Identifying IOT-like devices and using Collaborative XAI to understand their cyber security behaviour

Steve Moyle
Cyber Security Centre, Oxford University, Oxford, Untied Kingdom

Andrew Martin
Cyber Security Centre, Oxford University, Oxford, Untied Kingdom

Nicholas Allot
NQuiring Minds, Southampton, United Kingdom

October 10, 2023

Abstract

Cyber attacking is easier than cyber defending – attackers only need to find one breach, while the defenders must successfully repel all attacks. This work demonstrates how cyber defenders can increase their capabilities by joining forces with eXplainable-AI (XAI) utilising interactive human-machine collaboration. Effective collaboration requires that both parties can communicate and teach the other party novel concepts. This requires a common, shared, and comprehensible language for both the Human Cyber Defender and the XAI. Here we use the language and power of logic, which was also Alan Turing's preferred choice in his specification for constructing a machine intelligence. Unlike Deep Neural Network approaches, Machine Learning based on logic allows 1) pre-existing Human knowledge to be easily codified for the Machine learner to draw on, and 2) that any new patterns discovered by the machine learner can be communicated back to the Human. Inductive Logic Programming (ILP) is an XAI paradigm that provides powerful machine learning, and that has a track-record of producing patterns that have been published as scientific discoveries in numerous applications. We utilise and extend ILP to be an interactive Machine learner, so that it can collaborate with expert cyber defenders to analyse network security data, and defenders' existing knowledge, to produce logical specifications of network attached device behaviour. Two commonly deployed device security approaches are: A) End-point or host-based – where measurements directly from the device itself are utilised in making security decisions; and B) Network Monitoring (NetMon) – where eavesdropping the communications that a network device makes are used to make security decisions. Both techniques have advantages and disadvantages. We take the pragmatic approach of using network monitoring as it is least disruptive to an existing network of operational devices and non endpoint-supported IoT devices. The primary data source is network flow meta-data captured on a live business network with up to approximately 180 devices attached, which has been gathered over a period of almost six months. A Turing Machine permits a countably infinite range of behaviour, and this allows any particular device to be observed performing potentially a very large number of communications. From a cyber defender's perspective, one is interested as to which of the devices have more volatile behaviours (and are hence more difficult to defend), and which devices are less volatile (and potentially easier to defend). This work proposes

a volatility metric based on the (simple) Good-Turing Frequency Estimator (SGT), and applies it to the data collected from the live business network. The average SGT metric shows variations from lowest to highest of 28 orders of magnitude between devices in the observed period. It also identifies devices that are consistently low volatility. It is conjectured that devices with low SGT volatility are easier to defend. The rapidly growing population of deployed IoT devices are an increasing challenge for cyber defenders. The security posture of IoT devices tends to be poor, often due to limited compute power of the devices, combined with the immaturity of the IoT security engineering. There is a real need to identify, segregate, and defend IoT devices, so that they are not an easy target for network compromise. Fortunately, IoT devices typically perform simple operations repeatedly, thus we hypothesise that IoT devices have a lower volatility than many other devices (e.g. a human's workstation). Empirically, this is supported in the analysis of the network flow meta-data from the operational business network. First we generalise the network flow meta-data from a single day of an IoT-like device (i.e with consistently low SGT volatility) using the SEQUITUR algorithm to produce a form of grammar. The SEQUITUR grammars are displayed using a novel visualisation that allows a Human defender to identify particular sequences of behaviour that recur for the device. Next, a specification of the recurring behaviour of devices with low SGT-volatility is logically reverse-engineered using the XAI Human-Machine collaboration system Acuity (based on the popular ILP system Aleph). Acuity combines i) network flow data, ii) induced sequitur device network flow grammar, and iii) Human cyber defender wisdom – all encoded in the logic programming language Prolog. This reverse engineering is a two-way collaborative process whereby the human cyber defender guides the machine learner to utilise the data. The machine learner proposes the best hypothesis (specification of device behaviour) that it finds, while the Human cyber defender updates the learner with information that the cyber defender knows, or suspects – much of it contained within their expertise and experience. Once an hypothesis is agreed on, it can be tested against more data from the same device (e.g. subsequent days), and then tested on devices of a similar nature. Ultimately, with a satisfactory specification of a particular class of device, it can be deployed as a defensive white-list within a network router's firewall. Each device that is secured by such a system is one less device for the cyber defender to worry (so much) about. Cyber defenders are a global scarce resource. Each device that we adequately secure makes for less work for the cyber defenders. This work defines a novel metric for

the volatility of network communications between devices, identifies devices that are low volatility, applies a hierarchical grammar building technique to identify sequences of network behaviour for individual devices, and uses XAI, in a logical framework, to reverse-engineer specifications of behaviour that can be deployed as part of a defensive system. This work empirically demonstrates that by amplifying the skills of cyber defenders with an explainable AI Human-Machine learning system, it can improve the understanding and the security of the behaviour of network IoT devices.

Contents

Chapter 1

Introduction

1.1 Cyber Security and its Challenges

Defending assets from cyber attack remains challenging. There are many asymmetries that disadvantage the defender, including – rare symptoms of compromise are buried within burgeoning log files of (most likely) normal behaviour; – specification of the assets are rarely complete and accurate; – public knowledge of exploits are always out of date, with the continual emergence of new zero-day attacks; – systems may have a vulnerable component, but the deployment context and configuration may make it unexploitable. This leaves the defender having to reason in the complex world where the information they have access to cannot be relied upon. Ultimately, skilled cyber defenders admit that it is almost impossible for them to give any strong security guarantees [Mee15].

There is a severe lack of human skilled cyber defenders. An international body that supports cyber defenders estimates that there are more than three million fewer defenders than necessary [(IS22]: "To adequately protect cross-industrial enterprises from increasingly complex modern threats, organizations are trying to fill the worldwide gap of 3.4 million cybersecurity workers." Organisations with cybersecurity resources are those who can afford it or must have it – industries like banking, finance, government and military. The shortfall identifies only skills of industrial enterprises – it is greater still in the small business and the domestic settings. It is in these settings where many devices are simply installed from their packaging and left operating in an undefended environment, with little or no attention.

Real computer networks can be arbitrarily complex. For example, detailed analysis of a small organisation's network, where data was collected for half a year, showed that 181 devices were connected within its networks (see section 2.1 for further details). Some of the devices are for normal operational duties (for example laptops, network routers, door swipe access controllers), others are defensive in nature (for example anti virus software and vulnerability scanners). All devices pose a risk of expanding the organisation's *attack surface*.

In larger organisations the asset registers and design documentation can soon fall behind those of the true implementations. Systems architects often cannot agree what is and what is not actually running in their environments. Cyber defenders are keen to incorporate both *controls* (for example network firewalls) and *monitoring* as part of their defensive toolkit. Monitoring can be *passive* like the continuously running network security monitoring system `Zeek` [Zee21], or it can be *active*, like vulnerability scanners (e.g. `OpenVAS` [Ope23]) where probes for *known* vulnerabilities are commissioned.

Designing a secure device is fundamentally challenging. Internet-of-Things (IoT) devices are often resource constrained so as to meet demanding low-power and low cost specifications. Security of the device is a lower priority. Certifying the level of security for a particular device design is also challenging and costly[1] (e.g. (the now super-ceded) Orange Book[Boo02] and Common Criteria[Org00]). A certified secure device at installation time allows us to *presume* that it is *good* at that time, but some time later a *vulnerability* in that version of the device is discovered. This does not automatically mean the device is *bad* – the vulnerability needs to be *exploitable*, and that a path-way to exploitation is present.

Installers and operators of devices rarely know precisely how the device should behave. There is a growing appreciation for published device information from the device manufacturers themselves helping to improve security (e.g. [GM20], [MFM+22]). The *Distributed Device Descriptors (D3)* standard[ea21a] allows both manufacturers and interested parties to specify what resources a device class requires, and the behaviour that it produces. This provides the possibility to perform run-time comparisons of the behaviour of an instance of the device class and its actual, monitored behaviour. Deviations from specification can be caused by the device going *bad*, or simply that the specification has errors of

[1]The costs of security certification for systems is prohibitively high, and worse, only assure the current version under evaluation. Awarded certifications do not apply to a future bug fixes or feature enhancements.

omission or commission.

The number of computable programs is countably infinite[Tur37]. If we consider a crude partitioning of all programs into *good programs* and *bad programs* it is obvious that each of these partitions are themselves countably infinite. It follows that it is unrealistic, in general, to be able to guarantee that a persistent attacker will not be able to entice the computer-based systems to execute bad programs. It is easier to intercept communications between devices than to access the runtime environment of the devices. Considering the messages between computing devices, these too, are countably infinite. This makes it difficult, in the general case, to use external eavesdropping of devices' communications for predicting whether the messages themselves are *sinister* or *benign*.

Standards and processes exist that allow information systems (a.k.a. computing devices) to have their security properties rigorously asserted (e.g. [Boo02] and [Org00]). Although these are typically only used for high-end systems (i.e. rarely for consumer devices and IoT) they do provide empirical evidence of the existence of some sliding scale of *easy-to-secure* and *hard-to-secure* device designs. As a cyber defender, are there devices which operate in a manner that make defending them *easier* than other devices? Can we determine by observing devices externally which of them are (likely to be) easier to defend? By studying their eavesdropped connections and messages to other devices, can easier to defend devices' predictable behaviours be defended by blocking novel behaviours (or at least alarming or alerting with the novel events occur)?

1.2 Machine Intelligence and Explainable AI

ChatGPT and other systems powered by large language models have captured the public imagination in recent times. However, not everyone considers that these are general purpose artificial intelligences ([Wol23]). In his 1950 paper [Tur50], Turing goes beyond his pragmatic imitation test of machine intelligence[2] and considers the question, "Can machines think?". Turing quickly dismisses nine common philosophical objections to his question about thinking machines, and then focuses on three strategies which might lead to the creation of a thinking machine. Broadly (from [Mug14] these are 1) A.I. by programming, 2) A.I. by ab initio machine learning, and 3) A.I. using logic, probabilities, learning, and background knowledge.

[2] *Machine Intelligence* was the term Turing used. It was not until 1958 that John McCarthy introduced the term *Artificial Intelligence*[McCarthy1958].

Ab initio machine learning is the current state of the art of A.I.. It takes recorded examples of some phenomena that we wish to forecast and use algorithms for the production of models or code (see [Fla12]). *Ab initio* refers to the way the models are updated when the new data arrives – typically by discarding the model and starting the learning again, but with the additional data to utilize. Many of the current techniques give excellent forecasting accuracy when interpolating with novel inputs. This gives an illusion of intelligence. In reality these are systems typically trained to perform one, narrow, task.

Michie[3] categorises machine learning styles as *weak*, *strong*, and *ultra strong*[Mic88]. His weak criterion of Machine Learning is a "system [that] uses sample data (training set) to generate an up-dated basis for improved performance on subsequent data". Indeed deep neural network models satisfy this criteria. He goes on to define the Ultra-strong criterion[4] of Machine Learning as a "system [that] satisfies [the] weak criterion and also can communicate its internal updates in explicit *and operationally effective* symbolic form". Beyond the standard accuracy of its forecasts, this requires that the machine learning generated model can be communicated to another agent[5]; and that the agent can be *coached* to improve its own performance on the same task that the model is used for. In other words "Don't simply give me the answer, explain it to me and teach me how to use it.".

Under this lens of Michie's *Ultra-strong Machine Learning*, Deep Learning and its *black-box* style models fall short[6]. Inspecting the matrices of weights that represent their models are not comprehensible, nor are they in a symbolic form. Some researchers have found ways to identify what it is that the Deep Learning has locked on to when making its forecasts. For example in object recognition from digital images, is it the polar bear or the snowy background. People argue that this is a step to explaining the A.I.[7] – but it clearly fails Michie's stronger definitions.

Returning to Turing's view [Tur50] on the matter of the construction of an A.I.. His third, and preferred alternative is: *A.I. using logic, probabilities, learning and background knowledge*. Turing states that "...one might have a

[3]Donald Michie was a wartime colleague of Alan Turing. They discussed Machine Intelligence whilst playing chess socially.

[4]The Ultra-strong criterion of machine learning is an incremental upgrade on the the Strong criterion adding the conjunction "and operationally effective".

[5]The agent we are most interested in is a *human*.

[6]Deep Neural Network models do manage to pass the less stringent criterion of *Weak* Machine Learning.

[7]It seems that A.I. simply means machine learning in this context.

complete system of logical inference "built in." ... the store would be largely occupied with definitions and propositions. The propositions would have various kinds of status, e.g., well-established facts, conjectures, mathematically proved theorems, statements given by an authority, expressions having the logical form of proposition but not a belief-value." In a sense, Turing is suggesting that underpinning the learning should be a compendium of existing knowledge – in a logical form that can be used to reason with. Such an approach provides benefits over the ab initio machine learning. 1). Existing knowledge can be utilized as "Background Knowledge". This has benefits for the learner – in that it is not forced to relearn every thing every time new examples appear. 2). New Learned knowledge can be independently verified. In Turing's world, it is possible to ask the learner after the fact, "What it is that you have learned?". The learner can then answer, for example, by emitting its learned model in logic. Even if the model is not perfect, it is in a form that can be tested, debugged, and then adopted into the background knowledge for future learners to use.

