Machine Learning to Block Trackers Embedded in Apps

Michael C. Black, BA Mod

A Dissertation

Presented to the University of Dublin, Trinity College
in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science

Supervisor: Doug Leith

April 2022
Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

___________________________________________
Michael C. Black

April 19, 2022
I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

______________________________
Michael C. Black

April 19, 2022
Tracking infrastructure is rife within modern networked ecosystems. Identifying that the accumulation of virtual identity within the hands of analytic vendors is not always in the consumers interest, tracker blocking services have gained popular promise. While conventional blocking services have blocklists, this paper proposes an alternative strategy which learns in real time through a multi-armed bandit framework what connections are 'blockable' and what are not. The model learns through the reinforcement learning paradigm whereby beliefs on trackers are updated in response to reward assumed on the actions taken by the model. A number of MAB solutions are subject to analysis, with an assumed suitable model injected into Blokada, an existing open-source tracking firewall.
Acknowledgments

To my mother, twin brother, sister & girlfriend who all mean so much to me

Michael C. Black

University of Dublin, Trinity College
April 2022
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
<td>iii</td>
</tr>
<tr>
<td><strong>Acknowledgments</strong></td>
<td>iv</td>
</tr>
<tr>
<td><strong>Chapter 1  Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Research Questions</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Thesis Structure</td>
<td>3</td>
</tr>
<tr>
<td><strong>Chapter 2  Background and Related Work</strong></td>
<td>4</td>
</tr>
<tr>
<td>2.1 Background</td>
<td>4</td>
</tr>
<tr>
<td>2.1.1 Tracking Across Platforms</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Related Work</td>
<td>6</td>
</tr>
<tr>
<td>2.2.1 Blokada</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Related Projects</td>
<td>7</td>
</tr>
<tr>
<td>2.2.3 Summary</td>
<td>8</td>
</tr>
<tr>
<td><strong>Chapter 3  Framework</strong></td>
<td>9</td>
</tr>
<tr>
<td>3.1 Multi-Armed Bandit (MAB)</td>
<td>9</td>
</tr>
<tr>
<td>3.2 MAB Blocking Firewall</td>
<td>10</td>
</tr>
<tr>
<td>3.3 Firewall Implementation</td>
<td>10</td>
</tr>
<tr>
<td>3.3.1 Real-Time Firewall (Group MAB)</td>
<td>11</td>
</tr>
<tr>
<td>3.3.2 Dual-Stage Firewall (Classic MAB)</td>
<td>11</td>
</tr>
<tr>
<td>3.3.3 Policy</td>
<td>11</td>
</tr>
<tr>
<td><strong>Chapter 4  Simulation Study</strong></td>
<td>13</td>
</tr>
<tr>
<td>4.1 Setup</td>
<td>13</td>
</tr>
<tr>
<td>4.1.1 Assumptions</td>
<td>13</td>
</tr>
<tr>
<td>4.1.2 Editor and Template</td>
<td>13</td>
</tr>
<tr>
<td>4.1.3 Model Requirements</td>
<td>14</td>
</tr>
</tbody>
</table>
# 4.2 Algorithms

- 4.2.1 Classic Policy
- 4.2.2 Group Policy

# 4.3 Evaluation Specification

- 4.3.1 Termination Criteria
- 4.3.2 Optimisations

# 4.4 Path Evaluation

- 4.4.1 Decision Analysis
- 4.4.2 Classic Policy
- 4.4.3 Group Policy
- 4.4.4 Regret Analysis

# 4.5 Output Evaluation

- 4.5.1 Metrics
- 4.5.2 Classification Stats
- 4.5.3 ROC & AUC
- 4.5.4 Summary

## Chapter 5 Implementation

- 5.1 Blokada Specifications
- 5.2 Application Interface
- 5.3 Application Architecture
  - 5.3.1 Rust – Kotlin Interoperability
  - 5.3.2 Interoperability Constraints
- 5.4 Blokada Engine
  - 5.4.1 Initialisation
  - 5.4.2 Blocking Logic
- 5.5 Injection
  - 5.5.1 Implementation
  - 5.5.2 Model
  - 5.5.3 Notification
  - 5.5.4 Authority - Execution Loop
- 5.6 Functional Testing
  - 5.6.1 Functional Experiment Bounds
  - 5.6.2 Functional Output
  - 5.6.3 Observations & Considerations
- 5.7 Limitations
List of Tables

4.1 Termination Criteria Statistics ........................................... 21
4.2 Classification Base Stats .................................................. 29
5.1 Blokada Injected Sample Output ......................................... 44
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>GTS Confidence Termination - Sequential Selections: 5</td>
<td>21</td>
</tr>
<tr>
<td>4.2</td>
<td>GW Confidence - Sequential Selections: 5</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>Proportion Selection, BL - T:150</td>
<td>23</td>
</tr>
<tr>
<td>4.4</td>
<td>BL Proportion Selection, BL - T:1500</td>
<td>23</td>
</tr>
<tr>
<td>4.5</td>
<td>Proportion Selection, UCB - T:150</td>
<td>24</td>
</tr>
<tr>
<td>4.6</td>
<td>BL Proportion Selection, UCB - T:1500</td>
<td>24</td>
</tr>
<tr>
<td>4.7</td>
<td>Proportion Selection, GTS - T:150</td>
<td>25</td>
</tr>
<tr>
<td>4.8</td>
<td>BL Proportion Selection, GTS - T:1500</td>
<td>25</td>
</tr>
<tr>
<td>4.9</td>
<td>Proportion Selection, GW - T:150</td>
<td>26</td>
</tr>
<tr>
<td>4.10</td>
<td>BL Proportion Selection, GW - T:1500</td>
<td>26</td>
</tr>
<tr>
<td>4.11</td>
<td>Cumulative Regret - k: 10, n:0.2</td>
<td>27</td>
</tr>
<tr>
<td>4.12</td>
<td>Cumulative Regret - k: 10, n:0.3</td>
<td>27</td>
</tr>
<tr>
<td>4.13</td>
<td>Cumulative Regret - k: 100, n:0.2</td>
<td>27</td>
</tr>
<tr>
<td>4.14</td>
<td>Cumulative Regret - k: 100, n:0.3</td>
<td>27</td>
</tr>
<tr>
<td>4.15</td>
<td>ROC - k:10, n:0.2</td>
<td>30</td>
</tr>
<tr>
<td>4.16</td>
<td>ROC - k:10, n:0.3</td>
<td>30</td>
</tr>
<tr>
<td>4.17</td>
<td>ROC - k:100, n:0.2</td>
<td>30</td>
</tr>
<tr>
<td>4.18</td>
<td>ROC - k:100, n:0.3</td>
<td>30</td>
</tr>
<tr>
<td>5.1</td>
<td>Blokada Main Screen</td>
<td>34</td>
</tr>
<tr>
<td>5.2</td>
<td>Blokada Activity Screen</td>
<td>34</td>
</tr>
<tr>
<td>5.3</td>
<td>Blokada Package Screen</td>
<td>35</td>
</tr>
<tr>
<td>5.4</td>
<td>Blokada Blocklist Toggle Screen</td>
<td>35</td>
</tr>
<tr>
<td>5.5</td>
<td>Blokada Operation Flow Chart</td>
<td>40</td>
</tr>
<tr>
<td>5.6</td>
<td>Injection Operation Flow Chart</td>
<td>40</td>
</tr>
<tr>
<td>5.7</td>
<td>Notification Generated over Snapchat UI</td>
<td>41</td>
</tr>
<tr>
<td>5.8</td>
<td>Notification within Android Pull Down Widget</td>
<td>41</td>
</tr>
<tr>
<td>1</td>
<td>Regret - Incorrect optimisation set 3</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>Regret - Incorrect optimisation set 3</td>
<td>51</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Tracking has existed since the start of the commercial web in 1989. It has emerged as the superior framework that facilitates a free service model for consumers. Through this model, a symbiotic relationship exists between client and business in which clients give their consent to be tracked. Businesses may utilise this data to streamline user experience and generate profit through more directed advertising. Ostensibly this agreement is mutually understood. However an imbalance of information exists in that clients rarely read or understand the full terms and conditions applying to the arrangement. Applications released by corporations take the form of one-way mirrors, on one side the application interface, but behind the glass, a body continually taking notes. Decades of unravelling this business model, can build substantial databases of virtual identity, likely far in excess of the client’s awareness.

Tracker blocking services intercept trackers before they can be resolved for the associated server. These services are not as trivial as umbrella blocking access from any and all trackers detected, they require more elegant solutions. Most tracker blocking services rely on pre-compiled blocklists for deciding blockable domains. They are operated by privacy enthusiasts in a crowd sourced manner; new domains are continuously appended and redundant entries discarded. A blocking service is only as capable as the underlying blocklists, and as will be discussed, even within the most competitive blockers in the market, the maintenance of such is prone to error. Hashmi et al. find the largest ad blockers “use less than 1% of their rules on the Alexa [F1] top 5,000 websites”. It highlights how inadequate detection and culling leads to coverage and moreover efficiency concerns.

