opting for a smooth, consistently decreasing distribution; a clear indicator this is not 'natural' data. Thirdly, zooming in to the area where the distribution splits, as seen in (Figure 3.19), shows a complete disregard for the silence period, producing points throughout it and then failing to model the ramp up in intensity when the source distribution actually starts. Fourthly, the mean interarrival time of the generated distribution is noticeably different than the source distribution; 1.365 sec in the source and 3.594 sec from the Hawkes model, a clear indicator the distributions are not similar (it is however closer than any Hawkes or intensity free model tried so far).

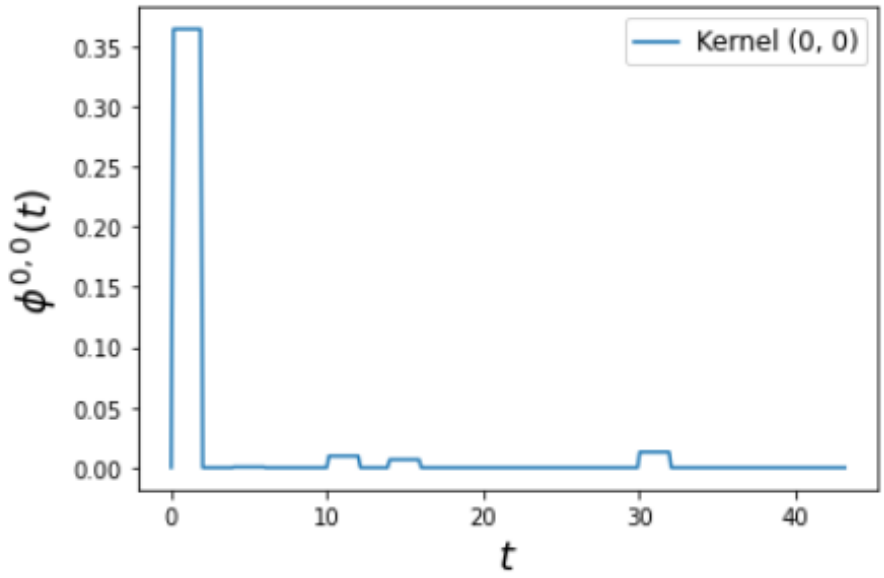



Figure 3.20: Hawkes EM kernel for modelling the RHS

It is likely many of these issues were caused by the simplicity of the model, shown in (Figure 3.20), only containing 18 timeframes to control over a period of 36 seconds. However, more complex models with more timeframes to manipulate, and thus more freedom, performed worse as flexibility increased. Theoretically, increasing the flexibility should have allowed it to produce a better fitting model, especially since the simple model was able to capture the general shape. However, it is likely there is an issue with the learning process, some technicality in the implementation of the learner, which is preventing it from improving.

**Intensity Free**

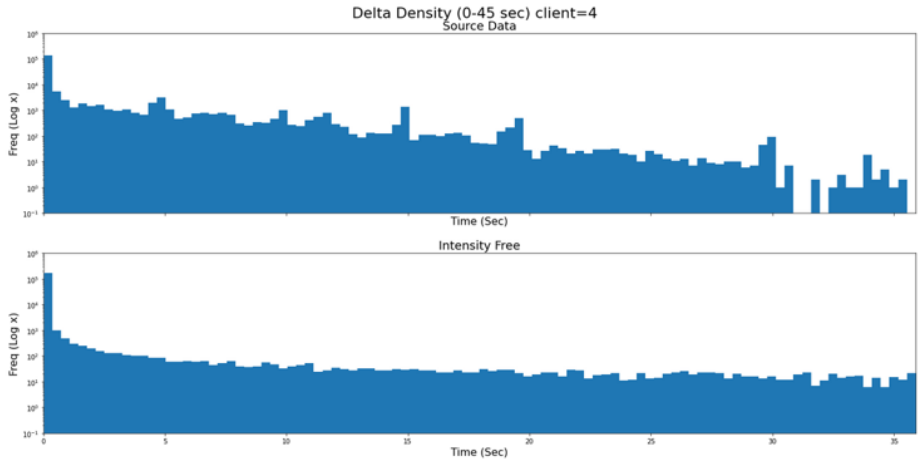



Figure 3.21: Comparison of interarrival distributions on the RHS of the split, using an intensity free model