Going beyond systems that simply offer good predictions, Explainable Artificial Intelligence (or XAI) as described in [GVWT21] can be seen as patching a gap in contemporary black-box machine learning techniques, so as to overcome limitations in communicating the machine learning results to humans. However, Turing, in 1950, had already provided a pathway to *Explainable Artificial Intelligence*. Michie [Mic88], too, was insistent on machine learning results being communicated in a comprehensible manner. Comprehensible communication remains challenging, even between humans, and continues to be researched (e.g. [MSZ+18]).

1.3 Machine Learning challenges for Cyber Security

The application of machine learning techniques faces particular challenges in the domain of cyber security. Cyber defence in itself is plagued by many asymmetries that advantage the attacker, the greatest being that the defender must prevent all attacks (many of which are previously unknown), whilst an attacker need only one way of breaking in.

Machine learning – by definition – requires example data[8] to learn from. There is no shortage of data in the cyber world – for example vast quantities of

[8] Examples used to learn from are known as the *training [data] set.*

log files are collected relating to systems and their operation. However, most machine learning techniques are *supervised* (for a general description of machine learning practices see [Fla12]) requiring that each data item is *labelled* with the outcome observed. For this discussion we can use the simple labels of **sinister** and **benign**. Unfortunately, the vast log data is *unlabelled* making supervised machine learning techniques inappropriate. When attacks are analysed and the log records associated with them labelled (as sinister) there remains a vast imbalance in the training set. Machine learning algorithms give the strongest results when trained on balanced training sets: in this case equal amounts of sinister and benign example records. When the proportion of sinister to benign records is low (e.g. say, one in a hundred – which is likely to be quite an over-estimate in real-world operational environments), the models produced will almost certainly predict a new log record as benign. This leads to false negative errors where the sinister records are identified as benign – making the detection a useless defence. Pushing the learning to focus on the sinister records (e.g. by using *boosting* [Fla12]) will only lead to a model that has a high false positive alarm rate – misidentifying benign records as sinister. Cyber security systems with high false positive alarm rates are considered untrustworthy, and often lead to them being ignored.

In one sense, there is too much data (unlabelled logs), yet in another sense we don't have enough (sinister labelled logs). To a naive first approximation everything is considered safe. To make progress we must keep expert human cyber defenders in the machine learning loop, and provide them with tools to make sense of what they see. If humans are key, then so too is XAI, as the cyber defenders will require machine learning frameworks that output models that they find comprehensible.

1.4 Human-Machine collaboration

The practice of extracting patterns from collected data has a long history[9]. Humans are somewhat distinguished from most other animals by making tools. The usefulness and ease of use of which are important factors. If one seeks a deeper understanding of the tangible world then using the tool of *scientific method* is a good practice. For example, the inductivist incorporation of empirical observations as a basis for constructing general laws (this can be traced back at

[9]Kepler is credited with fitting Tycho's planetary observational records to an elliptical orbit.

least to the 17th century [BUG96], [Jev12], [Pei96]).

For Humans to be able to collaborate with the data science tools requires that both agents – the human and the machine – understand a common language. Quite often the scientific expert is not a data analytics expert, so a team-based approach is required. Significant successes have been achieved this way where the actual scientific "discovery" is initially proposed by a machine (learning algorithm). Mutagenesis [KMSS96] and drug discovery [FMPS98] are examples where published results in a discipline were proposed by a machine learning system (see also [Gil96]). In these cases it is key that the output of the systems are comprehensible to the experts, requiring translations into the experts' preferred formats.

Considering the human cyber defenders. They apply their skill and reasoning in much the same way that a detective solves a murder case. They constantly form hypotheses, collect more evidence, and focus their investigations. They will use *abductive* reasoning to form hypotheses and seek yet-to-be discovered evidence, whilst forming constraints based on the evidence that will rule out incompatible hypotheses and lines of enquiry. They may be able to conclude, based on reasonable probability, the motive, opportunity, and weapon used by a suspect. Once the case is solved, then it is possible to follow a *deductive* reasoning process that explains the case and its artefacts. Furthermore, with the solution of similar styles of cases it is possible to apply *inductive* reasoning to learn general patterns.(e.g. [RM21]). The machine side of the relationship will be required to operate in similar ways.

We require that the machine can make use of the existing domain knowledge with which the human expert is familiar. A bi-directional shared language is necessary. One that is directly executable is Logic programming, which has a long history of knowledge representation and reasoning. Despite its longevity and many successes standard `Prolog`, with its sound semantics and elegant computation model (e.g [Llo84]), does not directly support the forms of reasoning that humans often rely on. The more expressive Answer Set Programming (e.g. [Lif22]), allows a broader range of reasoning, and is not restricted to Logic Programs with single models. Subsets of Logic Programming using constrained natural language are enabling humans to have two-way interactions with systems([Kow11], [Sch20]).

Science proceeds where scientists can replicate another's work. A more robust advance is where scientists can refute and argue about another's findings. Indeed, philosophers of scientific method suggest that refutation is key (e.g. [Pop35]).

A strong human-machine system will be one where each party can argue both for and against the communicated understanding of the other. A framework for this scenario is reported in [SBC22].

A candidate Human-Machine environment would need to support explicit symbolic knowledge, in a comprehensible and explainable form to the human, that is directly executable by the machine, where the state of knowledge can be refuted by each party, through multiple logical reasoning forms (deduction, abduction, induction). This seems to be similar to Turing's original 1950s preferred specification [Tur50] of what it will take to build a "thinking machine" – what we would now call an XAI. This is document takes reports on efforts to develop techniques for use in an XAI environment specifically to support Cyber Defenders.

1.5 Objectives

Motivated by how difficult it is for professional cyber defenders to keep their environments safe [Mee15] one of the goals is to identify ways to improve the security of Internet-of-Things (IoT) devices in a domestic setting. We wish to determine if it is plausible to use XAI-like tools to reverse-engineer behavioural models (c.f. [BS95]) (i.e. rules) that a specialized IoT gateway device could utilise to only permit necessary network interactions to IoT devices and their controllers.

The main aim of this work is to test the following hypotheses:

- Hypothesis 1 – Identifying **easier-to-secure periods** of operation using eavesdropped communications That it is possible to use eavesdropped communications information to identify periods of time in a network of devices that are *easier-to-secure* than other periods of time by measuring the volatility of **all** the devices' communications over time.

- Hypothesis 2 – Identifying **easier-to-secure devices** using eavesdropped communications That it is possible to use eavesdropped communications information to identify devices that are *easier-to-secure* than other devices by measuring the volatility of devices' communications.

- Hypothesis 3 – Reverse-engineering **behavioural models** using eavesdropped communications from easier-to-secure devices. That it is possible

to use eavesdropped communications information from identified *easier-to-secure* devices to build behavioural models of allowed (and disallowed) communication using XAI approaches.

These hypotheses chain together. Without evidence that there are time periods or individual devices that show reduced volatility, then it will be difficult to identify candidates that are easier to defend. Hypothesis 3 requires a positive result from Hypothesis 2. Should Hypothesis 3 be valid then a subsequent line of research would be to determine if and how the behavioural models of communication can be deployed as a defensive tool for the device(s). This is dealt with in chapters 4 and 5. Hypotheses 1 and 2 are tested in chapter 3.

1.6 Outline

This is an empirical work utilizing observational data collected from a real-world organisation. The computer network and the data are outlined in chapter 2. Then in chapter 3 we consider how to identify *easier-to-secure* devices by studying the eavesdropped connections and messages to other devices. Devices that have low network volatility allow predictable behaviours to be defended by blocking novel behaviours (or at least alarming or alerting with the novel events occur). We thus identify and target one particular IoT-like device to study in detail. A form of behavioural cloning is performed using machine learning of sequences of messages used by the target device in chapter 4. A novel tool to visualise the extracted patterns allows a human defender to select interesting recurring sequences generated by the target device for further analysis. Chapter 5 details an interactive logical induction approach to analysing fine grained sequence patterns in the presence of existing background knowledge. The results produced confirm to the expectations of explainable A.I. – they can be understood by the cyber defender. The discussion (chapter 6) includes suggestions for future research directions. Chapterr̃efsec:conclusion concludes the report.

Chapter 2

Observational Case-Study Context

2.1 Network traffic monitoring

Systems can be observed from many perspectives and many different granularities. For example we can observe the internal operation of entire computer systems by executing them from within a virtual machine environment – a form of *white box* analysis. An alternative perspective is to observe the inter-system communications passed into and out-of a computer system – a form of passive *black box* analysis. In this work, out of operational pragmatism, we choose the latter approach, and study the observation logs recorded from eavesdropping network connections and conversations between individual devices.

To collect data for the case study, we implemented the system architecture shown in Figure 2.1. Network traffic metadata is passively collected by a bespoke edge device consisting of a network TAP (i.e. a device that clones data flowing across a network) and system-on-chip processor running an established Network Security Monitoring system called Zeek [Zee21] (formerly known as Bro). This data is periodically pushed to a secure cloud storage facility. This IoT-like device – called the *brAIn-box* – was developed in-house integrating publicly available components, in a secure and robust way, and utilizing efficient server-less cloud processing design patterns.

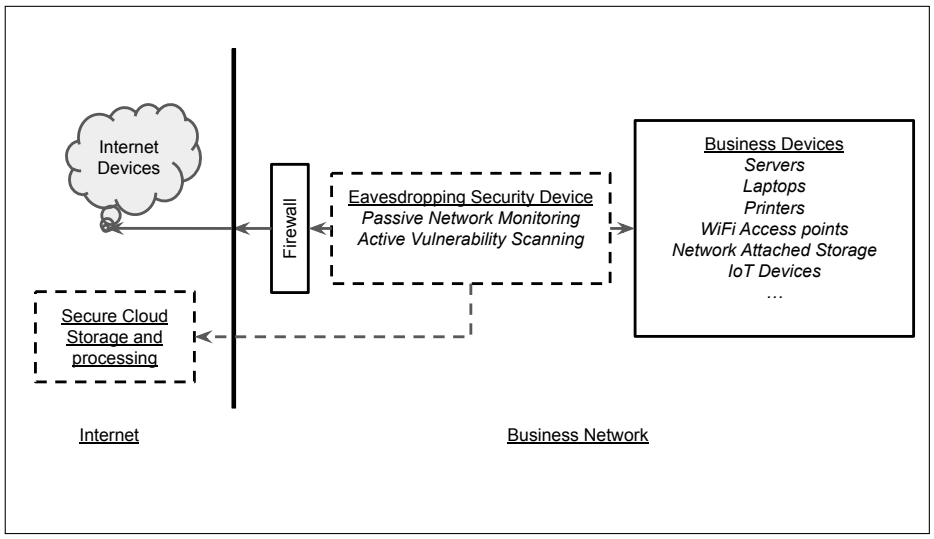

Figure 2.1: System Context diagram: A network monitoring edge device passively records metadata about observed network connections from and to an office with networked devices. Logs are securely stored and processed in the cloud.

2.2 Materials and Methods

This is an empirical line of enquiry based on previously captured eavesdropped communications data. The case study uses collected communications data from a small business (as loosely depicted in figure 2.1) from January 1st to June 20th 2021 (inclusive) – a period of 172 days. During the observed period there were between 28 and 185 active networked devices per day, resulting in between approximately 63K and 562K logged connection events per day.

The data has been collected by the Zeek[Zee21] Intrusion Detection and Network Monitoring System which logs metadata extracted from the observed network communications. Only the communications that passes through the network monitoring device are logged for analysis[1]. The map of the metadata schema is shown in Figure 2.2. The master data table is `conn`, which keeps a summary record for all connection events observed.

Much of the analysis focuses on the `conn`ection log. An example record and details of the meta data fields available in the `conn` log are shown in Table 1 (in the Appendix). The example shows an `http` (service field) connection between

[1]An eavesdropper can only record what it observes. Encrypted and wireless communications were not explicitly monitored by our device.

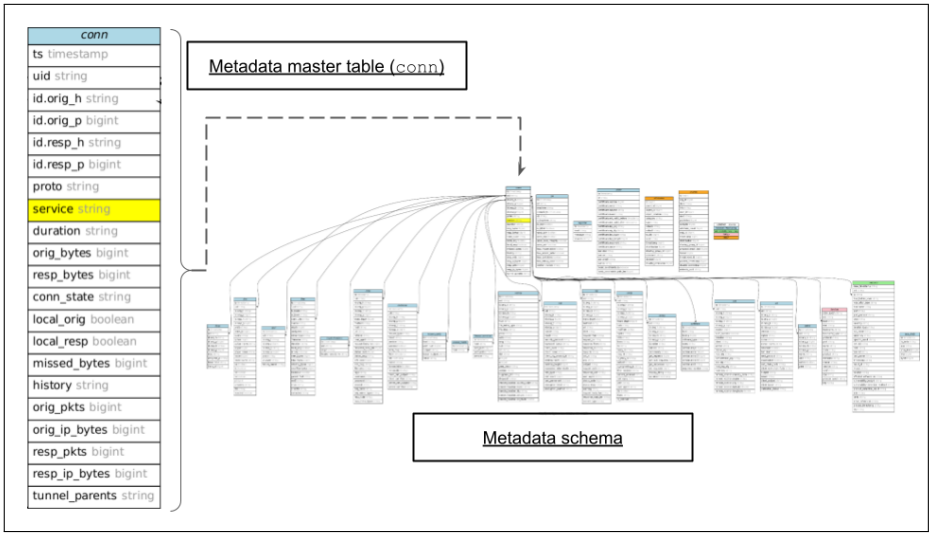

Figure 2.2: Eavesdropped network communications metadata data schema. The left hand side depicts the table name/columns for the `conn` master metadata table recorded by Zeek[Zee21]. The right-hand side shows the complete metadata schema map (for context only). Note the many subordinate metadata tables, which provide metadata details for each `service` recorded in the `conn` table (e.g. `http`, `dns`, `dhcp`, ...)

a client at `10.0.100.55:51475` (`id.orig_h:id.orig_p` fields) and a server at `192.29.32.24:80` via the `tcp` protocol, initiated at `2020-12-31 23:59:22` (`ts` field). Zeek maintains a unique `uid` field, `CSuVAz2v3MtaiPkaja`, which provides an index into detailed event records specifically from the `http` table (not shown).