Alrizah et al. aim to expose the reasons behind blocklist pitfalls and vulnerabilities. Their work advocates for the lack of deep understanding prevalent within the maintenance process of blocklists. The study finds false positive classifications stemming from filter list use is non-trivial, and the lifetime of such errors linger in deployment due to contingency weakness. Within the study on this dataset 33.5% percent of error stem from designer
1.1 Motivation

Acknowledging these challenges, this thesis considers how domain detection could be automated through machine learning. The blocking model uses a multi-armed bandit (MAB) framework whereby the proposed model learns of tracker ‘blockability’ through algorithmic blocking of competing resources (trackers). As this framework concerns the field of reinforcement learning there is no reliance on pre-compiled data. Data is supplied dynamically with no assumptions made on any one tracker; all distributions governing tracker ‘blockability’ are initialized equiprobable. This solution uses attentive feedback observed in real time by a user.

The proposed models are tested under varying connection sets and stress levels, induced from the noise associated with human input. The ultimate intention is for tracker classification to have heightened detectability while preserving the accuracy. In this thesis two types of firewall implementation are investigated

1. **Real Time Firewall**: The implementation utilises a grouped MAB approach.

2. **Dual-stage Firewall**: The implementation utilises a classical MAB approach.

Both implementations are considered within the simulation study. Candidate algorithms are explored here ‘within in a vacuum’. Accredited MAB solutions like Upper Confidence Bound (UCB) and Group Thompson Sampling (GTS) are analysed, as well as a custom Group Weighted (GW) solution. The implementation details the materialisation of a dual-stage firewall using an optimised algorithm within an open-source blocking firewall. The android ad and tracker blocker ‘Blokada’ is used in this part of the assessment. Insight is given into the blocking architecture of this application and how existing infrastructure limits the capabilities of assumed blocking logic. A functionality test is carried out in this section involving a user engaging in basic application ‘surfing’, while an injected tracker blocking model is active. The user provides iterative feedback to the algorithm until an assumed session duration has been completed.

1.2 Research Questions

1. Why explore variations of MAB policy?

2. What MAB solutions appear most effective within each policy class?
3. How can machine learning be used to emulate blocking services?

4. How is DNS be used to intercept and alter HTTP traffic?

1.3 Thesis Structure

Chapter 2: Background work and literary review.

Chapter 3: Introduction of MAB framework and its application.

Chapter 4: Simulated study where learning algorithms are explored and evaluated.

Chapter 5: Implementation of studied algorithms within Blokada.

Chapter 6: Conclusions and future work.
Chapter 2

Background and Related Work

2.1 Background

Tracking involves the collection of data on consumer’s engagement with networked platforms. This engagement data is stored in order to build a profile of events that allows algorithms to make habitual, economic and even political inferences on users. Such inferences vary depending on the intentions of the tracking organisation’s level of desired identification, but often includes data like keystrokes, product engagement, location and personal details. Trackers may work explicitly on the apps directly visited or implicitly, by sending data to an associated party of the service currently being used. The distinction between these two is categorised into first- and third-party trackers respectively. Third party trackers operate through first party mobile applications via the developers who embed such technology into the application source code. Under this infrastructure, activity across multiple applications and devices can be linked to a single user.

Identification of an individual is beneficial in that application interface can be shaped towards assumed user preference. Within retail, it may materialise as a lingering advertisement that pushes a pair of shoes a consumer engaged with on a separate application. In more nefarious cases, they have been used to push political agenda, given an assumed demographic. Such use cases have gathered enough notoriety, pressures is now being applied from legislative bodies. ¹

Information gathered from tracking has shown to have a larger footprint than the collecting organisation. Lax data sharing policies exist across the board of analytics partners. Some services claim they do not share information laterally but reserve the right to share within parent and associated companies. For example, Facebook graph API, one of the most persistent trackers uncovered in this study, freely shares data with Facebook

Ads. This considered, the line of consent between clients and tracking corporations is blurred and fractured in many ways. Corporate giants like Google and Facebook have a monopoly on the markets, with smaller enterprises often advertising through their paid service; and maintain that its purpose is to ‘streamline user experience’. This, a fitting surface level alibi for masking the ‘invisible part of the iceberg’ profit associated with storing data.

2.1.1 Tracking Across Platforms

On the web, Chrome, Safari and Firefox all share data on website visits with backend servers on the autocomplete search feature in real time, while Edge and Yandex employ a stricter environment that sends persistent identifiers to servers. Chrome utilises HTTP cookies – data packets that sites send to your computer to be stored in the web browser more freely. This technology allows a browser to remember site settings between visits such as language preferences and authentication. This may be the most familiar encounter we now see with tracking due to consent cookies dialogs blocking access to web pages. This is a legal requirement under GDPR and CCPA governing the implementation methods for receiving user consent. Similar tracking consent is required for apps but can be more obscured in the terms and conditions prior to installation.

Mobile tracking differs in type of information shared by the web and mobile versions of the same service. Tracking infrastructure varies greatly depending on the app ‘family’ category, parent subsidiary company ownership and jurisdiction. Unsurprisingly, the apps that host the most densely embedded trackers tend to be multiple service apps, this typically fits within the bracket ‘Social networks All in One’. Instead of saving data to browsers, apps save data to their root parent companies. For example, the Google, Google API’s, DoubleClick and YouTube apps all report to the parent ‘Alphabet’.

Binns et al find, 73% percent of advertising and tracking services within a studied dataset, retrieved with a trained classifier over the Alexa top 2000 global websites, are owned by Alphabet. It further finds a “high proliferation of cross device tracking, with 39%” of the same dataset with 17 of the top 20 largest tracking organisations having a presence on both the web and mobile ecosystems.

Android has a strict blocking policy imposed by Google through the Android distribution market, Google Play. Many blocking applications never reach this store and must resort to standalone downloads from official sites. Alternatively, there exists specially created catalogues of free and open-source software like F-Droid that have gained popular

---

2Support.google.com
3https://abc.xyz/
4https://adguard.com/en/blog/google-removes-adguard-android-app-google-play.html
prominence due to Google's stern policies.

2.2 Related Work

2.2.1 Blokada

Blokada is an open-source android and iOS blocking firewall that umbrella blocks all ads and trackers on a mobile device. It generates a local VPN to route all traffic on a device to a port where it can be compared against a blocklist. Any domains found on the blocklist are blocked, and all others are routed through to its destination IP. The resulting operation blocks only blacklisted entries keeping only non-advertising or tracking entries showing up on your device.

Within this service blocking works using processes within the Domain Name System (DNS). The DNS is in place so that users are alleviated from the burden of memorising numerous IP address. We use memorable URLs to look up a webpage; through DNS, the web readable record of this domain containing its IP address is retrieved in the background. A DNS resolver, often staged on port 53, is responsible for the tracking down of this DNS record. These records are authoritative instructions that provide information including IP address but also how to handle that type of domain queried. Returned records are typically cached on the local system and intermediate DNS servers to reduce the time and work done to fetch subsequent requests.

When one accesses a server like Facebook, a DNS lookup is performed for that domain. The primary content is served from the web servers owned by the content creator’s domain, e.g., facebook.com, while ad content is simultaneously served from servers on the ad networks domain, e.g., faceebookAds.com. If a VPN is set up to route all traffic seen by the DNS resolver, all server requests on the device could potentially be seen and interacted with here. Ad and tracking servers could be filtered out with all other requests allowed pass through.

Tracker blocking services work as well as they do because most app content providers don’t have the capacity to run their own tracking and advertising networks. Instead, they use third party advertising networks such as Facebook or Google, with the DNS entries in that ad network belonging to a distinct and re-used domain. With ad networks using well known domains, it is then possible maintain a list of these domains (Blocklists) and thus prevent name resolution for them.

Blokada has reputable performance within the mobile blocking community, sought after most because of its universal indiscriminate policy when it comes to blocking. In theory, next to no no-ads should be seen on any interface when using Blokada. Any
ads or trackers that do get through, do so because of un-detection, as opposed to in-access. The only application that occasionally demonstrates performances drawbacks with Blokada activated is Facebook, according to public review. This is primarily due to Facebook’s complex cross site tracking reliance however, many community blocklists still block the relevant domains causing this performance issue due to its excessive tracking implementation.

2.2.2 Related Projects

Tracker blocking is rarely implemented as an isolated service, it tends to be a paired service with advertising, a more coveted market. Advertising blocking services can themselves exist in isolation, especially within machine learning disciplines that use computer vision. Such technologies may rely on visual inspection of a resource (Ad Highlighter). ‘Transparent’ trackers must be detected more overtly, like through filter rule generation.