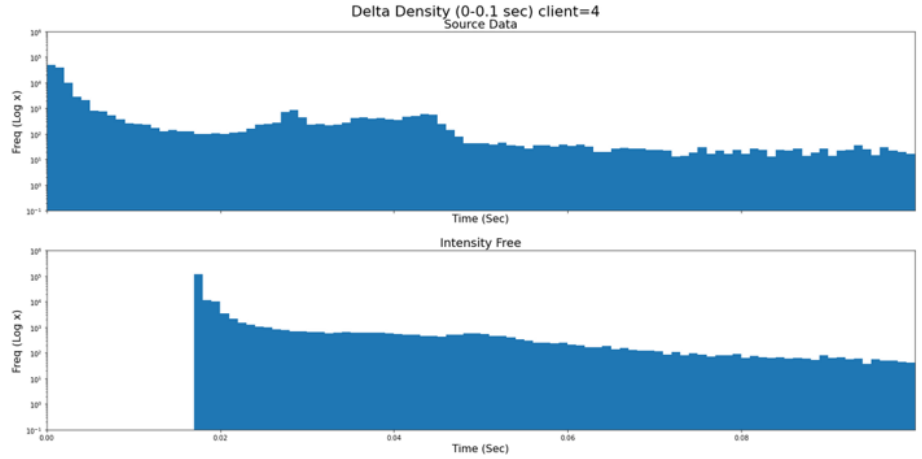



Figure 3.22: Comparison of interarrival distributions on the RHS of the split, at the split location, using an intensity free model

The results of applying an intensity free model can be seen in (Figure 3.21). On a large scale, the model can capture the large initial spike in interarrivals but falls short in modelling the rest of the distribution. After the spike, intensity quickly drops to below levels from the source distribution and consistently stays there, underestimating almost the entire distribution. Additionally, zooming in on the initial spike, seen in (Figure 3.22), reveals a 20ms long period where no points appear, almost as if the entire learned distribution has been shifted to the right by 20ms. Thus, intensity free is not a good fit for the right side of the split distribution.

### Hidden Markov Model

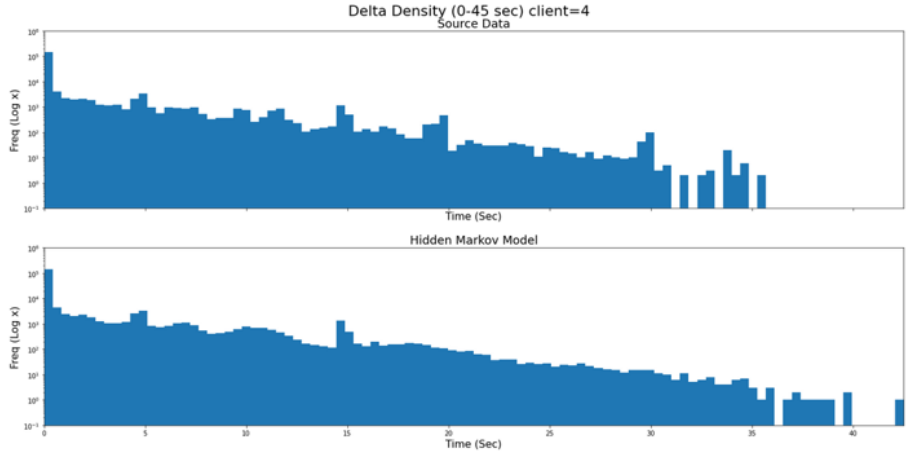



Figure 3.23: Comparison of interarrival distributions on the RHS of the split, using a Hidden Markov Model (45 seconds)

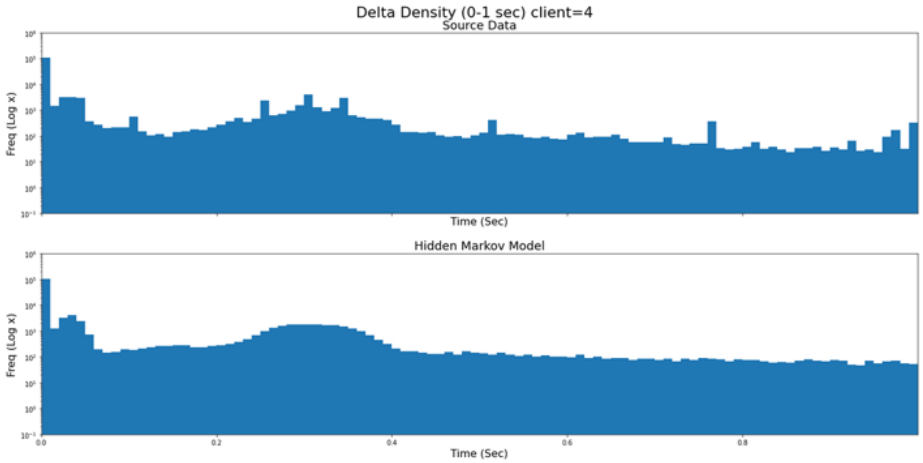



Figure 3.24: Comparison of interarrival distributions on the RHS of the split, using a Hidden Markov Model (1 second)

Applying a HMM to the RHS of the distribution produces similar results to applying it to the entire distribution. However, with less data to consider and a similarly complex model (25 states), it can produce a slightly better fit. The model better captures the minor variations in the source distribution and is a good fit, as seen in (Figure 3.23). Additionally, the mean of the learned distribution, 1.418 seconds, is close to the actual mean of 1.365 seconds, only off by about 4%.