Many of the Zeek tables have a similar initial set of columns with event timestamp – `ts`, unique identifier – `uid`, originating IP address and port – `id.orig_h`, `id.orig_p`, responding IP address and port `id.resp_h`, `id.resp_p` recurring in many tables.

2.3 From raw Network to new Cyber Defender knowledge

Raw network logs are only one source of information that cyber defenders draw on – particularly when responding to a reported cyber incident. We want to process, transform, and analyse the logs so that they can be used by the Human-XAI to co-create behavioural descriptions of the device being studied. This process is

sketched in figure 2.3). Note that the cyber defender plays a key role in selecting the device of interest to study (*H1*), providing the background knowledge about the environment and cyber security (*H2*), and interacting directly with the XAI system to produce a new understanding that is mutually acceptable.

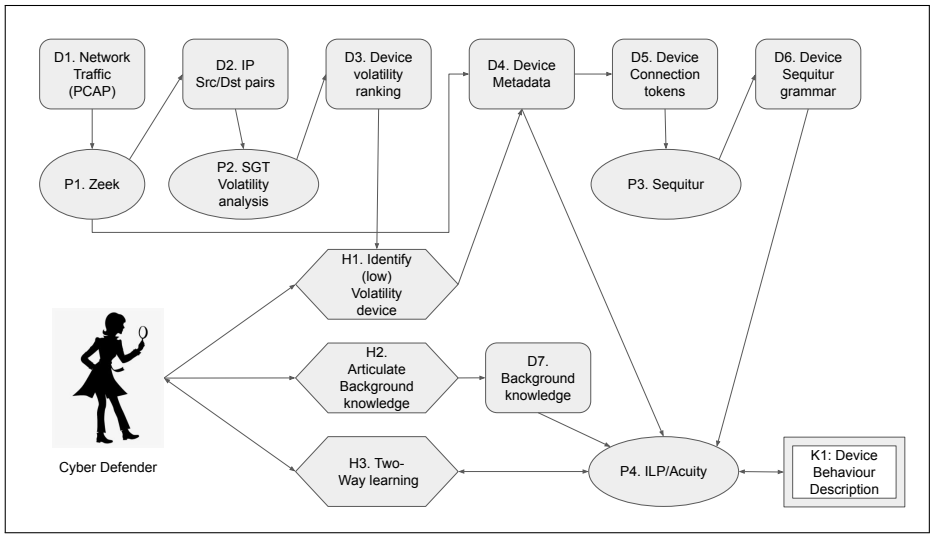

Figure 2.3: Process Flow diagram: Data Elements (**D**) start with eavesdropped Network Traffic (D1), and are processed by processing elements (**P**) with the human Cyber Defender's interactions (**H**), using eXplainable AI, to produce new knowledge (**K**) culminating in the understanding of device behaviour that can be used to improve security further. The data transformations *D1* to *D6* are along the top line of the figure; below are the processes *P1* to *P4* that perform the transformations. The human cyber defender plays their crucial role in *H1*, *H2*, and *H3*; with the ultimate outcome produced as as new knowledge in *K1*.

Chapter 3

SGT as Device Volatility

We first tackle the two initial hypotheses regarding **easier-to-secure** *periods* and *devices* outlined in section 1.5. How might we identify these from the raw network flow metadata captured by our eavesdropping IoT-like device? In this section we utilize an develop an analytic technique developed by Turing and a colleague whilst working as codebreakers.

During the second World War, cryptanalysts Jack Good and Alan Turing developed frequency-based attacks on enemy cipher systems, where their work included determining estimates of character bi-gram probabilities. They developed a frequency smoothing approach, based on the frequency of frequencies of observed bi-grams[1] in samples of plain-text. The models built by the technique provide an estimate of the *unseen probability mass* from a sample. In this work we use a simplified form of the algorithm from Gale and Sampson [GS95] to fit Good-Turing frequency models on samples of real-world network traffic data.

3.1 Good-Turing Frequency Estimation

The original motivation for the Good-Turing Frequency estimation relates to what can be sensibly inferred *beyond* a sample of nominal objects relating to objects not present in the sample. We will use Sampson's [Sam17] example to illustrate.

> Suppose you want to estimate how common various species of birds are in your garden. You log the first thousand birds you see; perhaps you

[1] Legend has it that Good [Goo53] cunningly published the technique with a motivating example relating to ornithology so as to avoid censorship.

> see 212 sparrows, 109 robins, 58 blackbirds, and lesser numbers of other species, down to one each of a list of uncommon birds. Perhaps you see 30 species all told. How do you use these numbers to estimate the probability that the next bird seen will be, say, a blackbird? Most people would surely say that the best guess is 58 ÷ 1000, i.e. 0.058. Well, that's wrong.
>
> To see that it's wrong, consider a species which didn't occur at all in the thousand-bird sample, but which does occasionally visit your garden: say, nightingales. If the probability of blackbirds is estimated as 58 ÷ 1000, then the probability of nightingales would be estimated as 0 ÷ 1000, i.e. non-existent. Obviously this is an underestimate for nightingales; and correspondingly 58 ÷ 1000 is an overestimate for blackbirds.

The Good-Turing Frequency Estimation technique is based on a power-law distribution of object frequencies. It takes as input the counts of objects in the sample, and then uses the counts of the counts (spectra) to fit a log-log regression equation. The full details are described in [GS95]. For each model, the method returns the *slope* and the *intercept* of the regression equation, as well as a goodness-of-fit measure (r^2).

Once fitted from the sample data, the model can be used to provide smoothed estimates of probabilities of objects occurring in the population, including an estimate of the *unseen probability mass* **P0**. This method works even when the underlying population cardinally is very large (or indeed infinite)[2].

Good and Turing were interested in the *estimation of population parameters*[3]. For this work, we are predominantly interested in the parameter **P0** – and using it as as an indicator of *volatility*.

3.2 SGT Experimental setup

This section describes the experimental setup and places the experimental aims into the broader context of improving the security of physical devices broadly categorised as the Internet-of-Things[4].

[2]This makes the technique useful in computational linguistics where the number of potential sentences in a natural language is for all practical purposes infinite.

[3]Hence the title of Good's paper[Goo53] *The population frequencies of species and the estimation of population parameters.*

[4]Definitions of what makes a device an Internet-of-Things vary. We will use the definition from Nick Allott (pers. comm.): "Generally it looks like a server
- likely to be always on - more likely to receive connections (which is why it needs to be always on) - than to initiate them - less diversity of connection types (e.g. Things it talks to/ways it talks to them) - more likely to "chat" on internal networks (connections on the intranet)"

1: $FirstDay \leftarrow$ `2021-01-01`
2: $LastDay \leftarrow$ `2021-06-21`
3: **for all** Day such that $FirstDay \leq Day \leq LastDay$ **do**
4: select every `id.orig_h` observed on that Day
5: Produce a count-of-counts frequency histogram F
6: Fit F to an SGT model and return $Slope$, $Intercept$, r^2, and $P0$
7: Return also the total number of unique devices D and the total number connections C
8: **end for**

Figure 3.1: Outline process for calculating Simple Good-Turing (SGT) analysis based on `id.orig_h` data recorded in the `conn` network flow meta-data table.

3.2.1 Analysing daily *conn* data with SGT volatility

The key objective of this section is to determine whether the Simple Good-Turing (SGT) models show some evidence of utility [5] in measuring network event volatility from daily `conn` data. All raw log data was observed by the *brAIn-box* eavesdropping system with unique serial number `5cbe4ac3` in the period `2021-01-01` to `2021-06-21` (inclusive). The process for analysing the data, particularly the SGT models, is outlined in Figure 3.1.

3.3 SGT Empirical Results and preliminary discussion

This section presents results from the analysis of the eavesdropped data, in the context of the hypotheses being considered. Preliminary observations are also discussed.

3.3.1 Hypothesis 1 – Observations

Hypothesis 1 is concerned with the SGT estimate of unobserved species of network communication events, $P0_{events}$. Does $P0_{events}$ vary over time, and if so, can we determine *low volatility* and *high volatility* periods of network traffic?

3.3.2 Support for Hypothesis 1

The daily data is plotted in Figure 1 (in the Appendix) – Simple Good-Turing frequency analysis of the Originating Hosts (Source IP addresses) observed by

[5] *Utility* is not formally defined. This may be considered for future work.

the network monitoring system with serial number `5cbe4ac3` during the period 2021-H1. The plot has four temporal panels, each described below.

- Panel 1 – purple bar-chart `P0 -- unobserved probability mass` – These are Simple Good-Turing model estimates of the unobserved probability mass, P0 (as a form of volatility metric) – high P0 being expected to be indicative of high volatility. For this plot P0 has its maximum on 2021-06-20 with a value of 331 microns. The minimum of 6.1 microns occurs on 2021-02-12.

- Panel 2 – cyan bar-chart `Daily count of observed networked devices` – This is the number of unique IP addresses seen on the monitored network. For this plot the maximum number of devices is 185 on 2021-06-16, and the minimum of 28 observed devices from time-to-time in the final 5 weeks.

- Panel 3 – orange bar-chart `Daily Count of monitored connections` – Each of he orange bars show the total number of connections in a one-day period. The maximum value is 561,742 connections at 2021-06-02. The minimum value is 63,710 on 2021-06-19. The seven-day-week pattern is evident in the periodic variation[6].

- Panel 4 – blue/read/green trend lines `Slope, Intercept,` R^2 `for SGT equation of best fit` – These show the variation in Simple Good-Turing model fit parameters. The model is a linear log-log model. The slope is the blue line, the intercept the red line, and the correlation coefficient (r^2) is green. We desire r^2 to be as close to 1.0 (approaching 1.0 from below) as possible, indicating a good model fit.

The following observations are noteworthy.

- Weekly cycle All four of the plots in Figure 1 display a weekly cycle (albeit the P0 value seems weakly correlated with the day of the week). The ten highest values and the ten lowest values for total daily connections (orange bar chart) are shown in Table 2. This confirms that connection rates are lowest on weekends [7].

[6]The evidence for the weekday is shown in Table 7 in the appendix.

[7]The observations were made from a company adhering to a standard western culture working week – workdays being `Monday, Tuesday, Wednesday, Friday` and non workdays being `Saturday, Sunday`.

- Connection rates (weakly) correlate to *low volatility* In all three cases where P0 is measured on a daily basis: **originating hosts** (Table 5), **responding hosts** (Table 6), and **originating-responding host pairs** (Table 7), there is a predominance in lower P0 values occurring on weekend days. See section .2.3 in the appendix for more details and discussion.

- Originating and Responding P0 are not tightly coupled

  Figure 3.2[8] combines three daily measures of P0: **P0 of Originating Hosts**, **P0 of Responding Hosts**, and **P0 of Originating-Responding Host Pairs**; as well as daily totals of each form.

  Studying Figure 3.2 we can see that the number of originating devices is about forty-fold lower than the responding devices[9]. It shows that the two sources of P0 are not strongly correlated.

[8] Take note of the different Y-axes scales.

[9] For the 243 day period from 2021-01-01 to 2021-06-21 the sum of the count of each days originating devices O was 16,372 and for the responding devices R was 621,389 giving a ratio of $R/O = 37.95$.