Within the last few years, studies are surfacing of filter rule generation implementations and the performance when benchmarked against conventional blocklists. Bhagavatula et al trained classifiers to detect ads and trackers URL’s using indicators such as keywords, lexical features, size, or proportion of domain requested for a single resource. The inefficiency of blocklists is explored with reference made to the hyper specificity of regular expressions written by contributing parties to target specific ads and placement schemes. Exposition on unacceptable circumstance is given where RAM can inflate to hundreds of megabytes per page visit. Ground truth and training data split is given by an old and new EasyList blocklist containing largely the same entries but varying classifications. Classification models trained on the old list were tested on the new list. Models trained, especially k-Nearest Neighbour (kNN) were shown to have high overall accuracy and moderate unseen data prediction accuracy. Considering the low dimensionality of data used, the predictability leaves only room for increase in accuracy. A classifier created by Gugelmann detects advertising and analytical domains far more effectively than presently available with an Ad-block Plus blocklist. This study pays attention to HTTP traffic features. It considers both the number of domains requested for a web server and importantly the size of the response, noticing the analytical services sparsity in returning large responses. The classification technique is tested on 24 hours of HTTP traffic recorded at university campus network. 400 additional domains are detected when compared with the reference blocklist, 200 of which verified as traditional advertising an analytical service. Among the remaining, retail and user utility service appear the most common.

\footnote{https://community.blokada.org/t/facebook-never-works-with-blokada/14620/4}
A recent study by Le et al. [10] details a more closely related multi-armed bandit (MAB) inspired ad blocking classifier. This implementation specifies the learning of ‘good’ and ‘bad’ filter rules. Rules are learned of through the reinforcement learning paradigm evolution in response to reward assumed on actions. The full description of MAB framework can be seen in chapter 3. The model deployed in this paper is used in conjunction with perceptual ad detection software for automating the reward process. Relative learning time is scaled down from 13 hours per site, pre-automations software to 1.6-9 minutes per site with the addition of this feature. The rules generated from the classifier are applied over the Tranco Top-5k sites and ultimately blocks with an accuracy roughly on par with EasyList 86%.

2.2.3 Summary

Machine learning classifiers trained on existing training blocklist inherently incorporate a circular dependency into trained models. The model can only be as good as its ground truth. Gugelmann combats this with the deployment of a tapped router to monitor authentic HTTP traffic. The results show a more promising diagnosing classifier.

Another ML discipline adopted in Le’s study showcases the utility of MAB learning and provides inspiration for future partnerships fated for the technology explored in this paper. This solution implemented in this thesis is localised to tracking and as such could not harvest automation with the use image detection, but potentially could use performance measuring software like spike detection to mimic this service. Using a MAB framework, an analysis can be made of immediate pressing tracker from the most primitive principles. No assumptions are made through the use of reliant blocklist training data.

\footnote{https://tranco-list.eu/}
Chapter 3

Framework

3.1 Multi-Armed Bandit (MAB)

The multi-armed bandit framework is an extensively studied problem in reinforcement learning and mathematics. Its classical implementation can be defined:

- There are $k$ arms with unknown reward distribution probabilities $\{\theta_1, \ldots, \theta_k\}$.
- At step $t$, an agent takes an action on one or many arms and receives a reward $r$.
- $A$ is a set of actions referring to the interaction with one arm. The expected reward of action $a$ is given by $Q(a_t)$.
- $R$ is a reward function. At step $t$, is the reward of action $a$ is given by $R(a_t)$.

The task is to iteratively play arms, to gather evidence on underlying distributions with the aim to predict the optimal distribution. It will select an action based on the reward observed from playing arms in previous rounds. The primary metric of measuring effectiveness of a MAB algorithm is regret. It is characterised as the cumulative difference between the reward of the best actions and the agents' actions.

$$Regret(a) = \sum_{t=0}^{T} (maxR(a') - R(a_t))$$

Where $a'$ is the optimal action to have been played at step $t$. This metric gives hindsight analysis, it is nonnegative with the optimal regret being zero. This framework gives rise to the exploration versus exploitation debate – should one explore arms more regularly to avoid overlooking optimal action or should one exploit arms based on prior (possibly naive) knowledge.
Classical MAB returns a single optimal arm as output when the program iterations are exhausted or terminated. The returned arm has been trialled the most, with the principle concern being that its selection induces no regret at any step $t$.

### 3.2 MAB Blocking Firewall

#### Classification

Within a blocking environment the blockability of a tracker is represented using a Bernoulli bandit. That is, all trackers can be described using a binary classification (e.g., 1 denoting a safely blockable connection and 0 denoting an essential connection that mustn’t be blocked). In theory these connection states are non-changing, it can be assumed that blockable connections will never become essential.

#### Data

Reward feedback is given in the proposed firewall through manual, human-in-the-loop, input. This means noise in the form of human error is introduced to connection state. The effect of this is that the same action played on two separate trials can induce different rewards. Thus, the binary classification is specified as a probability distribution which returns 1 with probability $Q(a_t)$ and 0 otherwise.

#### Terminology

It follows from above intuition that the MAB terminology ‘actions’ refer to blocks and ‘arms’ refer to connections in the assumed blocking setup. With all trackers being a distinct state, the blocking model requires identification of all inputs, no one tracker is more blockable than another.

### 3.3 Firewall Implementation

To aid description of both firewall implementations proposed in this paper a blockable subset $b$ is defined. This subset is characterised as the list of all blockable connection within the total list of connections. With $N(b)$ as the cardinality of $b$ it can be expressed $1 \leq N(b) \leq k$. 
3.3.1 Real-Time Firewall (Group MAB)

A solution that incorporates real time functionality is the employment of a grouping MAB policy. For this policy an optimal model is defined as one that converges to blocking the set \( b \). Under this assumption the blocking service continually blocks all trackers received on a device while a user has the service active.

3.3.2 Dual-Stage Firewall (Classic MAB)

This solution details an algorithm that learns with the blocking service activated, similar to MAB solution discussed in Le et al’s [10] work. Within such an implementation a classical MAB policy is deployed. This solution must be terminated and appended to a blocklist before functionality can be expected. This type of implementation matures to block the set \( b \) over a sequential number of rounds \( N(b) \), as opposed to at every iteration \( t \).

3.3.3 Policy

Inspiration in grouping policy is drawn from combinatorial bandits which have similar concept design. Chen [11] et al prove a theoretical combinatorial multi-armed bandit can work achieving the asymptomatic best regret of \( O(\log(n)) \) under its defined constraints. This solution assumes an ‘offline’ \((\alpha, \beta)\) approximation oracle that outputs super arm choice and returns a fraction \( a \) of the optimal expected reward. The framework encompasses both linear and non-linear rewards, e.g. the summation of all individual rewards, or the mean of all individual rewards as the super arm reward.

This thesis, in contrast considers one iteration as one singular feedback response from a user, thus a single binary reward must be returned, indicative only of a breakage in one combinatorial block (if any of all selected arms cause a performance hit). Intuitively, if this simulation were to adapt Chen’s reward system, it would work identically in implementation to a single selection UCB1, just that the calculation of measures would be delayed until the collection of all feedback. Under the single binary reward assumption, the iterative power of the grouped models are limited, that is, there is a cap on information returned per iteration.

The oracles used within grouping strategies in this thesis are curated from informed intuition and testing. In Group UCB and Thompson the measures like upper bound and posterior distributions are normalized between 0 and 1. Normalising the bounds gives the benefit of being able to gauge a relative confidence threshold by which to conditionally block data, i.e., the only the connections in the upper half of the upper bound array may
be blocked. Crucially this gives us a dynamic selection of arms in each iteration, the minimum selection being one arm. Exploration can be encouraged on this oracle through skewing block selections with randomness in either half.

A final group weighted algorithm is detailed in this chapter which does not use the same oracle. It has been curated from scratch and gives an additional algorithm benchmark.
Chapter 4

Simulation Study

4.1 Setup

4.1.1 Assumptions

Ground Truth

The true values of connections are simulated with a random function which is equally likely to make a connection blockable or essential. The ground truth is calculated at the beginning of every simulation, and it is the model’s responsibility to output the same values as this ground truth.

Feedback

Feedback (reward), is simulated through comparison of our blocked values with the assumed true values. Feedback is flagged and triggered as unsuccessful only if a true essential value is blocked, otherwise it is successful.

Noise

Noise introduced in our algorithms is simulated with a random function which when active inverts the feedback returned.

4.1.2 Editor and Template

MATLAB supports matrix operations and provides concise syntax for logical indexing and conditional execution. Connections are represented as vectors, each connection, corresponding to a distinct positional index. Applying an operation means manipulating the positional argument assigned a connection in any function vector.
### Listing 4.1: MATLAB Template

```matlab
ground_truth = random(list 1..k) 
for t = 1:T
    blocked = decide index(list 1..k);
    feedback = flag if different(blocked, ground_truth);
    if (random<n), feedback = ~feedback; end
    if(feedback)
        do something with successful block
    else
        do something with un-successful block
```

Where the following terms are specified:

- **k**: number of connections.
- **T**: max iterations.
- **R**: simulation runs.
- **n**: noise level.

This terminology is used throughout the duration of the simulation. Assuming k = 5, an example debugging snapshot may be

**ground truth**: [0,1,1,1,0]  #connections 2,3 & 4 are blockable
**blocked**: [0,1,0,0,0]  #blocks connection 2
**feedback**: 1  #no performance hit from blocking this connection

Programs may be terminated at the specific iteration that satisfies a confidence criterion. For a distinct number of connections k this metric gives good insight into relative confidence levels of a function but loses practicality when considered in a varied k MAB setup as seen in a blocking firewall.