At a slightly smaller timescale of 1 second, as in (Figure 3.24), the model is still capable

of capturing the general shape of the distribution. However, the distribution tends to be much smoother than the source, an indication that the data is fake, but can be improved by increasing the number of states in the model when training.

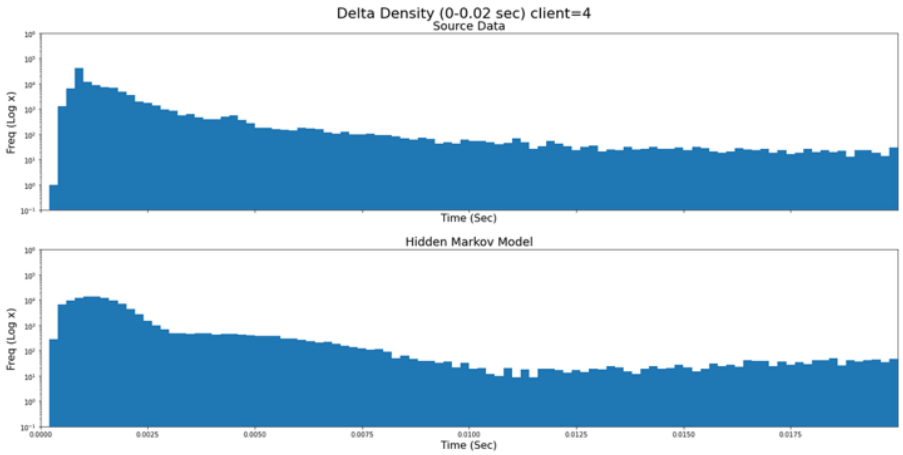



Figure 3.25: Comparison of interarrival distributions on the RHS of the split, using a Hidden Markov Model (20ms)

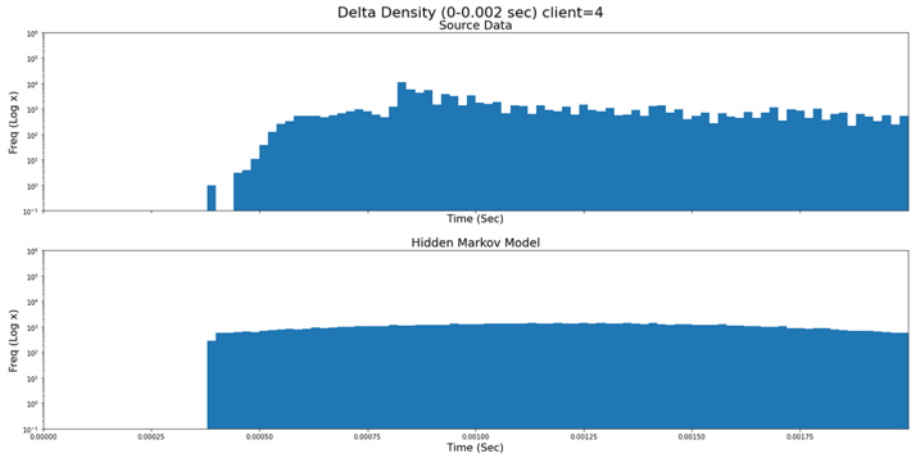



Figure 3.26: Comparison of interarrival distributions on the RHS of the split, at the split location, using a Hidden Markov Model

Significantly smaller timespans, such as 20ms as in (Figure 3.25), is where the model starts to break down. It is possible to see each individual state in the HMM, where the curve of the graph sharply changes. An easy remedy for this is to improve the resolution of the model (the number of states), with more states it will look better at smaller time frames. However, this then becomes a question of 'what is good enough?' which is a different problem entirely and out of scope for this project.

The last thing to consider is the model's performance around the split location. Like the previous HMM model it produced points through the split. This appears to be an averaging error, averaging the distributions intensity from 0 to the end of the first state instead of from the location of the first point. Any excess points appearing within the split are thus truncated too produce a valid model, as in (Figure 3.26).

Overall, the hidden Markov model produces the closest and most realistic fit out of the three temporal methods explored. Additionally, it offers the ability to be easily scaled up for better accuracy. The HMM will be used as the RHS synthetic distribution during the process of merging.

### 3.5.3 Forming the composite model

Neither Hawkes processes nor intensity free can model merging two processes effectively. They focus on modelling interarrival times, whereas merging the distributions requires modelling the order of events; the interarrivals come from the two distributions, left and right, which are being sampled from. To merge the distributions, a Hidden Markov Model is required. As training data, the model will take a modified version of the source event timeline, where events with an interarrival falling on the LHS of the split are converted into 0's and anything on the RHS converted into 1's.

The distributions to be merged will be the lognormal model for the LHS (discussed in section Section 3.5.1) and the HMM for the RHS (discussed in section Section 3.5.2) as these performed the best for modelling their respective sections.