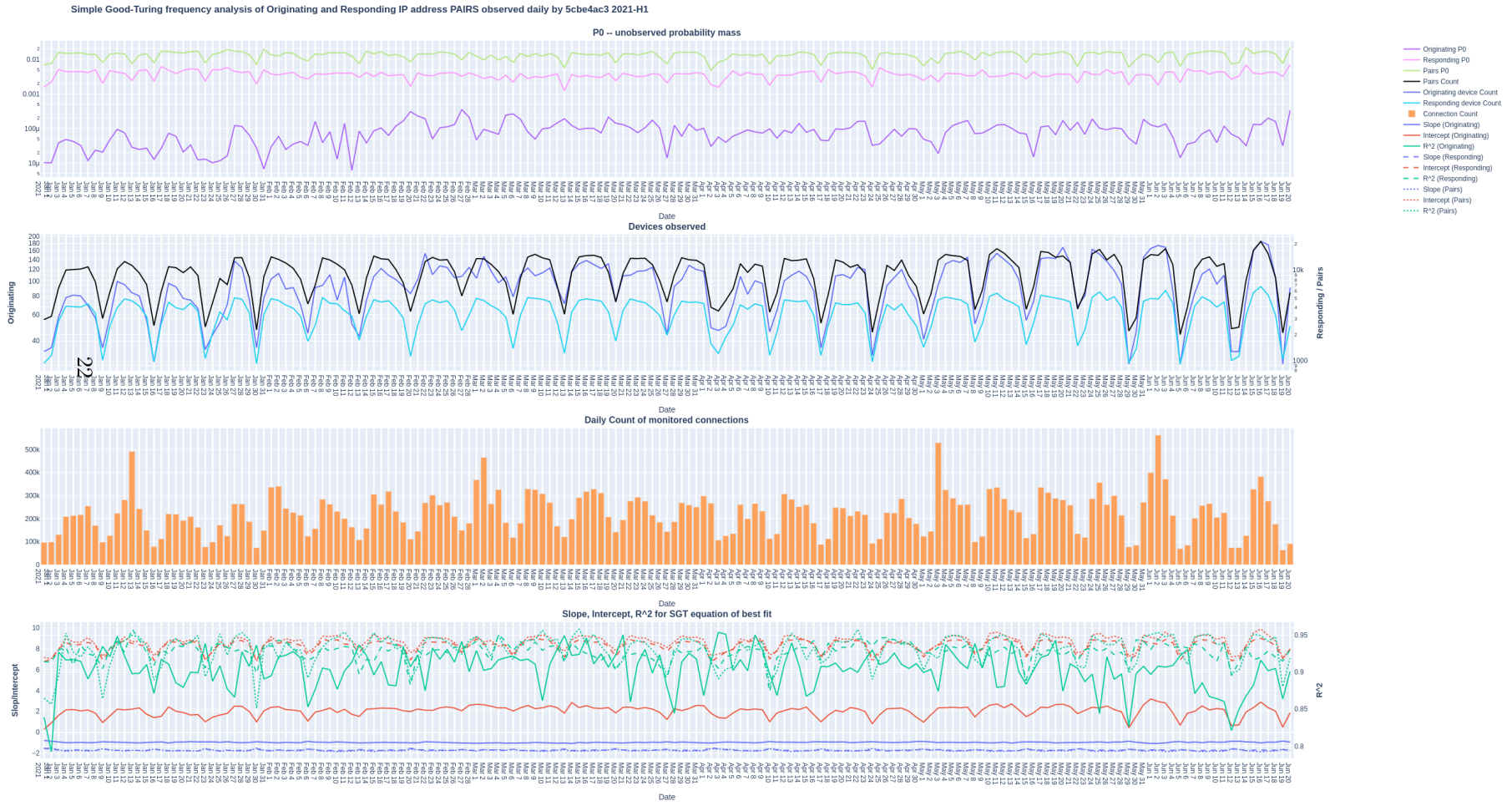

Figure 3.2: Simple Good-Turing frequency analysis of individual Originating IP addresses, individual Responding IP addresses, and combined Originating-Responding IP address pairs for 5cbe4ac3 2021-H1 – by day. Top panel – $P0$ SGT probability mass of unseen addresses (a proxy for volatility) – Originating IP addresses (purple); Responding IP addresses (pink); Originating-Responding IP address pairs (green). Panel 2 – Daily count of IP addresses – Originating IP addresses (dark blue); Responding IP addresses (light blue); Originating-Responding IP address pairs (black). Panel 3 – orange bars – number of monitored connections. Panel 4 – model fit parameters – intercept (red), slope (blue), r^2 (green) – Model for Originating IP addresses (solid lines); Model for Responding IP addresses (dashed lines); Originating-Responding IP address pairs (dotted lines).

3.3.3 Support for Hypothesis 2 – Identifying easier-to-secure devices and *stinkers*

Recall that Hypothesis 2 is about whether it is possible to use eavesdropped communications information to identify devices that are easier-to-secure[10] than others by measuring the volatility of devices' communications.

First we study the daily $P0_{conn}$[11] values for known internal devices. Internal devices as determined by their `id.orig_h` (i.e. IP address) are identified as they are on *non-routable* subnets. Two groups of `id.orig_h` values are shown in Figure 3.3 – from subnets `192.168.` and `10.0.0.`.

[10] *Stinkers* are those devices that have very high volatility and are therefor more likely to be difficult to defend.

[11] $P0_{conn}$ is the SGT estimate of the un-seen probability mass for items as measured in the *conn*ection logs recorded from a monitored network.

Figure 3.3: Simple Good-Turing frequency analysis of individual internal IP addresses observed daily by 5cbe4ac3 2021-H1. Top panel – $P0$ Originating IP addresses with internal IP address prefix `192.168.` . Bottom panel – $P0$ Originating IP addresses with internal IP address prefix `10.0.0.`. The X-axis is calendar dates, while the Y-axis is $P0$.

The following observations are from Figure 3.3.

- **New Network subnet** Towards the end of April 2021, a change to the network addressing occurred. An increase in devices with addresses on the subnet `192.168.` is observed.
- **Individual Device volatility** Even at the individual device granularity, $P0_{conn}$ shows daily variability from day to day, and device to device. This provides evidence that $P0_{conn}$ is tracking variability even at the individual device level.

Further aggregate statistics of each individual device $P0_{conn}$ for the entire period of 2021-H1 produces. There are 813 devices (as determined by having a unique source IP address) for which a non-trivial[12] $P0_{conn}$ value exists.

- The device's `id.origh_h` name
- The *sum*, *mean*, and *standard deviation* for each device across 2021-H1
- The number of days in the 2021-H1 period for which the device was observed on the network

This aggregate information can be ordered from lowest to highest average $P0_{conn}$ value as shown in Figure 3. The blue upper bar plot in the figure is the average $P0_{conn}$ for 2021-H1 – each bar represents a single `id.orig_h` (e.g. an IP address). The red central bar plot in the figure is the standard deviation $P0_{conn}$ for 2021-H1, and the green lower bar plot in the figure is the number of days the `id.orig_h`[13] was observed in the period 2021-H1.

From Figure 3.3, where the $P0_{conn}$ values are plotted on a logarithmic scale, we observe a sigmoid shaped plot, with some extreme values at each end and long plateau-like section in the middle. We can identify the five lowest and five highest devices by their period average $P0_{conn}$ (in table 3).

Figure 3.4 is a re-plotting of the data in figure 3 and focuses on the twelve lowest ranked $P0_{conn}$ devices, while figure 3.5 shows the daily average P0 values for three low-ranked volatility devices with IP addresses `10.0.0.137` (the blue trace), `10.0.0.145` (the red trace), and `10.0.0.173` (the green trace).

[12] Non-trivial is $P0_{conn} < 1.0$. A value of 1 is maximal uncertainty, and does not contribute further to the analysis.

[13] Recall that the number of days in the period 2021-01-01 – 2021-06-20 is 171.

Figure 3.4: Simple Good-Turing frequency analysis summary of individual internal IP addresses observed daily by 5cbe4ac3 2021-H1. Rank ordered by mean P0 value for the period – Zoomed in on the lowest 12 ranked P0 IP addresses. Top Panel – blue bars – the period average $P0_{conn}$ for 2021-H1 – each bar represents a single `id.orig_h` (e.g. an IP address); Panel 2 – red bars – the period standard deviation of $P0_{conn}$; Panel 3 – green bars – the number of days the

Figure 3.5: Simple Good-Turing frequency analysis summary of individual internal IP addresses with low P0 values observed daily by 5cbe4ac3 2021-H1. Rank ordered by mean P0 value for the period. Blue line – device with IP address `10.0.0.137`, Red line – device with IP address `10.0.0.145`, Green line – device with IP address `10.0.0.173`.

3.4 What devices are low $P0_{conn}$?

Having identified low ranked $P0_{conn}$ (e.g. Figure 3.4) , is it possible, given the data collected, to determine what the devices are? The sources of data collected, in some cases, goes beyond the network monitored logs from *Zeek*. Some of the devices have intermittent vulnerability scans from the *OpenVAS*[14] system.

Unlike passive network monitoring, vulnerability scanning actively interacts with devices to identify known vulnerabilities. The interactions are specified in a dictionary of tests, often developed from known exploits. The *brAIn-box* uses the *OpenVAS* system and schedules vulnerability scans to occur on selected network devices. The reported information is augmented, and then pushed to the cloud for storage and processing. The vulnerability scanner reports a number of fields for each test it performs on each device. A subset of the sample individual record for one of the lowest $P0_{conn}$ devices for which scan reports exist (the device with `id.orig_h = 10.0.0.145`) is shown in listing 3.1[15]:

```
1
2  IP 10.0.0.145
3
4  MAC b827eb5b2331
5
6  nvt_name
7  OS Detection Consolidation and Reporting
8
9  Summary
10 This script consolidates the OS information detected by several
       NVTs and tries to find the best matching OS.
11 Furthermore it reports all previously collected information leading
        to this best matching OS. It also reports possible
12 additional information which might help to improve the OS detection
       . If any of this information is wrong or could
13 be improved please consider to report these to the referenced
       community portal.
14
15 Specific_result
16 Best matching OS:
17 OS: Debian GNU/Linux 9
18 Version: 9
19
20 CPE: cpe:/o:debian:debian_linux:9
```

[14] See: `https://www.openvas.org` OpenVAS - Open Vulnerability Assessment Scanner.

[15] This vulnerability test is attempting to determine what, if any, known operating system is running on the scanned device. An *NVT* is a network vulnerability test – an atomic interaction to determine whether a certain vulnerability exists in a device.

```
Found by NVT: 1.3.6.1.4.1.25623.1.0.105586 (SSH OS Identification)
Concluded from SSH banner on port 22/tcp: SSH-2.0-OpenSSH_7.4p1
    Raspbian-10+deb9u4
Setting key ‘‘Host/runs_unixoide’’ based on this information
```

Listing 3.1: OpenVAS vulnerability report information for 10.0.0.145 having a low observed volatility with respect to all of the devices.

By studying the hardware *MAC* address, `b827eb5b2331`, sometimes it is possible to determine the manufacturer of the device. A website for performing MAC Address lookups is: `https://www.macvendorlookup.com/`.

Looking up the information for device `10.0.0.145` with MAC `b827eb5b2331` returns the information in listing 3.2:

```
MAC Address Details

Company
    Raspberry Pi Foundation
Address
    Caldecote Cambridgeshire CB23 7NU
    UNITED KINGDOM
Range
    B8:27:EB:00:00:00 - B8:27:EB:FF:FF:FF
Type
    IEEE MA-L
```

Listing 3.2: Manufacturer MAC address information for 10.0.0.145 having a low observed volatility with respect to all of the devices.

A summary of information about low $P0$ devices including Device Manufacturer and a guess as to what type of device it is, is presented in table 4.

We do know one device that is definitely in the network and is Raspberry Pi based – the network monitoring *brAIn-box*. The *brAIn-box* produces a daily diagnostics report which is stored in the cloud, and includes the IP address that it is using. The relevant contents of the diagnostics file are shown in Listing 1 (in the Appendix).

`inet 10.0.0.145` confirms that the low $P0_{conn}$ device is indeed the network monitoring device, the *brAIn-box*.

3.5 Investigating a high volatility device

Although the primary purpose of this work is to identify low volatility network devices, for balance, we investigate one device that is high volatility. The device

that is chosen is `id.orig_h = 10.0.0.8`. This is a device that has been on for the full duration. It is the device with the rightmost tallest bar in the lower plot of figure 3.