This thesis primarily considers completion of an algorithm using a maximum iteration threshold T. By pinning a distinct iteration threshold, insight can be given into relative performance of authentic setting when under similar environment conditions; i.e., completion of the blocking service is defined from manual termination by a user de-activating the service. A reasonable value for T is assumed depending on queried environment conditions.

### 4.1.3 Model Requirements

The model should:
1. return all connections in the blockable class (not a subset or superset).
2. return a solution in the fewest iterations possible.
3. surrender the least amount of regret in the learning phase as possible.

4.2 Algorithms

4.2.1 Classic Policy

Baseline (BL)

We begin our analysis by creating a baseline algorithm to act as a benchmark for all models. This algorithm works by cycling sequentially through each connection and calculating the average from the sum of the connections. The average is calculated only when the algorithm iterations are exhausted.

The true value of an action is the mean reward of that action being selected. In practice this can be estimated from the mean of the finite samples of each arm, the larger the number of samples the more accurate our estimation.

\[ Q_t(a) = \frac{r_1 + r_2 + ... + r_t}{N_t(a)} \]

Where \( r_t = R(a_t) \) and \( N_t(a) \) is the number of times action \( a \) has been chosen. By the law of large numbers, as \( N_t(a) \) tends towards infinity our estimation will converge to the true mean. In this way, we can see how the algorithm grows in confidence as the solution matures, but the task lies in reaching a level of confidence on our estimates within an assumed iteration threshold.

Listing 4.3: Baseline

```plaintext
for t = 1:T
    blocked = cycle list (1..k)
    # get feedback & add noise
    if (feedback)
        increase success_count(blocked)
    increase total_count(blocked)

for c = each connection
    output(c) = round(success_count(c)/total_count(c))
```
Upper Confidence Bounds (UCB)

The UCB algorithm is based on the principle of optimism in the face of uncertainty. This means we use the empirically observed data to assign each bandit an arm, called the Upper Confidence Bound which likely is an overestimate of the unknown mean. The magnitude of the over-estimation is dependent on the uncertainty of the distribution. Justification is provided in either scenario for picking the UCB optimistic connection:

- the optimism was justified, and we get a reward which is the ultimate objective.
- the optimism was not justified, and we reduce uncertainty leading us to not choose it in the future.

In UCB, we always select the action which maximises the observed reward average $\hat{Q}_t(a)$ added to the upper bound $\hat{U}_t(a)$:

$$a_t = \arg\max_{a \in A} (\hat{Q}_t(a) + \hat{U}_t(a))$$

Where $A$ is the set of all actions to be taken on $k$ connections. Note without the uncertainty term we are just left with a greedy solution that chooses the observed average reward on each iteration. Potential is thus measured by an upper bound of the reward value $\hat{U}_t(a)$ so the true value is below with bound $Q(a) \leq \hat{Q}_t(a)\hat{U}_t(a)$ with high probability. UCB is a family of algorithms, in this paper we explore UCB1, the upper confidence bound is given by:

$$\hat{U}_t(a) = \frac{c \cdot \log(t)}{N_t(a)}$$

Where $N_t(a)$ is the number of trials or blocks of connection $k$ and $c$ is a hyperparameter specifying exploration rate. Now we can see the action to be taken at any step $t$ is:

$$a_t = \arg\max_{a \in A} (\hat{Q}_t(a) + \sqrt{\frac{c \cdot \log(t)}{N_t(a)}})$$

Each time $a$ is selected $N_t(a)$ is increased, and as it is the denominator of the uncertainty term, the uncertainty term reduces. A similar truth holds for the inverse of this action, when any action other than $a$ is picked, $t$ increases, but as it is the numerator the uncertainty term increases. The use of a logarithm makes impact less severe over time meaning the optimal action(s) will be repeatedly selected as it matures.

**Hyperparameters:** $(c)$ specifies the rate of convergence, this value is defaulted at 2. Lowering $c$ makes the fraction expression bottom-heavy, resulting in a smaller value boosted to each connection’s success ratio. This gives less exigency to the verification
of uncertain connections and thus diminishes convergence time. Increased verification of other connections proves more useful when faced with greater noise.

Listing 4.4: UCB

```plaintext
for t = 1:T
    blocked = maximum(upper_bound)
    # get feedback & add noise
    if (feedback) increase success_count(blocked), end

    success_ratio = success_count / total_count
    delta_i = square_root((c * log(sum(total_count)))
                 / total_count)
    upper_bound = success_ratio + delta_i

    increase total_count(blocked)

output=(round(success_ratio))
```

4.2.2 Group Policy

Group Thomson Sampling (GTS)

The unorthodox logic of this probability matching algorithm was advocated for in paper [13]. As shown the classical version of this algorithm is unsuitable for a multi-output bandit but applying a grouping strategy makes it a notable candidate within this framework. It uses a heuristic which at each iteration it creates a posterior distribution for each of the connections. Within the Bernoulli bandit setup, we can assume \( Q(a) \) follows a Beta distribution, spanning over the interval \([0,1]\). Beta allows us to specify a prior \((\alpha, \beta)\) where we can provide the success count and the failure count. At each iteration \( t \) an action is selected according to the probability it is optimal:

\[
\pi(a|h_t) = P[Q(a) > a'), \forall a' \neq a|h_t]
\]

\[
E_{(R|h_t)}[1(a = argmaxQ(a)]
\]

Where \( \pi(a; |; h_t) \) is the probability of taking action a given history \( h_t \). At each iteration \( t \) the posterior distribution of \( k \) is calculated from the empirically observed prior. Then a randomly generated sample is chosen from this distribution. The subsequent action selected by the model draws only from the list of extracted samples. Thus, this process
draws from the principles of Bayesian inference, with each iteration the probability for a hypothesis is updated as more evidence becomes available.

Exploration originates from the randomization of the sample. When the algorithm is in the early stages and the prior is weakly informative, the variances of the calculated distributions are large and thus the fluctuations in the samples extracted will be too. As more data is accumulated the posterior distributions converge towards the true environment and this rate of exploration decreases. Give time complexity

**Hyperparameters:** (h) The randomness stemming from the posterior is sufficient to encourage healthy exploration, so no manual addition of randomisation is needed in the oracle. A hyperparameter can be tuned on the threshold by which blocked connections are selected. All values below this threshold are blocked without randomness.

```
Listing 4.5: GTS

Initialise k connections: blocked = cycle list (1..k)

for t = 1:T
    blocked = rand(1..k) < norm_sample
    # get feedback & add noise
    if (feedback) s(blocked) = s(blocked) + 1
    else f(blocked) = f(blocked) + 1, end

    success_ratio = s / total_count;

    samples = betarnd(1+s, 1+f);
    norm_sample = (samples - min(samples))
                 / (max(samples) - min(samples));

    increase total_count(blocked)

output=(round(norm_success_ratio))
```

**Group Weighted (GW)**

This algorithm has simple intuition relative to other explored model but can be effective in certain scenarios. It assigns only a weight for each connection, no other metrics are needed within the base algorithm.

- The weight of connections are raised to 1 when a connection is successfully blocked.
The weight of connections are lowered to some percentage $l$ of itself when any connection within a selection is flagged as an unsuccessful block.

At iteration $t$ the blocked connections are decided from a random vector that blocks all connections under the maintained weight. The blocking logic regularly blocks all blockable connections but sometimes also blocks essential ones. Towards the end, generally at least $n$ connections are blocked where $n$ is the number of blockable connections.

**Hyperparameters:** $(l)$: This solution anchors successful blocks with a value of 1, making it certain to be selected on the subsequent round. The lower coefficient denoted $l$ can be tuned to impact rate of convergence.

```
for t = 1:T
    blocked = rand(1,k)<weight;
    # get feedback & add noise
    if (feedback)
        weight(blocked) = 1
    else
        weight(blocked) = weight(blocked) * l
output=(round(weight))
```

### 4.3 Evaluation Specification

This evaluation is positioned to mimic the firewall application as closely as possible. The intention is to highlight model inner-workings and optimise performance (where possible) under distinct simulated bounds. It addresses that a model’s efficiency is assessed using iterations. Run time is a poor metric of efficiency, having unreliable output dependent on variables such as the OS and CPU load. The bracket of uncertainty, especially within smaller simulations, deems the utility of this tool impractical. In addition, run time would be an unreasonable measure when translated to a practical setup involving a human entity.

To this end, the use of termination criteria is considered to halt the models at a confidence level in as few iterations as possible. Minimising iterations, however, needs to be cognisant of the error rate. A model could be tuned to empirically minimise both iteration and error rate however true tuning cannot be achieved without the anchoring of one measure (or combination of it). If attempting a joint optimisation, this usually
necessitates compromises e.g. we would have to settle for an error rate higher than the minimum possible error rate given that there is also an iteration-minimising constraint to account for. In addition, a tuned algorithm may be a reflection more of the termination criteria than the properties of the model. Therefore, the decision is made to reflect primarily of the properties of a model with either iterations or error pinned, optimisation can be logically introduced from here.