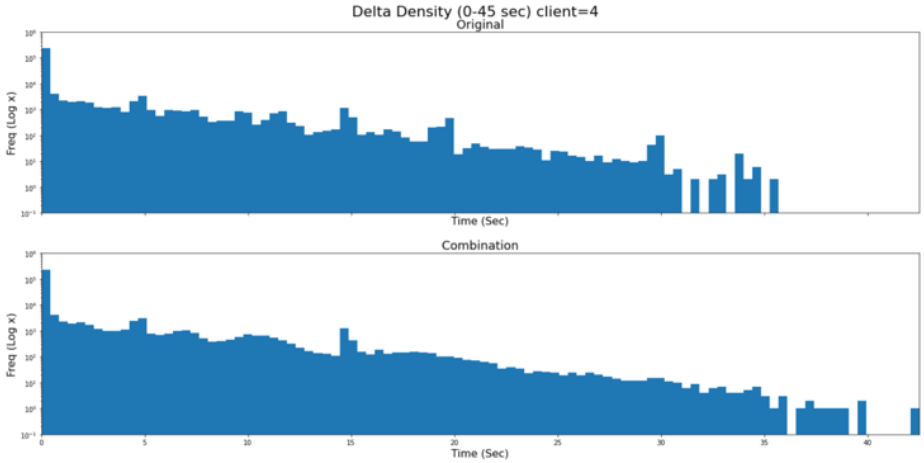



Figure 3.27: Comparison of interarrival distributions using a composite model

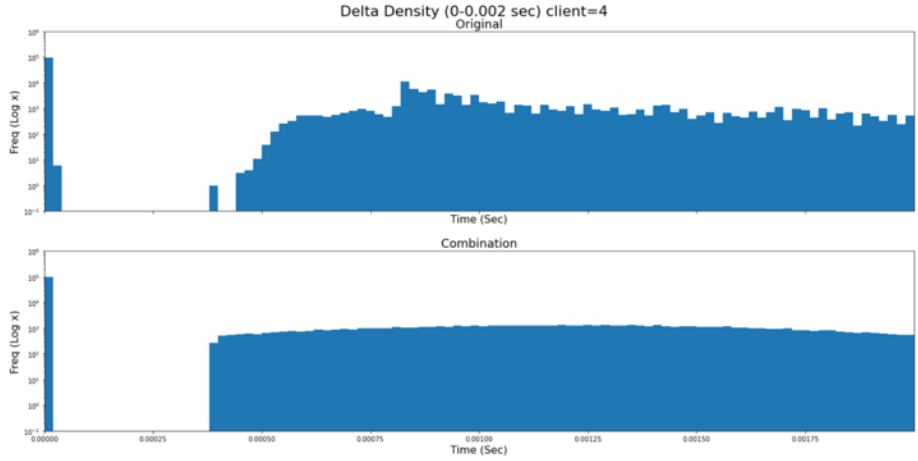



Figure 3.28: Comparison of interarrival distributions, at the split location, using a composite model

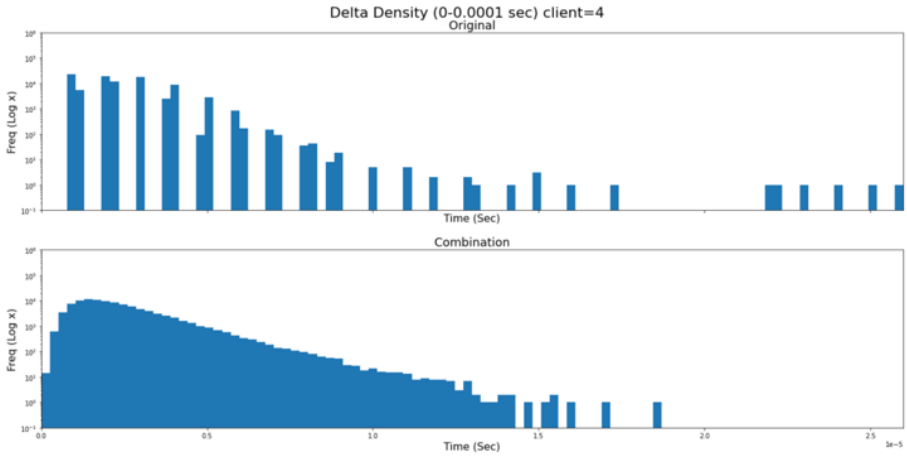



Figure 3.29: Comparison of interarrival distributions on the LHS of the split, using a composite model

On a large scale, the merged distribution mirrors the source distribution well, seen in (Figure 3.27), similar how it was in Section 3.5.2. This is due to the current timescale (40 seconds) and graph primarily being made up of points from the RHS distribution (all the LHS points are only present in the initial spike). Thus, the RHS of the distribution has been merged well.

Upon zooming into the split, shown in (Figure 3.28), it is clear the merged distribution contains the split in the distribution both in the correct location and of approximately the correct length. Additionally, the initial spike is approximately the correct size. Therefore, the split has been modelled correctly.