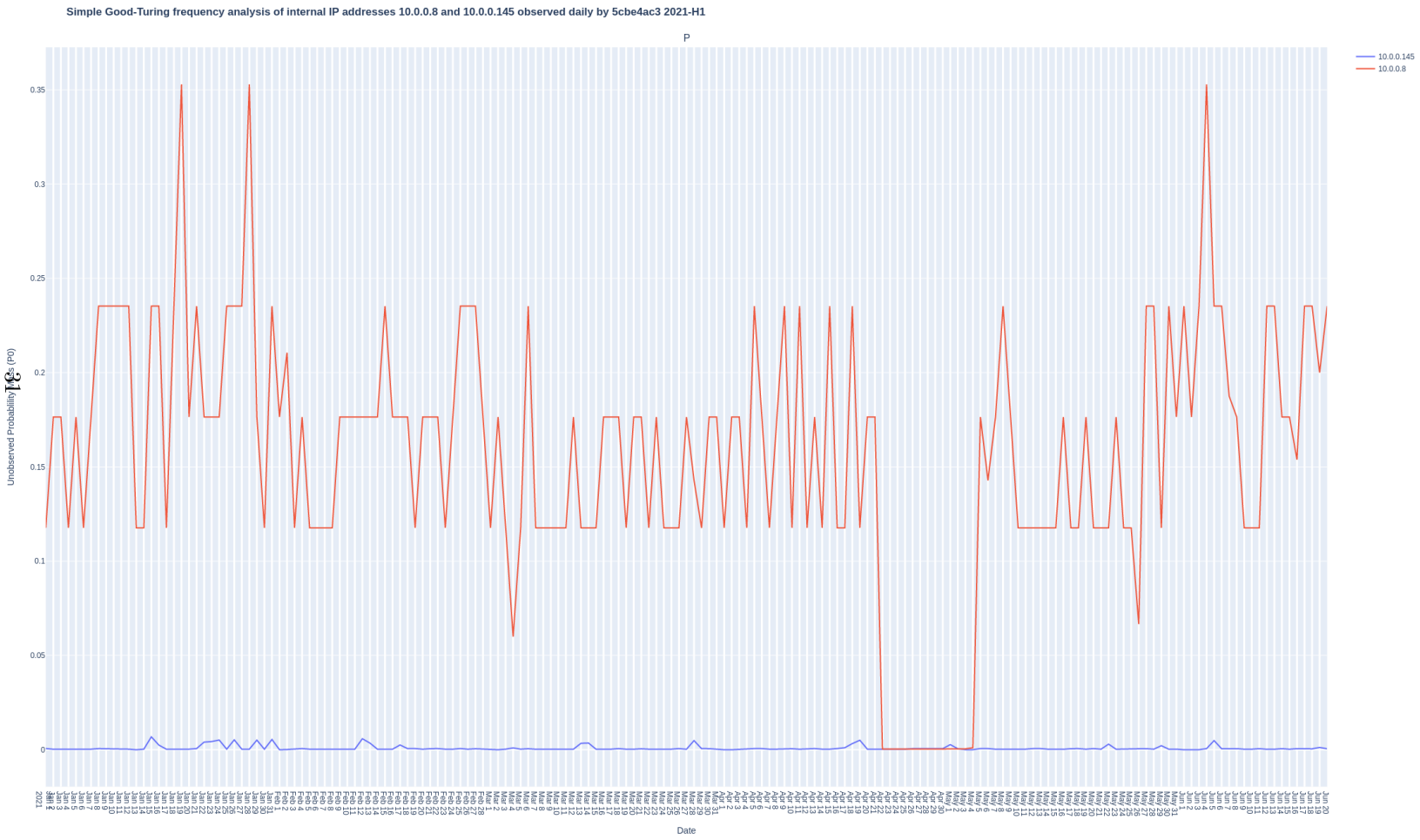

Figure 3.6: Simple Good-Turing frequency analysis summary of internal IP addresses 10.0.0.8 (high P0 values) and 10.0.0.145 (low P0 values)observed daily by 5cbe4ac3 2021-H1. Blue line – device with IP address `10.0.0.145`, Red line – device with IP address `10.0.0.8`. Note: The P0 axis is linear (not logarithmic as in previous plots).

```
IP  10.0.0.8
MAC 7483c24c00f9
nvt_name  OS Detection Consolidation and Reporting
Summary This script consolidates the OS information detected by
    several NVTs and tries to find the best matching OS.nn
    Furthermore it reports all previously collected information
    leading to this best matching OS. It also reports possible
    additional information  which might help to improve the OS
    detection.

If any of this information is wrong or could be improved please
    consider to report these to the referenced community portal.
Specific_result Best matching OS:nnOS:          Linux/UnixnCPE:
           cpe:/o:linux:kernelnFound by NVT:
    1.3.6.1.4.1.25623.1.0.105112 (Dropbear SSH Detection)nSetting
    key "Host/runs_unixoide" based on this information
```

Listing 3.3: OpenVAS vulnerability report information for 10.0.0.8 having a high observed volatility of all the devices.

More information about the high volatility device is gleaned by the vulnerability scanner's report shown in listing 3.3. Recall that an NVT is a network vulnerability test – an atomic interaction from the vulnerability scanner to determine whether a certain vulnerability exists in a device. Some more information can be extracted from public databases holding records on device manufacturers and the MAC address ranges assigned to them. Listing 2 supports the hypothesis that the device is a WiFI access point or router (hence being always on). This analysis has not reduced the possibilities in the same manner as the previously investigated Low $P0$ devices (see section 3.4).

3.6 SGT Discussion

The analysis, above, is a form of forensic investigation – providing a case study into the use of the SGT volatility metric $P0$ applied to NetMon event data as recorded by *Zeek* ([Zee21]).

With respect to the first hypothesis regarding whether $P0$ can be used to identify fluctuations in gross network network-as-whole traffic over time, the answer is a cautious *Yes*. With respect to the second hypothesis regarding whether $P0$ can be used to identify individual devices that show markedly lower (or higher) $P0$, the answer is a more confident *Yes*.

The gross traffic patterns for the NetMon data shown in figure 3.2 indicate

independence between the $P0$ and daily count. This suggests that the SGT $P0$ provides a useful indication of aggregate network traffic volatility.

It is possible to determine low $P0$ devices using recorded network connection events and recover the IP Address that the devices are using on the network. However, it is more challenging to identify what the device is. There is no current systematically adhered to standard for device identification[16]. The physical address – media access control address (MAC address) – is not recorded by Zeek's passive network monitoring, and furthermore, it is not always reliable as the MAC address is easily spoofed. Obtaining the MAC address of a network device can require active interaction with the network (e.g. probing, or even becoming a network transmission point).

Even with a plausible MAC address for a device, the information it contains can only get the device manufacturer with any real certainty. Many device manufacturers produce many types of network device, and without access to the manufacturer's records, precisely identifying their device becomes guesswork.

[16]The emerging MUD [GM20] and D3 [ea21a] standards seek to redress this.

Chapter 4

SEQUITUR Grammar induction and Device Behavioural Cloning

Behavioural cloning is the process of recording the performances of skilled agents and using induction algorithms on the traces of the behaviour so as to generate models that behave in a similar manner as the original agent ([BS95]). Typically the agents under consideration are skilled humans, but here we are interested in inducing models of device behaviour. For this purpose we study the network event traces and the grammatical induction system SEQUITUR [NMW97].

4.1 SEQUITUR Background – part 1

In this work we study *events*[1] as recorded by monitoring network traffic between devices. We are interested in communication network events that are discernible from the signals between devices on a computer network. Network events are atomic records that have a timestamp associated with them. For all practical purposes the timestamps are unique to each event record, and therefore convey a sequence ordering. In these initial investigations, we will (temporarily) ignore the timestamps, and only consider the sequence ordering between events.

Sequences are recognisable in many forms. For example, sentences are

[1] From Zizek [Ziz14] " . . . an event is thus the effect that seems to exceed its causes . . . "

sequences of words; words are sequences of characters. Some sequences contain repetition – recurring subsequences. Some subsequences have known recognisable forms, and it is natural to formally define grammars to specify conformant sequences. For example consider the overly simplified English grammar fragment:

> "A sequence of words is a *sentence* IF it STARTS with a sequence of words that is a *noun phrase* FOLLOWED by a sequence of words that is a *verb phrase*. A sequence of words is a *verb phrase* IF it STARTS with a word that is a *verb* and is FOLLOWED by a sequence of words that is a *noun phrase*. ..."

Grammars enable the determination of whether a sequence is *acceptable* to a specific grammar or not. In this work we seek to determine *what sequences of network communication events are deemed acceptable* (or not) from a cyber security perspective.

In this work we use the `SEQUITUR` compression algorithm from Neville-Manning and Witten[NMW97]. It is a lossless compression algorithm with a computational complexity linear in the size of the input (i.e. the size of the file to be compressed). Furthermore, the compressed file contains a human-readable data structure that explicitly encodes for frequent sub-sequences in the original file. From [NM21]

> "Sequitur is a method for inferring compositional hierarchies from strings. It detects repetition and factors it out of the string by forming rules in a grammar. The rules can be composed of non-terminals, giving rise to a hierarchy. It is useful for recognizing lexical structure in strings, and excels at very long sequences."

The *grammar rules* formed by `SEQUITUR`, in the process of it inferring compositional hierarchies, are the data structures that we use as representations of recurring subsequences of events.

4.1.1 Compression as Machine Learning

One way to consider machine learning is that it is the (algorithmic) extraction of patterns from data. Another view is that machine learning is about extracting generalisations from specific examples. The related Minimum Description Length (MDL) principle is that "what is learned by a machine learning scheme is a kind of theory of the domain from which the examples are drawn." [FHW16].

From Ockham's Razor we prefer the shortest theory with the same explanatory power, which means that MDL encourages the selection of the shortest theory that allows us to reconstruct the original example data. This is similar to the objectives of a compression algorithm that allows us to regenerate the original (data) file from the generalised (or compressed) file.

We can loosely formalise this somewhat with equation 4.1 below. Compression is achieved if the size of the input data E (or file) is greater than the sum of the size of the expansion rules T (representing patterns in the input) and the size of the exceptions not covered by the expansion rules ($E \setminus T$). Furthermore, we can re-cast this to a machine learning perspective and consider T to be the *theory* that we have learned from the training data E. Komolgorov complexity theory [LV08] shows that optimal encodings exist to measure these different components (but that there is no way of determining that the selected encoding is optimal). Some formulations of the Minimum Description Length principle also include the size of the (Turing) machine required to decompress the the theory (i.e. to execute the rules) on the right hand side of equation 4.1. A greater complexity decoder penalizes the compression factor.

$$|E| \geq |T| + |E \setminus T| \tag{4.1}$$

Naturally, truly random examples (or sequences) result in no effective compression and consequently no useful generalisation or learning can reliably take place.

4.1.2 `SEQUITUR` Document outline

The remainder of this document considers the background of the `SEQUITUR` algorithm and a simple worked example. We then describe an empirical study of network monitoring data using `SEQUITUR` to extract patterns from recorded network events. A visualisation leveraging the `SEQUITUR` induced grammar is presented. Finally, some preliminary results are discussed and a conclusion and directions for future work are provided.

4.2 `SEQUITUR` Background – part 2

`SEQUITUR` [NMW97] is a loss-less compression algorithm with run-time complexity that is linear in the number of input tokens. It infers compositional

hierarchies from strings[2] which are represented as grammars for expanding back to the original input string (or token sequence). In this work we use the `C++` implementation[3] of `SEQUITUR` available at `http://www.sequitur.info/`. The compressed output is a form of an expansion grammar rule set. This rule-set is human readable and explicitly describes the frequently occurring sub-sequences that were identified in the to-be-compressed input. An example showing the `SEQUITUR` inputs and outputs can be found in appendix .3.1.

The grammars produced by `SEQUITUR` are compositional hierarchies. The grammar rules generated from the input string "abcdabcd" are depicted in their hierarchy in figure 4. In this simple illustrative example, there is a solitary tree with root node *rule (0)*, below which are the expansions utilising (possibly) other referenced rules. The leaves, ordered left-to-right, provide the original input sequence. In the general case, there may be multiple trees emanating from the start rule **S0** with leaves also at that level. Leaves need not always be at a single level (but they are in the shown simple example's figure 4).

4.3 `SEQUITUR` Experiment with NetMon data

4.3.1 Data Understanding

For this section's experimental work we are focused data extracted from fields from the `conn` table only[4]. The fields and sample values are shown in table 1. For event records pertaining to network connections of the particular device at IP address 10.0.0.145, the fields `id.orig_h`, `id.resp_h`, `id.resp_p`, `proto`, and `service`, are projected into a concatenated string – referred to as a **token**. This string-based token is then hashed to a (unique) 9 digit integer to produce a **token_id**. This data preparation process is described in more detail below in section .4.2.

4.3.2 Data Preparation

In this section we describe the extraction and preparation of source data so that it can be used by `SEQUITUR` to extract sequence patterns. Note that for

[2]The term *string* is overly restrictive. Implementations of the `SEQUITUR` algorithm can also be used to compress *sequences* represented as numeric tokens.

[3]The command-line help for `SEQUITUR` is reproduced in the Appendix.

[4]Future work may analyse log tables other than `conn`.

SEQUITUR timestamps are not required as it is used in a manner where only sequence ordering is considered[5].

In section 3.4 we identified a specific network device which demonstrated both (1) low network volatility (i.e. easier to predict its network communications behaviour); and (2) with a known IoT-like purpose[6]. This device was operating with IP address `10.0.0.145`. This was selected as the device for extracting sequences and is the basis for the data that is described below. Initial data analysis has been restricted to an arbitrary single day of network recorded traffic for 2021-01-10.

The following steps were performed to prepare data into a format acceptable to SEQUITUR.

1. Use Zeek's **conn**ection table to build string tokens for each connection event. The string tokens were a concatenation of specific fields which were chosen based on the experience of a network traffic analyst. This step is one part of an SQL query which can be found the Appendix in listing 25.

2. Generate unique MD5-based 9 digit numeric `token_uid` to uniquely identify each string token. SEQUITUR can utilize (up to) 9 digit integers as identifiers. This step is the final part of an SQL query which can be found in listing 25. This information is exported, along with some other raw information, as a CSV file.

3. (Initially) ignore timestamps, and collect the sequence of `token_uids` into a single text-based training file – with one `token_uid` per line

The process (above) is illustrated with an example in figure 5 in the Appendix.

The resulting source data file is from network monitored meta-data recorded on 2021-01-10; where IP address `10.0.0.145` is mentioned as either *initiating* the network connection or *responding* to the network connection. In this period there were 3613 **conn**ection events resulting in 3613 string tokens with matching unique `token_uids`. The first 10 and last 10 `token_uids` are shown in listing 5 in the Appendix.

[5] Incorporating timestamps into the analysis is considered in the Future work Section

[6] In fact, the device was the network monitoring device that recorded the network communication event logs.

4.3.3 Modelling by Extracting sequence patterns from NetMon data

This section outlines how SEQUITUR is used to build a hierarchical grammar from the input file described above. Full details can be found in appendix .3.

The token_uid column of the input CSV file containing the token UUIDs is piped into the SEQUITUR executable using the command in listing 6. The resulting SEQUITUR output long (but shorter than the original input) – the first 10 lines are show in listing 7, and the final 10 lines are in listing 8.

4.4 Explaining the SEQUITUR output

One of the advantages of the SEQUITUR approach is that it produces human readable output[7]. To make sense of the output the following *legend* is instructive:

- Raw token_id – [<9 digit integer>] is the raw token_uid from the input file. If it appears in a SEQUITUR *rule*, then the raw token_uid was not able to be compressed in its input location.
- SEQUITUR Rule Id – An integer appearing without the square brackets [] refers to SEQUITUR's *rule id* (and refers to the number of the grammar rule produced by SEQUITUR).