Through pinning iteration threshold T all models can be compared under fair grounds in the decision process and performance evaluations. The resulting evaluation has similar environment bounds to the practical blocking firewall experiment whereby termination is controlled by the user through activating and deactivating the service at an arbitrary iteration.

A competitive iteration threshold is picked for the basis of our evaluation with the intention to optimise model accuracy at this point. A confidence bound terminator is imposed on the models to justify a competitive value of the iteration threshold. Thus, this evaluation optimises error. The evaluation to follow consider performance of models at the following environment conditions:

- Reduced connections and iteration set: $k:10, T:150$
  - Moderate noise level: $n:0.2$
  - Moderate noise level: $n:0.3$

- Increased connections and iteration set: $k:100, T:1500$
  - Moderate noise level: $n:0.2$
  - Moderate noise level: $n:0.3$

### 4.3.1 Termination Criteria

Termination criteria are shown in this thesis but are not considered within the optimisation process due to above considerations. Termination criteria detail that a model can be halted prematurely in response to satisfaction of a confidence-level based criterion. Such a feature is useful with a distinct number of connections $k$ and can give insight into model diagnosing ability. A termination condition is dependent on model properties and therefore needs be custom made for each model.

### Sequential Selection

An example criterion is specified whereby the program ends given the model has selected the same actions 5 times in a row. Increasing this value boosts the confidence level before terminations but too, increases iterations.
4.3.2 Optimisations

All algorithms are optimised in a consistent manor, they make use of MATLAB’s Global Optimization Toolbox \cite{f6} for one- and two-dimensional minimisation algorithms. Relevant hyperparameters are tuned using Pattern Search. The search can be restricted to a single plane so optimisation can be used for both univariate and multivariate algorithm hyperparameters.

Parameters are tuned so as to minimise the error rate of an output. The base analysis considers the reduced connection and iterations set with moderate noise level. Using these conditions optimisations are tuned to maximise f1 score. It returns the following.

\[ \text{ReducedSetOptimisations} : (UCB)c : 2, (GTS)at : 0.8, (GW)l : 0.6 \]

This search shows that the difference in optimised hyperparameters, when tested across noise levels, has insufficient impact from the base optimisation set. Within the increased connection and iteration set, the grouping algorithms benefit from tweaks to hyperparameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mean</th>
<th>Std</th>
<th>Flagged Error</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GTS</td>
<td>199</td>
<td>66</td>
<td>0.27</td>
<td>0.9667</td>
</tr>
<tr>
<td>GW</td>
<td>150</td>
<td>88</td>
<td>0.64</td>
<td>0.90172</td>
</tr>
</tbody>
</table>

Table 4.1: Termination Criteria Statistics
4.4 Path Evaluation

4.4.1 Decision Analysis

Proportion selection highlights the continued decision policy upheld by a model. It is given by the formula:

$$\sum_{A \to \infty} \left[ \frac{N_t(a)}{N_t(A)} \right] \approx 1, \forall a \in b$$

Where $N_t(a)$ is the number of times action $a$ has been chosen and $N_t(A)$ is the number of all actions chosen (synonymous with number of iterations $t$). Being a proportional measurement, this metric ignores the rate of blocking. A grouped policy would have greater number of actions per iteration $N_t(a)$ for any action $a$ but proportionally the total number of actions $N_t(A)$ would inflate with it. Thus, similar numeric outputs are given for each policy type.

Whether a group or single iteration strategy imposed, an ideal model should uphold the following:

Where $b$ equal to the total number of blockable connections. Given that this measure considers the complete history of actions, it will tend towards this value but never fully reach it. Within a grouping policy it is typical to see all blockable connections tending to the same proportion, $1/b$, all being selected as a group. Within an intelligent single selection policy (not baseline) the spread doesn’t necessarily converge, but should still abide by the formula above[A1]. In general, blockable connections should be selected more than $1/k$ of the time and essential connections should fall under this value.

Ground truth of connections are indicated through a colour scheme. Blockable connections have been assigned the summer colour pallet, while essential connection have been assigned the winter pallet. Ground truths vary among short and long iteration graphs. Legend syntax: C1:[P:1] indicates connection number C and the corresponding binary prediction value P (1: true, 0: false) after exhaustion of all iterations.

In the below analysis the following the reduced connection set is analysis with standard iterations T:150 and the elongated T:1500 to perceive how each model trends.
4.4.2 Classic Policy

Baseline

This strategy makes no decisions in response to reward received at each step. It is ‘unintelligent’ in this regard, the reward received for action $a_t$ bares now impact on how the algorithm progresses. Assuming a dual stage blocking model, this solution would guarantee all arm convergence on to the true values from the estimates as $t$ reaches a high number. All arms are explored, but inefficiently and with a dense regret path. This model scales to $1/k = 1/10 = 0.1$, as each tracker is selected an equiproportional amount of times.

Figure 4.3: Proportion Selection, BL - T:150

Figure 4.4: BL Proportion Selection, BL - T:1500
Upper Confidence Bounds (c:2)

This model has precise systematic update to each tracker. Note the path taken is more volatile, owing to the juggling of all blockable connections while limited to a single rate of blocking. Connections showing the most promise are played in rounds, due to the regular boosting of connections slipping out of relevance. Unpromising connections are also subject to this boost, however often it is not sufficient to be selected for action $a_t$. Here the five blockable connections scale to 2 with a moderate span width.
4.4.3 Group Policy

Group Thompson Sampling (at:0.8)

Group Thompson Sampling is less conscious of unpromising trackers, it has a slightly stricter selection policy, making it a more suitable candidate for group policies. The algorithm’s smoothness comes from its reluctance to experiment after a certain maturity. This reluctance may hinder the solution if a tracker is yet to be correctly identified up until this point. The elongated graph shows blockable trackers selection proportion tends to just under \(1/b = 1/4 \approx 0.25\).
This grouping algorithm ignores an initialisation sequence, unlike GTS. This model takes relatively longer to diagnose classes but acts reliably over long sequences of iteration. Corrections can be found later in the algorithm lifespan. The elongated model tends to just under $1/b = 1/5 \times 0.2$.

### 4.4.4 Regret Analysis

Regret analysis places the max iterations $T$ at a point where the solution will likely have converged. Regret is graphed over one simulation run, thus there is greater variation between runs. Standardisation is then needed across models for the ground truth used to combat the inflation of inherent randomness between models over an extended range.

Within combinational frameworks as specified in Chen [11] et al study combinational regret is weighted as the $a.b$ fraction of optimal reward. It justifies this by specifying how regret is bound by the reward encompassed in the function. This thesis assumes the same reward potential in each iteration and as such assumes the same regret scale as classic policy. Within one iteration, both strategies are bound to the same amount of potential regret.

It is, however, important to clarify that the grouped solutions may systematically induce more regret owing to the greater throughput of connections. The logic of grouped solutions facilitates more mistaken predictions. When any of the connections in a super arm is wrongly predicted, all are marked as incorrect.

We examine the cumulative regret over the reduced and increased connection sets.
k:10, k:100 for elongated iterations.

Figure 4.11: Cumulative Regret - k: 10, n:0.2

Figure 4.12: Cumulative Regret - k: 10, n:0.3

Figure 4.13: Cumulative Regret - k: 100, n:0.2

Figure 4.14: Cumulative Regret - k: 100, n:0.3

The baseline performs consistently poorly in all variations of the environment, adapting the same search path on any simulation. Note the variation of regret between runs of the same connection set. Any discrepancy here amounts to randomised variation in ground truth. The variation experienced by the baseline is also experienced by other models. Optimisation helps to tailor responses between reduced connection sets, (see how blanket optimisation performs when applied to all environment conditions App.A.1, A.2). UCB and GTS perform and scale similarly. Grouped strategies stand out as more impressive considering multiple selections in a given iteration are more susceptible to inducing regret. GW receives greatest negative impact from noise. This may be owed to the fact
that the GW logic anchors successful blocks to a maximum weight, as opposed to other algorithms which give only minor percentage updates of maintained probabilities. The potential work in a GW solution to get a misidentified connection back to a fair value, may thus misguide the solution.