Additionally, observing the spike formed by the LHS distribution on a timescale of 25$\mu$s in (Figure 3.29), shows the merge has preserved the lognormal distribution. This means the LHS has been merged successfully.

Lastly, comparing the merged mean interarrival of 0.87366 seconds to the source mean of 0.89756 seconds yields only a difference of less than 3%. With the four significant parts of the split distribution (LHS, split, RHS and mean) present and correct in the merged distribution, the merge has accurately modelled the source distribution of interarrivals.

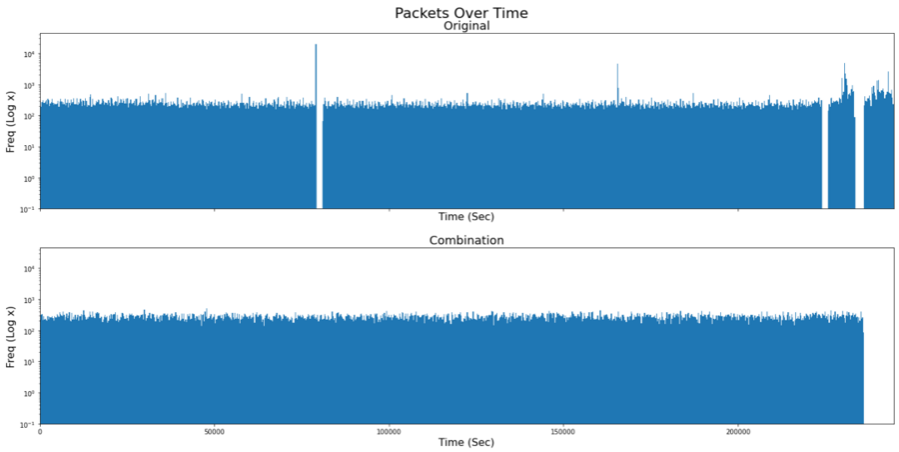



Figure 3.30: Comparison of interarrival distributions on the LHS of the split, using a composite model

The last thing to consider is the timeline of events formed by merging the two distributions. It is possible to form the correct interarrival distribution simply by drawing the correct amount of points from the LHS and RHS distributions. But, for example, could place all the LHS interarrivals at the beginning of the timeline and the RHS at the end. This would form a tight cluster of events with tiny interarrivals at the beginning of the timeline followed by a sequence of highly spaced events for the remainder. Observing the timeline in ¡fig a¿, this is clearly not the case. The HMM merging model has successfully interleaved the two distributions. This can be gauged by visually comparing the average event density over time for both distributions, both having approximately the same density and with a similar degree of noise.

With both the merged distribution of interarrivals and the subsequence timeline of events being approximately similar to their source counterparts, the composite model of lognormal for the LHS, HMM for the RHS and HMM to merge the two, is successful in accomplishing the task of autonomously generating a realistic fake Wi-Fi sequence.

# Chapter 4

# Evaluation

## 4.1 Result Summary and Analysis

### 4.1.1 Individual Technique Results and Evaluation

#### Hawkes Processes Modelling

The first model produced using Hawkes processes, directly applying the process to the data, would result in excessively high excitement and decay values for the intensity. The result of which is an excessive amount of clustering or even creating a pseudo Poisson process (due to all points being generated by the baseline). Clearly, this is a sub-par model, barely learning any features of the source distribution. This suggests there is a deeper pattern to the data that has not yet been captured, at least not by a simple application of a Hawkes process.

Breaking the data into clusters improved results over the direct approach but was still far away from being good. While excitement and decay values were less extreme and more in the realm of 'reasonable' values, the interarrival distributions generated were too uniform and different from the source distribution, either generating a longer or shorter distribution with different characteristics. Once again, this is probably the

result of applying an incorrect model to the data. While clustering is sound in theory, it does assume that related packets are always close to each other (not something you can guarantee) and does not consider the possibility of multiple independent data streams (e.g. concurrent downloads).

The final Hawkes model, applying the process to the right half of the split distribution was more promising than the previous two. It partially captured some key features, such as increasing intensity for the initial data spike and got the tail end right but was missing the density of points required and thus is too dissimilar. However, the model was simple and appeared to be trying to learn unlike previous Hawkes models. This leads to the belief that maybe this is a correct model of the distribution in some way. The chances are there is some technical issue holding it back rather than just being a bad model to apply. Theoretically, this makes sense as the EM kernel would work better if the actual intensity function to learn is very eccentric as suggested by the source interarrival distribution. Out of the three applications of Hawkes processes, this is the only one that kind of worked well.

#### Intensity Free Modelling

Both times modelling with the intensity free approach missed the mark, producing similarly incorrect results. In both the direct approach and with only the right half of the split yielded a gap at the beginning of the learned distribution and a rapidly declining interarrival density, leading to underestimation of the source distribution. The likely cause is a weight explosion issue, causing one of the components in the mixture to dominate all the others producing weird and incorrect distributions.