4.4.1 The Start Rule S0

The Start Rule **S0** is the remaining sequence once the hierarchical grammar has been constructed. Each integer in **S0** is the number of a **Grammar Rule** (SEQUITUR Rule Id; that needs to be fully expanded to recreate the original raw sequence) and a nine digit integer enclosed in square brackets is a **raw token_id** (that did not appear in a repeating sub-sequence at that point in the input sequence).

The start rule for the example data is shown in listing 4.1. The first 17 elements are rule numbers (1-16), and the 18^{th} is the raw token_id 840284070 (which represents the raw token 10.0.0.145_27.124.125.251_123_udp_-).

[7]The SEQUITUR grammar rules have their own grammar. This is provided as a BNF specification in listing 6 in the Appendix.

```
0 -> 1 2 3 3 4 5 6 7 8 9 10 11 12 13 14 14 15 16 [840284070] 9 17
    18 19 20 21 21 21 22 16 23 24 25 1 8 26 26 27 28 [840284070] 29
     30 31 32 33 10 34 20 35 11 31 36 36 36 33 37 38 39 35 40 41
    [213948788] [213948788] [271311686] [680399771] [680399771] 42
    42 43 [513489017] [513489017] 35 37 43 35 2 44 44 45
    [840284070] 40 [840284070] 42 [271311686] 11 29 44 46 47 45 48
    49 50 19 51 52 52 52 53 54 55 56 49 19 57 58 58 59 60 61 59 59
    59 62 63 64 61 65 65 66 67 68 17 69 66 66 70 71 [382930009] 64
    24 72 [271311686] 72 73 73 74 75 76 24 77 78 79 75 80 81 57
    [271311686] 78 82 83 67 51 2 69 84 70 85 43 50 86 87 88 84 89
    76 [840284070] 77 10 90 80 25 91 [100000001] 20 90 92 48 2 93
    87 94 95 83 96 97 92 8 98 34 50 17 99 100 55 43 86 13 99 101
    102 103 81 104 [350344446] 69 2 105 105 96 101 43 103 8 93 2 43
     [271311686] 30 77 11 68 106 106 50 55 37 12 [628100651]
    [866509023] 107 12 [382930009] 103 107 2 34
```

Listing 4.1: `SEQUITUR` Rule 0 – the *Start* grammar rule.

Consider rule 8 shown in listing 4.2. The rules' primary purpose is for being able to decompress (the start Rule 0) back to the original input sequence. Rule 8, itself, can be expanded by replacing every occurrence of `8` with '`112 [100000001]`' – that is replacing with whatever rule 112 encodes for, followed by the token `[100000001]`.

Rule 8 also carries with it two statistics (a) the number of times rule 8 appears in the rest of the hierarchical grammar (15 appearances in the hierarchical grammar); and (b) the number of times rule 8 is used in the expansion of the start Rule 0 (71 times in the start Rule 0).

Finally, rule 8 also provides the complete `token_id` expansion that the recursive uncompressing of the rule encodes for. In this example, every 8 in rule 0 should be replaced with `[100000001] [271311686] [100000001] [100000001]`.

```
8 -> 112 [100000001]  15  (71)  [100000001] [271311686] [100000001]
     [100000001]
```

Listing 4.2: `SEQUITUR` Rule 8 – A sample the grammar rule.

At the end of the `SEQUITUR` output some overall statistics are provided thus:
`508 symbols, 130 rules 20984 total space`

Note that the number symbols reported is not related to the number of unique `token_id`s in the input file (which for this data set is 2292).

```
1  RuleID: 18
2
3  Rule: [head(18),token(271311686),rule(162)]
4
5  Expansion: [271311686,847213017,847213017,271311686]
6  Length: 4
7  Frequency in training: 21/3613
8  Frequency in the grammar: 19
9
10 271311686:  10.0.0.145 10.0.0.137 (name_not_found) 53 udp dns
11 847213017:  10.0.0.145 3.209.99.56 (ec2-3-209-99-56.compute-1.amazonaws.com) 22 tcp
       -
12 847213017:  10.0.0.145 3.209.99.56 (ec2-3-209-99-56.compute-1.amazonaws.com) 22 tcp
       -
13 271311686:  10.0.0.145 10.0.0.137 (name_not_found) 53 udp dns
```

Listing 4.3: Rule 18 details.

4.5 Visualisation of SEQUITUR grammar rules

The *Visualisation of input sequence* (figure 11) shows the sequence (of tokens) overlaid on the rules that represent the repeating sub-sequences produced by SEQUITUR.

A way of visualising the raw sequence of tokens is to make a two dimensional tiling using token colour tiles. A *token colour tile* is a square of a single colour, where the colour is determined from the ID of the token[8]. A *rule colour tile* is a rectangle of a single colour, where the colour is determined from the ID of the rule[9]. A white background is a white rule colour tile that indicates that SEQUITUR did not find sufficient recurring sub-sequence using that particular token in the input sequence (i.e. no compression at that point).

In the sequel Rule 18 (see listing 4.3) will be used as a running example. The full expansion for rule 18 is a sequence of four tokens whose token IDs are – 271311686, 847213017, 847213017, and 271311686. We can visualise this by overlaying the token tiles for tokens 271311686 and 847213017, on the rule colour tile rectangle for Rule 18. Figure 4.1 shows the visualisation of Rule 18. In the figure 4.3 we can see some of the instances of Rule 18 in the input sequence which have been highlighted (manually) with a red rectangle.

A simple tool has been developed for visualising sequences and the grammar rules produced by SEQUITUR – *Simple Sequence and Sequitur Grammar explorer*[10]. It provides a way of browsing the original input sequence, guided by information overlaid from the SEQUITUR sequence extraction. There are three main areas of the tool:

[8]Tokens with the same ID have the same colour. Different token IDs have different colours.
[9]Different rule IDs have different colours.
[10]This prototype has been coded in the *XPCE* [WA02] graphical extensions to *swi-prolog* [WSTL10].

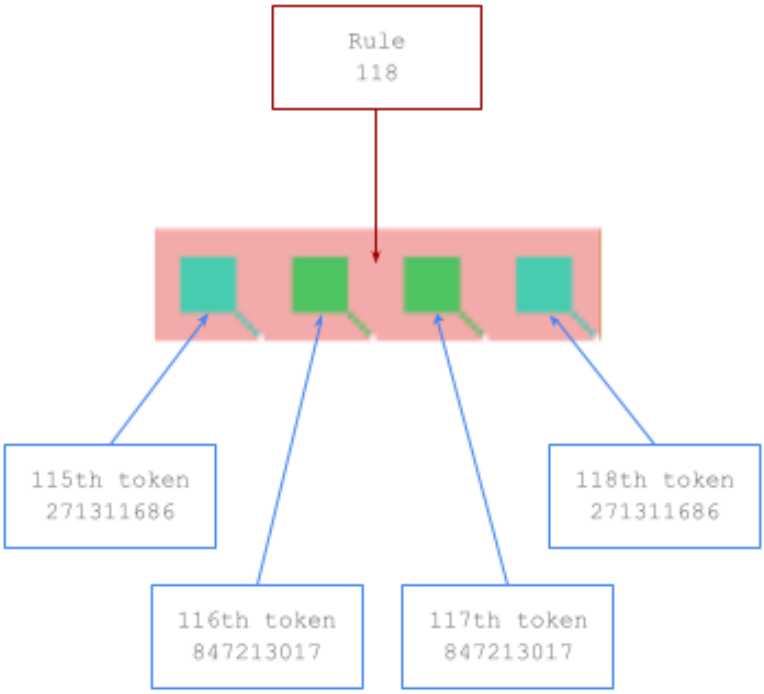

Figure 4.1: Simple `SEQUITUR` sequence viewer – an example rule.

1. Hierarchical `SEQUITUR` grammar navigator
2. Visualisation of input sequence
3. Item Details

For illustration, the screen shots and figures that follow relate to the use of the *Simple Sequence and Sequitur Grammar explorer* using the input sequence file `10-0-0-145_2021-01-10.csv`. This file has been produced from the NetMon data recorded by Zeek [Zee21] and extracted from a relation database using the SQL query in listing 25. It relates to network interactions initiated or responed to by a device with IP address 10.0.0.145 for the 24 hour period 2021-01-10.

Figure 4.2: Simple `SEQUITUR` sequence viewer. A tool for exploring sub-sequences and the hierarchical compositional grammar rules generated by `SEQUITUR`.

4.6 Other visualisation adornments

In this work we are interested in *events* which typically are associated with absolute points in time. In studying the *sequence* of events we drop the timestamps and focus only on the *ordering* of events[11]. This is the approach that is used to deploy `SEQUITUR` and build the *Simple Sequence and Sequitur Grammar explorer*.

Having discarded absolute temporal information and relied on event ordering to produce a visualisation, how might we incorporate some absolute temporal information?

4.6.1 Top-of-the-hour Markers

Might a Rule sub-sequence recur at the same relative time? To elucidate this question, we add a simple marker to the visualisation to indicate which is the first token tile to occur after the a new hour.

For example, figure 12 in the Appendix, identifies the token (35034446 at position 945 out of 3613) to be the first token in the input sequence to occur

[11] Dropping timestamps for sequence visualisation purposes is also used in [NTA+18].

after 06:00 UTC.

4.6.2 Segment Start Markers

How are the token arrival times of tokens clustered within a Rule sub-sequence? Do they clump towards the timestamp of the first token of the Rule sub-sequence? Are they evenly spread throughout the Rule sub-sequence?

To explore these and allied questions, we consider a simple One Dimensional clustering method and indicate the start of such clusters on the visualisation. The algorithm (from [tyr21]) has a single parameter ϵ which is used for identifying clusters. It simply iterates along a sorted list of the token timestamps and assigns the events to a new cluster if the two adjacent event timestamps differ by more than ϵ, making the last of the two events the first of a new cluster. If the difference in adjacent timestamps is less than ϵ then both token timestamps are in the same cluster. Illustrative `Python` code is given in listing 9.

We add a simple red underline marker to the visualisation to indicate which is the first token tile at the beginning of a cluster. For the analysis depicted, ϵ was set to 5.0 seconds. The cluster markers on the visualisation is shown in figure 13 in the appendix.

4.7 NetMon Case Study `SEQUITUR` Results

This section captures early experience in using the techniques and tools presented applied to the network traffic monitoring data that is described in section 5.3.

4.7.1 Explainable Patterns

Can the visualisation be used to identify a known repeating pattern in NetMon data captured from a single device? One of the repeating sub-sequences extracted by `SEQUITUR` is that of rule Rule 18 (see listing 4.3). Figure 4.3 shows a portion of the visualisation of token tiles which includes the rule 18 occurring five times (manually highlighted in red rectangles). The portion of the NetMon log that is used is 2021-01-10 00:00:25.957 (the timestamp of token tile 51) to 2021-01-10 06:00:28.900 (the timestamp of token tile 1000).

From the visualisation we observe that the every occurrence of rule 18 is also the start of a temporal cluster bounded by $\epsilon = 5.0$ seconds. This is illustrated by

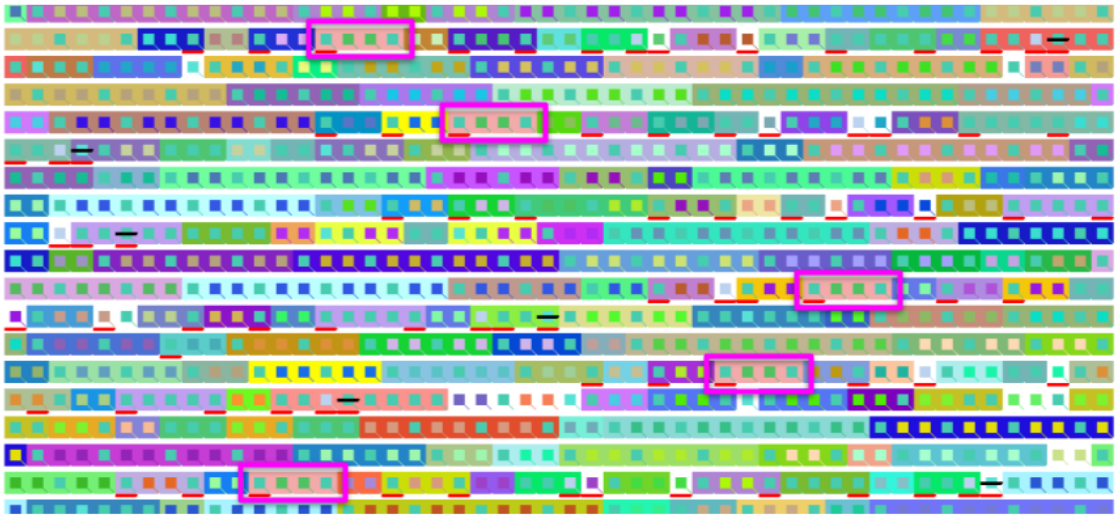

Figure 4.3: Rule 18 in the context of the the input sequence of tokens from 00:00 UTC to approximately is 06:00 UTC. Each instance of rule 18's sub-sequence is in its own cluster ($\epsilon = 5.0$ seconds), and *not* at the top-of-the-hour.

the single red underline at the first token tile in the sequence for each instance of the sub-sequence from rule 18.

From the simple hour markers (black lines through tiles) we observe that none of the instances of rule 18 are on the hour. Referring to the original source data with timestamps that relates to rule 18 (see table 9) we see that the events for the sequence start very shortly after 10 minutes past the hour. We also notice that `SEQUITUR` seems to have formed an over-specific rule, rule 72, which is a specialisation of rule 18 (having additional suffix tokens `733850792, 733850792`). Rule 72 accounts for the rule 18 pattern at the 02:10 time-slot[12].