4.5 Output Evaluation

4.5.1 Metrics

An output can have four possible classifications after completion of a simulation:

- **True Positive (TP):** A blockable connection predicted blockable
- **False Positive (FP):** An essential connection predicted blockable
- **True Negative (TN):** An essential connection predicted essential
- **False Negative (FN):** A blockable connection predicted essential

The calculation of terms utilised in this study are derived here by means of the classification classes expressed above. Accuracy denotes how close a given set of measurements are to their true value. Observational error can be derived from the inverse of this metric.

\[
\text{Accuracy} = \frac{TP + TN}{P + N}
\]

Sensitivity refers to the true positive rate (TPR) of blocks, i.e., the probability of a positive test conditioned on it being positive. Specificity refers to the true negative rate (TNR), i.e., the probability of a negative test conditioned on it being negative. The latter’s importance should be noted. The inverse of the TNR is the fall-out rate (or false positive rate (FPR)). This describes the unwanted scenario when the output falsely predicts an essential connection as blockable. In circumstance like this, lasting performance hits will occur on applications which is highly undesirable.

\[
\text{Sensitivity} = \frac{TP}{TP + FN}, \quad \text{Specificity} = \frac{TN}{TN + FP}
\]

The F1 Score is the harmonic mean of precision and recall is an abstracted take of an algorithm’s overall performance. This metric is used for optimisations within the evaluation. Within this classification analysis, even if the f1 score is slightly less desirable, high merit should be given to algorithms with large specificity.

\[
\text{Precision}(PPV) = \frac{TP}{TP + FP}, \quad F1 = 2 * \frac{PPV * TPR}{PPV * TPR}
\]
The F1 Score is the harmonic mean of precision and recall is an abstracted take of an algorithm’s overall performance. This metric is used for optimisations within the evaluation. Within this classification analysis, even if the f1 score is slightly less desirable, high merit should be given to algorithms with large specificity.

$$\sum \frac{1(output \neq groundtruth)}{R}$$

Where R is the simulation runs. This metric gives is a brute estimation indication of improper solutions.

### 4.5.2 Classification Stats

The classification statistics are generated for each model for the basic reduced connection and noise set. Further analysis is given across increased connection and noise set in ROC analysis below.

<table>
<thead>
<tr>
<th>Algo</th>
<th>FE</th>
<th>ACC</th>
<th>SEN</th>
<th>SPEC</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL</td>
<td>0.07</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>UCB</td>
<td>0.08</td>
<td>0.99</td>
<td>1</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>GTS</td>
<td>0.32</td>
<td>0.95</td>
<td>0.94</td>
<td>0.96</td>
<td>0.95</td>
</tr>
<tr>
<td>GW</td>
<td>0.82</td>
<td>0.72</td>
<td>0.65</td>
<td>0.8</td>
<td>0.71</td>
</tr>
</tbody>
</table>

Table 4.2: Classification Stats n:0.2 - FE: flagged error, ACC: Accuracy, SEN: sensitivity, SPEC: specificity, F1: f1 score

UCB performs equally well as the baseline solution. It handles noise comfortably enough under all brackets. This comfortability is tested under further higher noise stress shown in ROC analysis.

Group Thompson Sampling (GTS) performs the best out of the grouped strategies but underperforms either of the single strategy solutions.

### 4.5.3 ROC & AUC

A Receiver Operating Characteristic (ROC) curve analysis is given to illustrate the diagnostic ability of each model after terminations beyond assumed thresholds. The scores for ROC curves are gathered from collected probabilities output before they are classified as a binary output. Poorer models have a straight line from either corner of the graph, indicative of no class separation. Better models hug the upper right corner, indicative of greater class separation.

It is calculated from running the simulation 100 times
The performance of the ‘unintelligent’ baseline algorithm is striking under these bounds of evaluation. It performs near perfect. Intuition tells us it is likely that an algorithm which pays equal attention to all connections will have the fairest probability estimate of each. Thus, the result tends to be on par with the top-grade solutions when considered in this light. The accuracy is the benefit of employing such a taxing unintelligent decision path. As can be seen through UCB less costly paths can achieve a similar accuracy.
Grouped Policy

GTS has impressive AUC given this solution adapts a grouping policy. It benefits the most from optimisation tweak, meaning the algorithm may not be very adaptive across environment variation.

4.5.4 Summary

This simulation examined a snapshot of various model with a policy divide, when implemented in a multi-output Bernoulli MAB framework. This analysis shows the classical UCB1 as the optimal canditate for a blocking when implemented as a dual-stage service. It is shown how classic policy MAB has somewhat of an advantage over grouped policy in the given reward restrictions detailed. The output performance of candidate can have great predictive power even if the the algorithm lacks wit but unintelligence has a cost, namely regret.

Grouped policies grant the ability to funcion in real time and as so offer undeniable advantage over the dual stage implementation, however this feature comes with a relatively slower converagence realisation. Greater optimisation is needed for grouped policy when applied over increased connection set and noise level. As such there may be an added overhead resulting from research and optimisation in this light.
Chapter 5

Implementation

This chapter details the implementation of a dual-stage UCB blocking model into an existing tracker blocking firewall. The model is injected into the engine of the destination firewall and utilises the existing networking architecture to host the model. This thesis uses the android application Blokada.

5.1 Blokada Specifications

Blokada slim is a lightweight version of the applications only containing features allowed by Google Play policy and the only version of Blokada installable from the Play store. It does not allow real time ad blocking and in-app payments, this version blocks through changing DNS servers which works similarly to the full application but is less configurable. Changing DNS server choice in slim mode may thus cause functionality problems. Blokada have included the ability to upgrade to its blue-chip model Blokada Plus in app but requires installation and payment from its linked official sources. Plus mode includes a VPN service using WireGuard to hide a user’s IP address and encrypt networking activities. Other versions include the iOS equivalent Blokada 6, which is fully available for download from the apple store and Blokada Cloud, a faster cloud-based solution that can be used on any platform that supports DNS over TLS/HTTPS (Windows, MacOS, Linux).

Blokada has the blocklist OSID\(^1\) active by default, a general purpose blocklist with an emphasis on functionality over blocking. It highlights Goodbye Ads, Phising Army, and DuckDuckGo Tracker Radar but has the additional selection of a greater span. It supports additional functionality of exceptions or whitelisting, a service that inhibits the blocking of certain applications or websites that should never be blocked. Whitelisting

\(^1\)https://oisd.nl/
has gained popular prominence in recent years and as such public opinion necessitates its existence in commercial blocking services. It acts as a way of expressing gratitude toward a particular organisation, whitelisting their domain means organisations will generate the full revenue from advertising endeavours.

The current split of internet protocol means firewalls such as Blokada need offer support for both available versions. IPv4 has given rise to the newer IPv6 owing to the rapidly growing scale of the internet. The 4.3 billion addresses offered from the a 32-bit address space of IPv4 is upgraded to an 128 hexadecimal address space. A study carried out by Kuebris et al [12] proves ipv6 adaption is moving at a slow pace, however. It denotes only 26 of 215 (12%) economics measured has increasing levels of IPv6 capability over a three-year study period. Within Rust, DNS support is given through A and AAAA record types respectively.

This thesis uses the official Blokada repository\textsuperscript{2} for android and iOS available from GitHub. Four distinct flavours are specified from within the build Gradle file:

- **Full**: build lands on Blokada.org
- **Google**: build lands on google play
- **Escaped**: can be built on top of google to get all features
- **Droid**: lands on of F-Droid

Droid is sufficient for the implementation of this experiment, using this flavour lands us with a standard libre account.

\textsuperscript{2}\url{https://github.com/blokadaorg}
5.2 Application Interface

Figure 5.1: Blokada Main Screen

Figure 5.2: Blokada Activity Screen
5.3 Application Architecture

Having been released in 2016, Blokada is built using the newest technology stacks Kotlin and rust. This pairing is part of an increasingly popular development tactic to extract concurrent and sensitive logic to lower-level languages while keeping main implementation residing in higher-level frameworks. Doing so with rust can squeeze extra performance out of an application as well as boost the security with memory safe logic. It additionally allows for the engine to be abstracted from platform specific frameworks. Within this repository Android uses Kotlin and iOS used Swift. When running either platform the engine is compiled into a native library and accessed through a Java Native Interface.
5.3.1 Rust – Kotlin Interoperability

Rust and Kotlin interoperability works through exposing the C language. For android, a Native Development Kit (NDK) allows the use of this. A cargo extension ‘Cargo NDK’ handles all environment configuration needed for building libraries from a rust codebase. It provides neat implementation for specifying the correct JNI libs directory structure. A shell script file ‘build.blocka.sh’ groups all cargo NDK instructions in one place, every time a change is made to the rust engine this script needs be re-run. Running this builds the specified tool chains and platform binaries into a JNI library directory within the android application.

When built, functions can be called from the native library to either trigger execution of a process or pass data between languages. Data transfer is bi-directional between Kotlin and Rust, but data must first be converted to that of its c implementation. There are supporting tools for C data types on either end, libC for rust and Cinterop for Kotlin.

5.3.2 Interoperability Constraints

Kotlin multiplatform uses compilations for producing artifacts. These compilations can interact with native libraries thus interoperability is often configured within the cinterops block of the compilation in the build Gradle. Since Blokada opted, when being developed, for using swift as the iOS language, the build system has no need for a multiplatform target. Without the use of a multiplatform project the uses of the interoperability feature cinterop is removed. Thus, any composite data types such as strings or lists cannot be read from within the Kotlin language. Only primitive types, which are inherently mappable to its equivalent Kotlin primitive type (e.g. C: int to Kotlin: Int) can be understood. This assumed, the communication from Rust to Kotlin (not Kotlin to Rust) is much more barebones and tedious. Though the documentation describes the potential workarounds of using cinterop like POSIX for Linux, a design choice was made to communicate via primitives rather than learn of this operating system interface.