#### Markov Modelling

Clearly, Markov models were the superior model to Hawkes and intensity free, at least during the previous experiments. Both in the direct application and with just the right half of the distribution, it was able to consistently approximate the source distribution

with relative accuracy. It always contained several key features of the source distribution including, the initial spike, when the distribution terminates, and small variations in the distribution and their locations. There is little surprise of the effectiveness of the Markov model, it has proven itself over decades to perform exceptionally well with sequencing tasks. The only issue with it is its limited memory. This was not explored well enough in this project, but while some of the statistics of the distributions it could produce may look good (such as the mean), it may not be achieving the correct ordering of points. E.g. after many short interarrivals there might be a good chance of a longer one. These are the kinds of orderings that the more advanced Hawkes processes and intensity free models (including NN backed method like the transformer) should be more capable of achieving.

#### Distribution Split

This is an important discovery in the context of the project as it consistently appears throughout the datasets with similar size and in similar locations. This suggests the source distribution is actually a composite of multiple distributions and may provide an explanation for why models trained on the entire distribution were producing odd results, especially since their results improved once accounting for the split.

#### Composite Model

The composite model's ability to model the distribution is by far superior to anything else attempted, accurately modelling important features of the source distribution such as the initial spike, the split and general variations in intensity as the distribution trails off. Every other model tested failed to properly model the split, usually generating points all throughout it. This is one of the things that separates it from the rest.

Despite its success, it does not entirely fit the goal of this project. It does meet the requirement of being autonomously produced, generated through a series of learning algorithms, and it does generally look good, but this project is missing a concrete way

of defining how realistic the model is. What this requires is an autonomous method of producing a number which encapsulates the realism of a model. This can then be used as a solid metric for determining if the model is realistic enough and provides an effective way to compare the realism of different models quickly and accurately. This is discussed further in Section 4.3.

Overall, the only model with the potential to be realistic enough for use in a honeypot is the composite model, all other models are too different from their intended targets and can easily be identified as a fake.

### 4.1.2 Binary Nature of Network Traffic

An overarching explanation for why many of the models were having trouble learning the distribution, especially Hawkes processes and the intensity free model, and for why the split in the distribution exists could be to do with the nature of computer network traffic. Computers on a network generally transmit data as fast as they can or not at all (spool up time caused by TCP is negligible, especially over longer data transmissions), thus intensity is either at maximum or zero. This is unlike the more natural ramping up and ramping down of intensity expected by something like a Hawkes process. This is relevant when you consider the types of applications something like a Hawkes process is typically used for, modelling more natural processes like earthquakes, or social media virality; the key natural factor being humans. While humans do have a degree of input in computer networks (traffic is generated when the human does something) most of the control is by the machine, determining when and how messages are sent after the initial trigger. As a result, there is less natural behaviour; explosions/bursts of packets sent at a constant intensity before abruptly ending when done. This could be a hint to why initial models with Hawkes processes would favour incredibly high excitement and decays, to try and model this on/off behaviour. An explanation for the large initial spike in tiny interarrival times could also stem from this, being produced by a cluster of successive packets send back to back. Additionally, the split in the interarrival distribution can be explained by this theory; the silence in the distribution

could be caused by the latency of the connection. While this would mean the split should grow and shrink to cater for different latency connections, which is not observed here, note that the data used in this project was collected from a company network and it is reasonable to assume that most of the traffic is accessing internal resources. This would result in very consistent latency as the distance to the server is never changing.

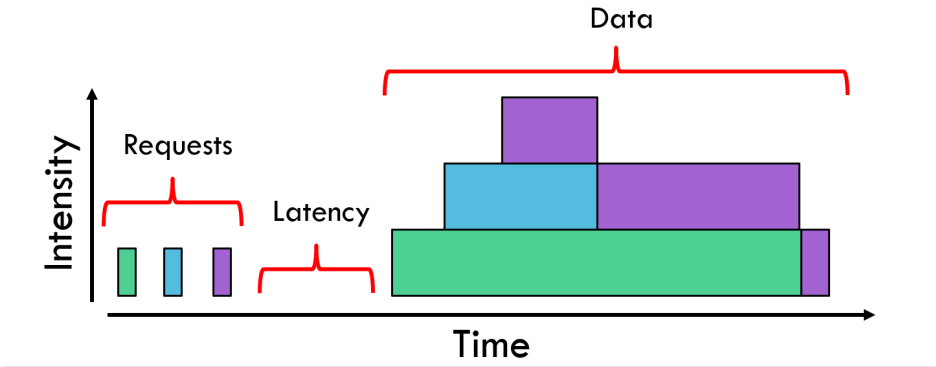



Figure 4.1: Example packet stream. Note the latency between the request and corresponding response and how intensity immediately increases or drops as new connections start/terminate

## 4.2 Project Challenges

Over the course of the project, numerous setbacks were encountered, these are detailed below:

There were numerous issues with the GPU server provided by CSE around the times GPU models needed to be trained. Since a GPU is a necessity for these models a switch was made to using a free online GPU runtime offered by Google Colab. This enabled work to continue with minimal impact and in fact may have been an improvement, enabling easy collaboration and sharing of the codebase.