4.7.2 White-box cross-referencing

Does Rule 18 relate to any known functionality? The NetMon data source is from a device that we have access to the design and code sources. It is a simple IoT-like device having many of its software processes being controlled by a simple clock/calendar-based scheduler[13]. Studying the scheduling code we find that

[12]Rule 72 can be seen in figure 4.3 starting 3 rows below the last token of the second occurrence of rule 18. The tokens are the same colour but the background rule colour is a green, of similar shade to the middle tokens.

[13]The system uses `cron` as the scheduler.

ID	IP Response Host Name	Count of IP Addresses	Comment
1	'ec2-3-209-99-56.compute-1.amazonaws.com'	1	Device support line
2	'name_not_found'	8	Internal network
3	'ntp2.ds.network'	1	Time service synchronisation
4	's3-ap-southeast-2-w.amazonaws.com'	169	Cloud file storage

Table 4.1: IP Resp Host names and mapped IP addresses in the NetMon data. *'name_not_found'* items are internal IP addresses for which there are no public DNS records. Note that the service using *'s3-ap-southeast-2-w.amazonaws.com'* switches between 169 actual IP addresses within the 24 hour recorded NetMon data period.

there are processes that are triggered by the `cron` listing 5.1 in section5.5, which specifies that the software process is triggered every hour of every day, at 10 minutes past the hour. The process contacts a named resource – which at the time of recording was on IPV4 address `3.209.99.56` – for service updates. This correlates with the `SEQUITUR` extracted sequences from rule 18 and friends that refer to token ID `847213017`.

4.7.3 Other Patterns

Having white-box access to the device provides information that assists in determining the patterns that should be identifiable in the NetMon data. In the collected data, most network interactions are with a single cloud storage resource (mostly uploads with a few downloads). That cloud resource has a single persistent URI, however the cloud service provider uses a load-balancing strategy that means the DNS resolution of a named URI to an IP address changes from moment to moment. Of the 179 unique IP Response Host addresses seen in the data, 169 relate to that same cloud service (see table 4.1).

Why might this be an issue? Different tokens for events with the same semantics prevents `SEQUITUR` from extracting useful sub-sequence patterns.

Chapter 5

Logical XAI for Cyber Defense

5.1 Logic for Human-Computer interaction and reasoning

Forms of mathematical logic have been used for capturing knowledge and using in computer systems from the the earliest ideas. As early as the 1870s Jevons had developed a mechanical system using levers and pulleys [BC06]) that implemented a logical system subsequently to be rediscovered nearly a hundred years later ([Rob65]) and used as the foundations of Logic Programming. Early expert systems and other approaches at codifying common sense also used logical formalisms ([McC59]). Indeed, which it comes to a strategy for building a *Thinking Machine* Turing sees logic as a fundamental component ([Tur50]).

Logic provides a way of explicitly encoding knowledge with well defined semantics. A form of logic primarily for knowledge sharing and reasoning has been used for more than 20 years to power the WC3's *Semantic Web* ([KM01]). There is a growing number of tools to construct knowledge-bases, including ways of mapping natural language making it simpler for Human-Computer interaction ([DNPV22]).

The forms and power of reasoning vary. Most often the form of logical reasoning used is *deduction*, being a relatively efficient and safe way of drawing conclusions from what trusted information that is already known. Deductions

take known *general laws* and *cases* to derive sound *results*. However humans do better, reasoning with incomplete, and sometimes contradictory evidence, more akin to *abductive* reasoning. Abductions take known *general laws* and *results* to speculate on unknown *cases*. Abductive Logic, and its implementations (e.g. [ACS+18]) are making some initial contributions to cyber defence ([MAM23]).

A final form of reasoning is *induction* takes as input *cases* and *results* and hypothesises general laws. The next section introduces Inductive Logic Programming as a machine learning paradigm for implementing the induction of logical rules from example data. This has been used in a range of cyber security problems ([Ko00], [MH03], [RHM16], [RM21]).

5.2 Machine Learning in Logic: Inductive Logic Programming

Inductive Logic Programming (ILP) [MR94] is a form of symbolic machine learning where all aspects of the learning problem are specified and modelled as fragments of logic programs – typically as `Prolog` programs. The central aim in ILP is to ensure transparency of the inputs to, and the models produced, as well as enabling the results to be *comprehensible* for humans. This makes the approach well suited to interactions with domain experts in their quest to explain some phenomena.

A simple tutorial illustration demonstrates how Inductive LP differs from Deductive LP is shown in figure 5.1. The conditions for ILP, where B is the existing background theory (in the example these are known relationships in the family domain; but in general can be anything); $E^{+,-}$ are the positive/negative examples (in the example there are two positive examples of the relationship grandfather – and no negative examples); H is the hypothesis or explanatory model (to be generated).

The normal formulation of the conditions for ILP is as follows:

- ILP condition 1 - Prior Satisfiability: $B \cup E^- \not\models \square$. That the existing background knowledge is not inconsistent with the negative examples.
- ILP condition 2 - Posterior Satisfiability: $B \cup H \cup E^- \not\models \square$ That the produced model H when added to the background knowledge and the negative examples is not inconsistent.

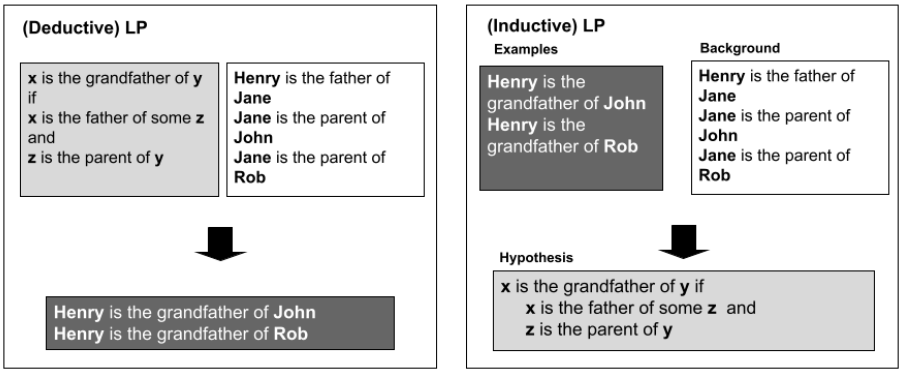

Figure 5.1: Tutorial illustration of Inductive Logic Programming. The left-hand panel – *(Deductive) LP* – depicts the normal deductive usage of Logic Programming where sound conclusions are derived from general rules about the grandfather relationship and specific facts. The right-hand panel – *(Inductive) LP* – depicts the (potentially unsafe) production of a general law about the grandfather relationship, produced from specific examples of grandfathers and other associated background facts.

- ILP condition 3 - Prior Necessity: $B \nvDash E^+$. The positive examples are not already modelled (explained) by the existing background knowledge.
- ILP condition 4 - Posterior Sufficiency: $B \cup H \models E^+$. When combined with the existing background knowledge, the model H explains the positive examples.

We also try to minimize, as much as reasonable, the number of negative examples that can be explained through the a form of the Posterior Sufficiency condition (4), (i.e. $B \cup H \nvDash E^-$).

5.2.1 Mode Directed Inverse Entailment, ALEPH, and ACUITY

Much of the foundations of Machine Learning and early AI are based on search and optimisation. ILP is no different, after all, we are seeking *good* explanatory hypotheses (models) from a potentially infinite search space. The first task is to define the search space in a principled manner, then sensibly restrict it, and finally, to computationally search the space and return useful hypotheses, H.

The key task is to *generalise specific* examples. The logic of generalisation mirrors that of specialisation (See: [Plo71]). For a particular example "Henry is the grandfather of John" we can replace the specific names (Henry, John)

with (universally quantified) variables (say x and y)[1]. Clearly, this is an over-generalisation, as not every person x is the grandfather of every other person y.

In this logical formulation, there exists a *lattice* of alternative hypotheses between the very specific example (e.g. `grandfather(henry, john)`) and its over-general fully variablised counterpart (e.g. `grandfather(X, Y)`) We use *Mode Directed Inverse Entailment* [Mug95] to find an anchoring hypothesis at the bottom of the lattice (known as the *most specific clause* or *bottom clause*), which relates the *seed* example to the supplied background knowledge (B). A (beam) search through the subsets of the *bottom clause* is typically done from the top (shortest hypothesis) downwards – testing each candidate – until acceptability criteria are met. The implementation used in this work is Srinivasan's `ALEPH` ILP system [Sri17].

`ALEPH` provides an extensive set of features and capabilities, requiring configuration parameters to be specified that relate to the specific machine learning context and task. Normally, this process is done in a batch fashion, but in this work, an interaction with a domain expert, in which they constantly feed in more guidance and expertise as the search for an explanation that is acceptable to them emerges. `ALEPH` has some support for such an interactive usage mode, but extended features are provided by the `ACUITY` wrapper system ([RHM16], [RM21]).

5.3 Experiment to Logically Explain NetMon data

This is an empirical line of enquiry based on previously captured eavesdropped communications data. The case study uses collected communications data from a small business office (as loosely caricatured in figure 2.1) from January 1st to June 20th 2021 (inclusive) – a period of 172 days. During the observed period there were between 28 and 185 active networked devices per day, resulting in between approximately 63K and 562K logged connection events per day.

In the sequel we will follow the CRISP-DM six-phase model (see: [CCK+00]) to build the explanatory model of a network event sequence.

[1] In `Prolog` this would be `grandfather(henry, john)` becoming `grandfather(X, Y)`.

5.3.1 Problem Understanding

The problem that is being addressed in this work is the identification of IoT-like devices operating within a network, and the explanation of related network events that can be incorporated into a (partial) specification of what the device *normally* does.

Often, a key challenge for securing IoT devices is first identifying their presence on the network (as asset registers do not exist, or they have not been properly maintained). Previous analysis of the data ([Moy22]) has identified that the *volatility* of network connections to/from a device provides an indication, relative to other devices on the same observed network, of whether it belongs to the class of *IoT-like* devices. From that analysis the device with IP address `10.0.0.145` had very low volatility and was selected for further the explanatory analysis which follows.

5.3.2 Data Preparation

The data transformation process is depicted in figure 2.3. The following summarizes the process for building the inputs to the XAI machine learning system.

- Select a days Zeek data for the identified device
- Mostly focus on Zeek's `conn` table
- Build string tokens combining the `conn` table fields: `id.origh_h` – source IP address, `id.resp_h` – destination IP address, `id.resp_p` – destination port number, `prot` – identified protocol, and `service` – identified service for each network connection that the device has.
- Hash the string tokens to a unique 9 digit number as connection IDs.
- Input the sequence of 3613 (check) connection IDs into `SEQUITUR` to generate the hierarchical compression grammar.
- Use the sequence visualisation to identify interesting (repeating) subs sequences as examples for XAI machine learning.
- The following items are then utilized by the XAI
  - Raw Zeek connection table event records;
  - Extracted Zeek dns and dhcp table event records;

- Sequence pattern data extracted by the SEQUITUR system (cite some previous report) including a
- sequence grammar and supporting data elements.
- Hand produced Prolog code providing logical relationships between entities.

Detailed Data Understanding and Preparation information can be found in appendix .4.1

5.4 Explanatory Modelling

In this section we outline the modelling that is performed to explain a particular recurring sequence in the netflow event data. The approach is to use Interactive Inductive Logic Programming via the `ACUITY` system (see: [RM21] for more background). The formation of the explanation is a **co-discovery** with the ILP system being guided by a (network forensics) domain expert. For such an interaction between human and machine, it is important that the both parties can comprehend ([Mic88]) any intermediate results, and react to what has been produced in a way that refines the outputs into an *acceptable*[2] explanation.

5.4.1 The Interactive Explanatory ILP Process

The objective of the Interactive Explanatory ILP Process is to combine data, existing background knowledge, and the experience of a domain expert, and produce an acceptable explanation of specific exemplars.

1. **Select** specific exemplar to be explained.
2. Let `ALEPH` build a most specific clause and search the lattice with respect to input data, background knowledge, constraints, and settings – outputting a proposed explanation.
3. **If** the proposed explanation is acceptable, **then** quit[3], otherwise the human communicates to `ACUITY` that the proposed explanation is unacceptable, and makes changes to settings and other constraints, then repeat step 1.

[2]We do not provide a strong definition of acceptability here, only that the domain expert is satisfied with the explanation arrived at.

[3]Or go back to step 1 and select another unexplained exemplar.

5.4.2 Explaining SEQUITUR sequence rule 8

This section documents a proof-of-principle experiment using ACUITY to explain one of the subsequences extracted by SEQUITUR from network monitoring data of device at IP address 10.0.0.145 on 2021-01-01.

The SEQUITUR subsequence exemplar we will study is grammar Rule 8[4] Rule 8 contains 4 elements with token hashes(see listing 29) [363783709, 100233664, 100233664, 363783709]), and occurs 18 times in the twenty-four hour period under study.

5.5 Evaluation

The clause produced by the human-computer learning sesssion (ACUITY Rule 1 above) explains the SEQUITUR rule 8 pattern thus:

> The sequence of events is explained if: the sequence rule starts with a Zeek UUID (B) and that the next event after B is C and the next event after C is D. The event D connects to port service ssh at the server 'ec2-3-209-99-56.compute-1.amazonaws.com', at the beaconing frequency of 10 minutes past the hour.