Within Blokada version 5’s Rust engine, a cache maintains a history blocking activity carried out in the Rust engine. Where this cache is linked with the stats service within the iOS platform, no such connection exists for android. The path of information flow in the android platform is thus unilateral, only from Kotlin to Rust. Perhaps this can be explained by the incompatibility reasoning run into above.

Instead of receiving stats of blocking activity from the Rust engine, the system runs in parallel from within the android application. Primary tunnel configuration and socket engineering is evoked from the application engine. From viewing logs, it can be seen that new domains are intercepted here simultaneously to the detection from within the
rust engine. Blocking functionality happens from within rust while interface updates are serviced from the application. This becomes an inescapable issue within this thesis and dictates why prevented domain resolution stemming from our model in the rust engine don’t affect the output on the android interface.

5.4 Blokada Engine

5.4.1 Initialisation

Upon activating the blocking service, DNS proxy constraints are specified within Kotlin and passed into the Rust engine for initialisation. Data is passed through the exposed c interface through the methods described above. From Rust the server is bound at specified listen address 127.0.0.1 + port 53 with the runtime library Tokio. Relevant DNS services are initialised at this point with the Trust DNS library. All sockets connecting to this address are processes concurrently on separate threads.

When the service is deactivated from within Kotlin a 'close_dns' function receives a handle of the operating engine processes: threads, runtime, resolver and cache and frees the allocated memory.

5.4.2 Blocking Logic

Primary event logs from when a connection is detected to it being routed:

Listing 5.1: Rust Engine Logcat Output

```
1. trust_dns_server::ser:: received udp request from: 127.0.0.1:48203
3. trust_dns_server::aut:: query received: 41172
4. trust_dns_server::aut:: searching authorities for: mtalk.google.com.
5. trust_dns_server::aut:: searching authorities for: google.com.
6. trust_dns_server::aut:: searching authorities for: com.
7. trust_dns_server::aut:: searching authorities for: .
8. trust_dns_server::aut:: request: 41172 found authority: .
```
DNS requests connect to the proxy through ephemeral ports like seen in the logcat extract (line 1). These ports are temporarily allocated for the session duration and discarded after. The same port may be used for a separate DNS request name thereafter.

When a request is received from UDP, Trust DNS performs a recursive lookup to return the relevant authority. (4-8). Blokada implements an authority to be used at this point, which forwards resolution to upstream resolvers. Within this authority, before resolution, Blokada uses this opportunity to now query the domain against a blocklist. Here Rust has access to domain requests in the form of Trust DNS client queries. Trust DNS hosts this query utility for ease of communication with client, meaning raw DNS messages are not needed in communication. Queries are formatted structs used for looking up resource records.

The architecture within Blokada’s authority, importantly specifies a Rust trait that implements a function ‘next_action()’, i.e. next blocking action. This trait is applied to all blocklist related services including an internal cache.

Blokada hosts a Least Recently Used (LRU) Cache within, to first verify if the tracker has been seen before. Rust implements this data structure operating at O(1). It stores the domains name as a key and with the blocking action, Unix time and number of requests as its pair value respectively. If queried domain is present in the cache it simply mimics its previous block action stored and updates associated details. If non-existent a block action function propagates through a sequence of conditional checking like if the user has whitelisted/blacklisted a domain and subsequently though all blocklists active.

When the search is completed and the cache update, the authority can perform a lookup on the domains nameserver depending on the block action. In the above extract the domain was not found in a blocklist (and is therefore granted resolution). It is passed into a new forward lookup function with necessary information to resolve it, such as the

---

3[https://github.com/bluejekyll/trust-dns](https://github.com/bluejekyll/trust-dns)
Trust DNS query, record type and resolver (line 11/12). Access to the resolver is guarded under a mutex (line 13/14) to avoid hitting the memory limit. Resolver type depends on relevant record type (e.g. A for IPv4 or AAAA for IPv6) (line 12), but when it is found the record can be returned. Finally, the browser can make a HTTP request to the returned IP. (line 16)

```
11. trust_dns_server::aut.: handling forwarded resolve: 50474
12. trust_dns_server::ser.: response: 50474 response_code: 0
```

Where the search returns a match found in a blocklist a separate path is taken (starting line 11). In this case the authority performs a variation ‘blocked lookup’. whereby the Trust DNS query is assigned a time to live (TTL) value of 10 seconds from the current Unix time as well gives the record an UNSPECIFIED address. In IPv4 and IPv6 this corresponds to the address 0.0.0.0 and 0.0.0.0.0.0.0.0.1 respectively. With this record the browser will make a HTTP request to the hole address returned.

### 5.5 Injection

The implementation of classic policy UCB is detailed below.
5.5.1 Implementation

![Blokada Operation Flow Chart](image1)

![Injection Operation Flow Chart](image2)

5.5.2 Model

Model functionality is handled with universal state management from Rust class-like structures. The progressive stages of a given iteration as seen in the simulation study are implemented over of number of methods, to be callable from concurrent socket processes within the authority. The most important of such include:

The model’s functions above are accessible from each authoritative server thread. Instead of examining the cache, like within the original service, the first action is to store the queries received and query the UCB model for a domain to they can be blocked and unblocked from the authority.

- **Connection Addition:** Trust DNS queries are stores within a vector so as to be retrievable through an index, due to the rust-kotlin interoperability constraints explored above.

- **Connection Selections:** Adopts a UCB policy for deciding connections.
• **Model Value Updates:** Upon receiving feedback measures like upper confidence bound need be updated at every iteration.

### 5.5.3 Notification

![Figure 5.7: Notification Generated over Snapchat UI](image1)

![Figure 5.8: Notification within Android Pull Down Widget](image2)

Feedback is collected through an android notification service. Existing notification prototypes existed within Blokada in the form of blocking updates and expiry notices. It was possible to utilities these for creation of a separate algorithm notification prototype. This channel is given max priority so as to eliminate any delays in feedback of the model.

Raw communication from the rust engine is materialised from within the stats service. Here the Blokada DNS JNI is regularly queried to check for feedback request. In the case of a request only an index is transmitted from the engine. An internal stats list maintained from the android side of the engine containing an identical stats list existent from within the engine is indexed upon and the domain in question is retrieved. When a domain is retrieved from the stats service the algorithm notification is triggered.

An algorithm service class need be set up to handle broadcast reception and forwarding of received data. Functions within this class handle communication from stats services that should have the desired domain. From the notification, an iteration (maintained
from android side) and domain is displayed. The users yes or no feedback is receiver through an algorithm android broadcast receiver and passed back to the Blokada engine. It suffices to send back only the binary reward response, the Blokada engine maintains a blocked queue which is pop which each iterative response.

When feedback is out for query, all algorithm operations are paused from within the blocking engine using a waiting state Boolean. When active this prevents unblocking services from hastily unblocking connections waiting on feedback. Only when feedback from Kotlin is detected is the state variable updates and algorithm blocking is continued.

5.5.4 Authority - Execution Loop

Since sockets are handled concurrently, any shared data resource need be safely accessible and lockable from within these processes. Any algorithmic of fetching resources are thus protected with Arc’s (Rust’s implementation of thread safe reference counting pointer) and mutexes. During the initialisation process we specify this protection and pass both values into the authority server. This means all concurrent operations form within the authority can access shared universally values like algorithm metrics and static feedback requests.

The primary execution loop is thus carried out from various thread within each authority. This means progression of the algorithm relies on continual request from domains. An intervaled progression loop can be unreliable but from empirical analysis the rate of DNS requests received is still many times faster than the rate at which feedback can be processed. When controlled browsing takes place there can be anywhere up to 30 requests a minute, or 1 request every two seconds. This solution is very much makeshift but is sufficient for a proof-of-concept implementation.

It also means we ‘hijack’ the domain intended to be performed blocking or name resolution on. We store the domain of the current request to the shared algorithm resource and churn out the UCB models’ decision for connection to be acted upon. All queries churned out form the algorithm are blocked. When blocked, all state variables can be updated to notify the algorithm a connection is out for feedback. During the prolonged time period, a design decision was made to process received connections as normal. This means when some connections are out for feedback, Blokada operates as normal on all DNS requests received in the meantime. This is also beneficial as non-resolved connections can interfere with our results through related connections. Noise may also appear in the form of connection requests that re-emerge and interfere with the current query.
5.6 Functional Testing

5.6.1 Functional Experiment Bounds

1. A domain is blocked via the injected model once the blocking service has been activated.
2. Upon blocking the user is identified via a notification specifying the blocked domain.
3. The notification queries the performance of the specified domain, allowing the user a binary ‘yes’ or ‘no’ response.
4. The user returns to the specified application to gauge performance and responds to the query.
5. After response the algorithm progresses and repeating from step 1.
6. The user should continue browsing between feedback as to evoke as much DNS requests as sufficient.
7. The experiment continues until an assumed iteration threshold has been met at which point the service can be deactivated.