About a third of the way through the project, after completing background research into neural network methods and setting up workflows to put these models into practice, the direction of the project shifted slightly. This involved moving away from NN methods,

such as the transformer, to temporal and discrete sequence modelling techniques, such as Hawkes processes. The cost of this move is a significant portion of time spent initially researching the original NN methods and the additional time spent to research discrete sequencing techniques. This delayed the implementation of these methods and practical research into their effectiveness. As a result, the entire project timeline was shifted and accelerated to compensate. However, certain areas of the project were overlooked to meet the new timeframe, these are described in Section 4.3

Over the course of the project, many different implementations for Hawkes processes were looked at. Many of these implementations were either too old to run on newer platforms (due to outdated OS or GPU driver requirements) or did not contain clear enough documentation detailing how to use them. Thus, researching and trialling different implementations for this project until finding one that worked well posed a reasonable time sink. Despite this, the time spent could have been reduced by moving on from unusable codebases earlier and will be taken into consideration for future projects.

A very unexpected and unavoidable event during this project was the advent of COVID-19. Not only did it hamper progress, affecting normal work routines, it also barred multiple opportunities offered as a part of this project. This included travel to CRC related activities and to research labs as a part of the industry partnership. There is no real solution to this issue except persisting, and evidenced by the completion of this report, that is exactly what happened.

## 4.3 Future Work

Resulting from the work described in this report, there are four areas which can be extended.

The first area is in the comparison of two distributions. Throughout the project, comparison of two distributions (namely interarrival distributions) is mainly performed through visual inspection. While this is a decent method for general analysis and

quickly identifying any underlying issues, it has many issues of its own. Visual inspection is too slow to be used in an automated process and requires a human to be present to perform the inspection, preventing the process from being entirely automated as required in the project outline. Additionally, visual inspection provides no hard way of measuring how good the fit is, only that it is 'generally good' or 'generally bad'. Using an error estimator such as mean squared error would be a good step forward, allowing full automation of the analysis process and can be performed quickly at different time resolutions to analyse how well something fits at differing time scales.

Second, some of the results from Hawkes processes looked promising but were not quite good enough for the purposes of this project. When modelling the distribution on the RHS of the split, the EM kernel appeared to learn something, appearing to try and capture the initial peak and the rest of the distribution. Unfortunately, the best model did not have enough resolution to form an accurate model. It is possible this was caused by some technical limitation since more flexible models which in theory should have performed better actually performed worse. Some investigation into this issue might reveal the cause and thus allowing the possibility of creating a more accurate Hawkes model.

Thirdly, the range of network data used in the project is extremely limited, only originating from a single source. Performing the same experiments detailed in Chapter 3 with additional data originating from differing sources would help to verify results. This was not done mainly due to time constraints for the project but additionally due to not having an automated training - analysis loop to help speed up research progress.

Finally, additional research into methods for merging the composite distribution could be performed. Since merging two distributions requires modelling an order-based process (there are no interarrivals for these points, only the order they arrive in), the NN based methods (LSTMs and the transformer) discussed in Chapter 2 would be appropriate for this.

# Chapter 5

# Conclusion

Generation of a model capable of synthesising realistic network traffic would still require additional research into this topic area. However, good progress has been made, forming a working model that captures many important characteristics of the distribution it is given and making some important discoveries into the nature of the data. From this, it is definitely possible to achieve the goal of fully autonomously generating realistic honeypots, as evidenced by the simple but effective composite model, it simply needs more research.

# Bibliography

[BBGP17] E. Bacry, M. Bompaire, S. Gaïffas, and S. Poulsen. tick: a Python library for statistical learning, with a particular emphasis on time-dependent modeling. *ArXiv e-prints*, July 2017.

[Che16] Gang Chen. A gentle tutorial of recurrent neural network with error backpropagation. *arXiv preprint arXiv:1610.02583*, 2016.

[Gra13] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[GSK+16] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[LTP15] Patrick J Laub, Thomas Taimre, and Philip K Pollett. Hawkes processes. *arXiv preprint arXiv:1507.02822*, 2015.

[Nor05] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 2005.

[NW96] Thomas R Niesler and Philip C Woodland. A variable-length category-based n-gram language model. In *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, volume 1, pages 164–167. IEEE, 1996.

[PK11] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 258–267. Association for Computational Linguistics, 2011.

[Ras18] Jakob Gulddahl Rasmussen. Lecture notes: Temporal point processes and the conditional intensity function. *arXiv preprint arXiv:1806.00221*, 2018.