5.6.2 Functional Output

Without engaging in a controlled experiment, sanity checks can be carried out on the functionality of the model. Note these results demonstrate application functionality only and should not be considered as a controlled classification.

5.6.3 Observations & Considerations

From multiple re-runs of the above functionality experiment, the following considerations arise.

Accuracy

Investigations of performance hits must be thorough owing to the following considerations:

- **Shutting down Application**: Applications need be refreshed every time a new domain is blocked for the networked content to be the true interpretation. In certain cases, however applications need be shut down for performance hits to be displayed, due to application network state linger.
<table>
<thead>
<tr>
<th>Domain</th>
<th>SB</th>
<th>TB</th>
<th>SR</th>
<th>UB</th>
</tr>
</thead>
<tbody>
<tr>
<td>scontent-dub4-1, cdninstagram, com</td>
<td>2</td>
<td>3</td>
<td>0.6666</td>
<td>2.4919</td>
</tr>
<tr>
<td>edge-mqtt, facebook, com</td>
<td>7</td>
<td>7</td>
<td>1.0</td>
<td>2.8243</td>
</tr>
<tr>
<td>mqttt-mini, facebook, com</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>1.8287</td>
</tr>
<tr>
<td>in, appcenter, ms</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>1.8287</td>
</tr>
<tr>
<td>chat-e2ee-mini, facebook, com</td>
<td>2</td>
<td>3</td>
<td>0.6666</td>
<td>2.4919</td>
</tr>
<tr>
<td>mtalk, google, com</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>1.8287</td>
</tr>
<tr>
<td>alt2-mtalk, google, com</td>
<td>4</td>
<td>5</td>
<td>0.8</td>
<td>2.6246</td>
</tr>
<tr>
<td>connect, facebook, net</td>
<td>6</td>
<td>6</td>
<td>1.0</td>
<td>2.8244</td>
</tr>
<tr>
<td>graph, facebook, com</td>
<td>6</td>
<td>6</td>
<td>1.0</td>
<td>2.8244</td>
</tr>
<tr>
<td>web, facebook, com</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>1.8287</td>
</tr>
<tr>
<td>graph, instagram, com</td>
<td>6</td>
<td>6</td>
<td>1.0</td>
<td>2.8244</td>
</tr>
<tr>
<td>time, android, com</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>1.8287</td>
</tr>
<tr>
<td>youtubei, googleapis, com</td>
<td>0</td>
<td>1</td>
<td>0.0</td>
<td>1.8287</td>
</tr>
<tr>
<td>i, ytimg, com</td>
<td>4</td>
<td>4</td>
<td>1.0</td>
<td>2.8248</td>
</tr>
</tbody>
</table>

Table 5.1: 50 Iteration Example Output - SB: success blocks, TB: Total Blocks, SR: Success Rate, UB: Upper Bound

- **Pre-loaded application components:** Applications may have components pre-loaded, accounting for this is important when observing performance hits. Facebook for example may pre-load videos sitting near the top of a timeline, switching to a separate timeline may be sufficient to demonstrate true network performance.

- **Subdomain Identification:** Many trackers identifiable by a specified parent domain may be responsible for subdomain services. As a result their network implications may be reserved to this focused domain.

**Accessibility**

Requests are made for tracking domain servers regardless of the client’s utility of the service. For example, multiple requests were made within functional testing from api.revenue-Cat.com despite the domain having no recording login of the client user nor associated application installations on the device. This tracking domain is self-sufficient, in that the server is not outsourced to tracking domains supplied from analytical companies. Observation of performance within a controlled setting, require either ignoring this domains or pre-installation of the service before experimentation. The former solution impacts model coverage and applicability. The latter may be difficult to implement within a scaled database.
Speed

Carrying out 50 iterations of the experiment took on average 30 minutes. This scales to about 1.6 minutes an iteration. Ultimately the runtime of the experiment is arbitrary once human input is considered. Vigour of inspection is subjective between clients and cannot be linearly tracked.

Connection Backlog

There exists a black log of queued connections adopting this solution. While data structures maintain the full list of connections seen, only the ones blocked by the algorithm are assumed active. It is important to control the feeding of new connections into the active bracket. An over willingness to add new connections would have the model performance substantially slower, in the suboptimal case, never converging. The control process adapted in the implementation only added connection where the iteration number is prime, empirically assessed to be a maintainable number of connections. This conditional allows the introduction of greater connections earlier in the model, with introductions getting less dense the more the algorithm matures.

Assuming the new connection protocol, 50 iterations of the functional experiment permit 15 active connections, 100 iterations would permit 25 active connections. Within the above functional experiment in the time period where 50 iterations we elapsed, 100 unique domain requests were logged, meaning the connection backlog assumed a 75% connection backlog rate for adding new connections.

5.7 Limitations

The imposed solution is limited to a dual-stage firewall service, it does not have real time capabilities. The decision to adopt this implementation was heavily owed to following constraints.

- **Architecture**: Rust is a hybrid language primarily assumed functional, though it has object-oriented characteristics. It is often duped an ‘expressive’ language, identifiably by its pure function ideals. Most code chunks have specified return values, leaving functions better behaving but restricted to tighter boundaries. State management is less accommodated as in the OOP language. As a result, alteration of a single function return value may have a compound effect on functions reliant on it. Thus, it may be harder to abstract and reproduce logic from a system with its architecture already in place.
• **Margin for Error:** For a real time add blocker, feedback is requested over a progressively larger field of connections. Work done within this feedback stage grows at the speed active connection are introduced, meaning potential for error inflates in tandem. Beyond one-hundred iterations the work done measuring the performance at each iteration would be impractical to measure by a human observer.
Chapter 6

Conclusions & Future Work

6.1 Future Work

Performance evaluation of the implemented model is left for future work. Credible evaluation by human input requires deep understanding of the observations and considerations highlighted above. This experimentation has identical bounds to that outlined at the beginning of functional testing however execution of it should be controlled, and potentially over a more significant number of model iterations. The evaluated model could then be compared to that of existing Blokada output under well-defined blocklists enabled.

Additionally, the final form of this technology is yet to be realised. The performance under time and accuracy constraints induced by the human entity. Performance evaluation software could in future work be partnered with this service to unlock the full potential of its service. Spike detection software could measure the strain put under an application in response to some stimulus. This could efficiently and reliably return reward, significantly boosting the firewall’s efficiency and accuracy. As reflected in closely related projects towards the beginning of this thesis a model is ultimately only as capable as its ground truth.

6.2 Conclusion

This thesis has been an investigative effort into the creation of tracker blocking firewall which worked using the injection of a MAB solution. Within the simulation study models were assessed based on both merit and functionality. Depending on the policy employed, singular or grouping, the use case of the blocking service is be determined. (Real time or otherwise). When using a classical policy, considerate algorithms like Upper Confidence Bounds perform impressively what AUC score never dropping below 96%. When using a
group policy the ability to block in real time is introduced, though it comes with the cost of more algorithm throughput and moreover, convergence time.

Within our simulation a reduced and increased connection set is specified, with models primarily being assessed under these bounds. By holding certain metrics fixed at the outset we optimise solutions fairly under the same environment conditions.

Grouping policies didn’t necessarily induce more regret, despite the greater combinatorial sample space given to a restricted binary reward feedback. Under these bounds as well as the functionality offered, Group Thompson maintains its streak of applicability, despite its unorthodox implementation. The smoothness of its proportion selection indicates its learned indecisiveness and makes it an easy model to assume some termination criteria for. Group Thompson did benefit the most from optimisations, indicative that it may generalise more poorly.

An implementation is detailed in the penultimate chapter for a classical UCB model within the android blocking firewall Blokada. We prove the functionality of the injected model through preliminary functionality tests but reserve more thorough examination of its performance for a future study. Observations are disclosed of the testing carried out to aid future experimentation but should also mustn’t be ignored on when considering the technology as a whole. True eradication of these concerns may only be combated with automation in the place of the human entity.
Bibliography


Appendix

A.1 Regret

(Unpreferable) Optimisations: (UCB)c:2,(GTS)at:0.8,(GW)l:0.6

Figure 1: Regret - Incorrect optimisation set

A.2 Regret

(Unpreferable) Optimisations: (UCB)c:2,(GTS)at:0.92,(GW)l:0.52

Figure 2: Regret - Incorrect optimisation set
Figure 3: Regret - Incorrect optimisation set

Figure 4: Regret - Incorrect optimisation set

B.1 ROC

(Unpreferable) Optimisations: (UCB)c:2,(GTS)at:0.8,(GW)l:0.6

Figure 5: ROC - Incorrect optimisation set 1
Figure 6: ROC - Incorrect optimisation set 2

B.2 ROC

(Unpreferable) Optimisations: (UCB)c:2,(GTS)at:0.92,(GW)l:0.52
Figure 7: ROC - Incorrect optimisation set 3 Figure 8: ROC - Incorrect optimisation set 4

D Network Issues

Figure 9: Youtube Music Network Issues

Figure 10: Youtube Network Issues

Figure 11: Facebook Network Issues