[RIGE16] Hassan Ramchoun, Mohammed Amine Janati Idrissi, Youssef Ghanou, and Mohamed Ettaouil. Multilayer perceptron: Architecture optimization and training. *IJIMAI*, 4(1):26–30, 2016.

[RLMX17] Marian-Andrei Rizoiu, Young Lee, Swapnil Mishra, and Lexing Xie. A tutorial on hawkes processes for events in social media, 2017.

[SBG20] Oleksandr Shchur, Marin Biloš, and Stephan Günnemann. Intensity-free learning of temporal point processes. *International Conference on Learning Representations (ICLR)*, 2020.

[SSL17] Zejian Shi, Minyong Shi, and Chunfang Li. The prediction of character based on recurrent neural network language model. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 613–616. IEEE, 2017.

[Whi17] Ben Whitham. Automating the generation of enticing text content for high-interaction honeyfiles. In *Proceedings of the 50th Hawaii International Conference on System Sciences*, 2017.

# Appendix

## A.1 Table of statistics describing the split in the source distribution

| | |
|---|---|
| Mean Location | 556.4$\mu$s |
| Median Location | 540.0$\mu$s |
| Standard Deviation | 106.1$\mu$s |
| Maximum Value | 828.0$\mu$s |
| Minimum Value | 390.2$\mu$s |

Table A.1: Distribution split statistics

## A.2 Additional graphs demonstrating the split in the distribution from various datasets

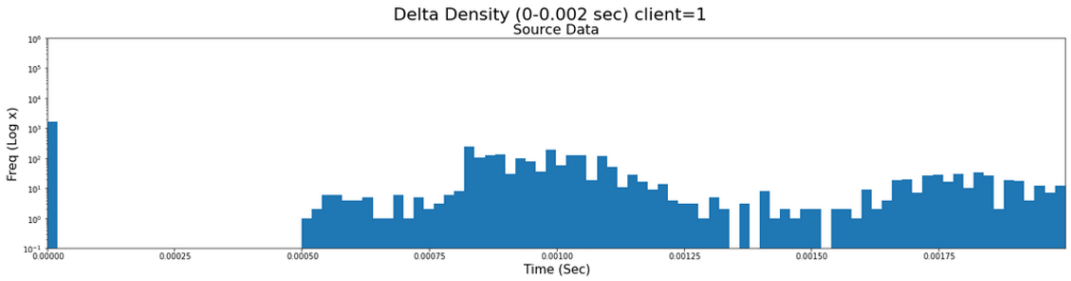



Figure A.1: Interarrival distribution split. From dataset 1.

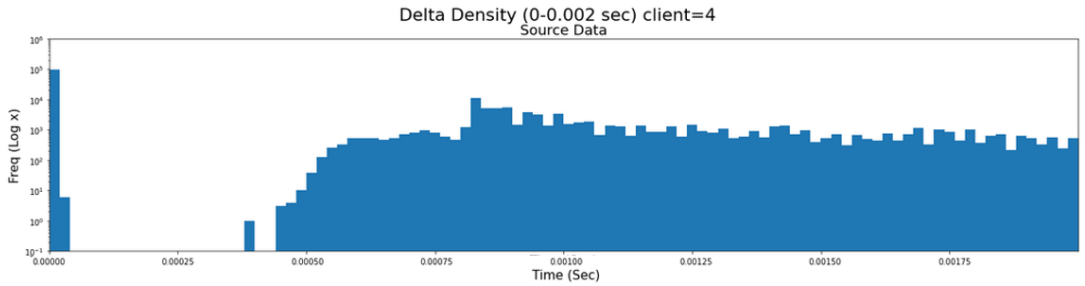



Figure A.2: Interarrival distribution split. From dataset 4.

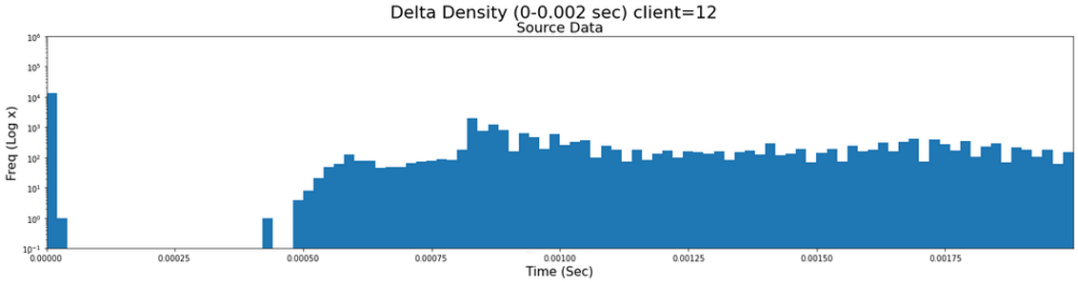



Figure A.3: Interarrival distribution split. From dataset 12.