Note that this explanation has been produced from a single example, with one negative example added, and one negative example inferred through the rebuttal process.

The device manufacturer confirms that the actual device with IP address 10.0.0.145 has code that attempts to establish a connection to the named AWS host every 10 minutes past the hour.

```
1     :
2 */10 * * * * /opt/ai-0.0/bin/ai-aws-ssh.sh >> /var/log/ai/aws-ssh.
      log 2>&1
3     :
```

Listing 5.1: Beaconing ssh connection in device 10.0.0.145's cron.

We can check the coverage of the final theory with respect to the loaded data. We find that both SEQUITUR rules 8 and 191 are predicted by the final theory.

[4]Many of the data relating to Rule 8 are provided as illustrative examples in the appendix in section .4.9.

```
?- findall(A, (rule_start_zeek_uuid(A,B), next_to(B,C), next_to(C,D
    ), has_port_service(D,ssh), has_dest_name(D,'ec2-3-209-99-56.
    compute-1.amazonaws.com'), beaconing_every_hour(A,10)), As),
    list_to_set(As, Ss).
As = [8, 8, 8, 8, 8, 8, 8, 8, 8|...],
Ss = [8, 191].

?-
```

Listing 5.2: Coverage of final theory.

Further inspection of rule 191 shows that it is an extension of rule 8 (and is probably produced by an over specification[5] by SEQUITUR).

```
?- seq_data:rule(191, Rule).
Rule = [head(191), rule(8), rule(307)].

?- seq_data:grammar_frequency(191, GFreq).
F = 2.

?- seq_data:training_frequency(191, TFreq).
F = 2.

?- seq_data:rule_expansion(191, Seq).
Seq = [363783709, 100233664, 100233664, 363783709, 547401371,
    547401371, 363783709].

?-
```

Listing 5.3: SEQUITUR rule 191 details.

5.5.1 Deployment

As this project was a proof-of-principle only, there has been no deployment to date.

5.6 ACUITY Discussion

This work has produced a logical and comprehensible explanation for network flow event meta data sequences observed from an IoT-like device, using a co-learning approach – by symbolic machine learning joining forces and being guided interactively by a cyber security domain expert.

[5]We prefer over specialisation by the distribution free SEQUITUR system so that the generalisations can be produced by ACUITY.

Network security is a domain rich in structure and knowledge. From networking and its architectural aspects that define connections between devices, to layered communications protocol specifications and their implementations. Combined with cyber security aspects including threat actors and system vulnerabilities. All of this richness makes it a highly sophisticated knowledge domain, where highly skilled human experts are critical to problem solving.

Computer systems are, in theory, deterministic, so information about how any device is operating and communicating can be supported by collecting appropriate data. However, the data quantities available are vast – including full network packet captures and device system logs – making machine learning an ideal approach. But most of the data is unlabelled, and when labels do exist the examples are severely unbalanced leading to models that have high error rates (both false positive and false negative errors).

To effectively explain cyber security network phenomena requires the intelligent combination of these three aspects:

- Human cyber security expertise;
- Data relating to devices and their communications;
- Background knowledge relating to devices and networks; threats and vulnerabilities.

Intelligent combination between human and machine requires effective two-way communication [6] to ensure the human can comprehend what the machine has produced while the machine can respond to what the human can contribute to the problem solving.

We have taken network flow event data, produced a hierarchical grammar of events to extract sequence patterns (using `SEQUITUR` [NM21]). Interesting sub-sequences are chosen to be examples, for which we wish to generate an acceptable explanation. The hierarchical grammar with the event data makes a base layer of available information. Higher level concepts from cyber security defence are also included. In particular, the concept of *beaconing* as part of a command-and-control centre architecture is defined and made available as part of the background knowledge. This is all brought together in the interactive Inductive Logic Programming system `ACUITY`[RHM16] (based on `ALEPH`[Sri17]). A co-learning interaction between a cyber security defender[7], the data and the

[6] Srininvasan and Bain [SBC22] argue for an even stronger position in that both human and machine must be able to *refute* the other's assertions.

[7] The mode of cyber security defence is akin to cyber forensics.

background knowledge, produces an explanation of a particular recurring network sequence of events from an IoT-like devices. The explanation produced identifies the IoT device contacting its controlling cloud service on a fixed, hourly, basis.

Chapter 6

Discussion

Cyber defenders are deluged by data that lacks labels to make machine learning and other artificial intelligence techniques a challenge to apply. The human defenders themselves are extremely scarce and will be significantly enhanced if they can team up with an XAI that amplifies their skills. This requires both the Human and Machine can share with each other what they know and what it is that they have newly determined. In this work we have used network monitoring meta data as a first source of data. This data is transformed and filtered through a *volatility* measure, which provides the ability to identify both periods of operation as well as individual devices that are easier to defend. This is a help as devices that primarily repeatedly do the same (simple) tasks can be understood to the level that allow-list behaviours can be enforced by custom network devices. Battling the theoretically infinite possibility of computable behaviours, the historically used tactic of building deny-lists is an endless task of constant updates, always one step behind the next zero-day attack.

It can be difficult for academic cyber security researchers to obtain real world data. For this work it was possible to utilize approximately six months of operataional network data from a modest sized organisation. The organisation had a variety of devices in their network, but there was only one fully identified device, the network monitoring system that was installed for this observational case study. The full information about the network context and the data collected is presented in section 2 . 185 individual devices were observed during the study period.

In section 3 we show how the *Simple Good-Turing frequency estimator* unobserved probability mass ($P0$) can be applied to the eavesdropped network

metadata. Gross patterns of daily network traffic show periods of less volatility (and there is a strong correlation to periods when traffic and volatility can be expected to be relatively less – e.g. on non-workdays). Average daily network volatility varies substantially throughout the 172 day period with a minimum value of 6.1 microns and a maximum value of 331 microns. In the same period, the number of observed devices varies between 28 and 185, whilst the total number of observed network connections in a one-day period varies from 63,710 to 561,742. The ratio of Originating device connections to Responding device connections is about 1 to 40, and that these two sources of $P0$ are not strongly correlated.

That using observed daily average $P0$ of individual devices show 3 orders of magnitude variation between devices suggests that there is a spectrum of volatility in real device behaviour. It gives some initial support to being able to identify easier-to-defend devices. Post-hoc identification of network devices, without physical access to the devices, is difficult. The analysis of the lowest ranked five devices *suggests* these are devices with repetitive simple behaviours. Unsurprisingly, the network monitoring system itself is a low volatility device, being the third lowest ranked device. As this device is fully specified and identified in advance, it allows any analyses to be compared to its designed IoT behaviour, making it a good target for further detailed study.

Turning to the detailed study of the network interactions of a specific device in section 4 , the `SEQUITUR` compression algorithm is used to perform an initial phase of *behavioural cloning* to an arbitrary days worth of metadata traffic containing 3613 network `conn`ection events. `SEQUITUR` is used as a machine learning algorithm as its internal basis records a *hierarchical grammar*. This grammar consists of rules that encode those repeating sequences. The whole original sequence can be visualised using a coloured tiling that encodes both the individual events and the repeating sequences that they have been identified to be a component of.

The use of the `SEQUITUR` algorithm does not take account of the timestamps, only the sequence of events. This is similar to approaches taken elsewhere also in applied security domains (e.g. [NTA+18]). However, adding back some information into the sequence visualisation is helpful: 1) marking the beginning of each hour; and 2) clustering the events along the time dimension using the original timestamps. The human cyber defender can use the visualisation to identify some (possibly recurring) sub-sequences that they want to explore more deeply. Each occurring instance of an identified becomes an *example* that can

be used for the defender to give to the XAI.

The final phase of generating a behavioural cloning understanding of specific device behaviour is performed in section 5 using the interactive inductive logic programming system `ACUITY` (see [RM21]). This takes as input the example of the sequence of behaviour to be explained, and contextual background knowledge. These inputs are all expressed as fragments of Logic Programs, which are comprehensible to the Human defender, and directly executable by the XAI (which conforms to Michie's description of an *ulta-strong machine learner* [Mic88]). The defender is able to directly refute the XAI's working hypotheses, to shape an explanation that sufficiently fits both the observed data and tacit expertise of the defender.

The result of this co-inductive Human-XAI collaboration session explains the `SEQUITUR` rule 8 pattern from a single example, with one negative example added by the defender, and one negative example inferred through the defender's rebuttal process.

> The sequence of events is explained if: the sequence rule starts with a Zeek UUID (B) and that the next event after B is C and the next event after C is D. The event D connects to port serevice `ssh` at the server `'ec2-3-209-99-56.compute-1.amazonaws.com'`, at the beaconing frequency of 10 minutes past the hour.

This explanation includes the defender's high-level concept of *beaconing* which is a common design pattern used in some IoT devices where they periodically contact a master controller for any pending commands at some regular frequency. The above explanation is rendered in English, but has a direct representation as a fragment of `Prolog`, which can be used to reason with, or even deploy as an allowable behaviour of the device that can be permitted on the network at run time. This has some similarities logical induction of intrusion detection system rules reported in [Ko00], but does not require significant, prelabelled balanced datasets.

Revisiting Michie's version of Explainable A.I. which he calls *Ultra-strong Machine Learning* the following tests must be met:

- **[M1]** *[a] system [that] uses sample data (training set) ...*
- **[M2]** *... to generate an up-dated basis for improved performance on subsequent data ...*

- **[M3]** *... and also can communicate its internal updates in explicit ...*
  - **[M3.1]** *... and operationally effective ...*
  - **[M3.2]** *... symbolic form*

The results reported were generated from a system that used sample data, which originated as network traffic, and was identified and selected by being filtered through volatility, then extracted as recurring sequence patterns. This satisfies Michie's **[M1]** requirement. The beaconing rule – updated basis – was applied to a randomly selected day of that devices network logs, and found that the behaviour appears 24 times at the hourly time stamps forecast. This, then satisfies the **[M2]** requirement of improved performance, as without the rule, the system would not be able to identify the behaviour in unseen data. Dealing with the combined **[M3/M3.2]** – the cyber defender was able to verify the beaconing rule (indeed they helped create it) in the symbolic natural language formulation above, thus demonstrating that the system was able to communicate its internal update to a human. Finally, the combined **[M3/M3.1]** we see that we can directly utilize the isomorphic `Prolog` form of the beaconing rule, in both identifying the behaviour in subsequent network logs, and also as part of a more fine-grained defensive system. Furthermore, we can add this operationalization to the manufacturer's specification for that device class (see [GM20] [ea21a]) for use in other deployment contexts.

Although we have demonstrated XAI Human-Machine collaboration applied to network security, it still falls short of replacing human cyber defenders. The results above utilise restricted reasoning that is limited to deduction and some induction. Cyber defenders in reality must apply their skill and reasoning in much the same way that a detective solves a murder case. They constantly form hypotheses, collect more evidence, and focus their investigations. They will use *abductive* reasoning [Pei96] to form hypotheses and seek yet-to-be discovered evidence, whilst forming constraints based on the potentially contradictory evidence that will rule out incompatible hypotheses and lines of enquiry. They may be able to conclude, based on reasonable probability, the motive, opportunity, and weapon used by a suspect. Once the case is solved, then it is possible to follow a *deductive* reasoning process that explains the case and its artefacts. Abductive reasoning is difficult to achieve using standard logic programming techniques (e.g. like `Prolog`). Advances in *Answer Set Programming*([Lif22]) provide for environments that describe their world in a comprehensible form

(e.g. logic) whilst permitting missing and or contradictory evidence. Some initial explorations have been made in the field of cyber security in the early `s(CASP)` environment ([MAM23]). Providing high-level symbolic interaction in natural language continues to be explored in the `s(CASP)` system (see: [SDFP23]).

Chapter 7

Conclusion and Future Work

To centre this work work clearly in the realms of explainable A.I. we have chosen to align with Donald Michie's definition of *Ultra-strong Machine Learning* ([Mic88]). As his definition insists at the outset on comprehensible internal models that are learned by the A.I. there is no need to reverse-engineer ([GVWT21]) any black-box system to understand what it might do when making predictions.

Supported by an empirical case study we demonstrated that passively collected Network Monitoring event data can be used to identify the volatility of communications of network devices. It provides further evidence to motivate a systematic approach to the identification of individual devices Network traffic volatility as provided by the Simple Good-Turing frequency estimator ([Goo53], [GS95]) shows empirical evidence that there is a spectrum of easier to defend network devices. This supporting Meer's conjecture [Mee15] in that it is very difficult to defend whilst much easier to attack.

The empirical case study has also demonstrated that the symbolic compression algorithm, `SEQUITUR` ([NMW97], can be used as a sequence pattern extractor from a stream of temporal network events. Furthermore, an elementary method for visualising the sequence patterns provides a basic human-computer interface for navigating the network data. Humans can explore the network event interaction sequences via visualisations, and select targets for further investigations

The symbolic compression techniques can be used as a form of behavioural cloning ([BS95]) by extracting hierarchical grammars to describe sequences of network interactions. The grammars provide a component of background knowledge that can be deployed in an interactive inductive logic programming

system (ACUITY [RHM16]). Further rich knowledge knowledge about the cyber security domain and specific network security context can be added. For example, high-level concepts that cyber defenders understand, like the behaviour pattern of device beaconing, can also be incorporated into the discourse. We demonstrate that a cyber defender can have a robust comprehensible direct interaction with the XAI machine learning environment to guide it towards a mutually understood explanation of the sample network phenomenon selected from the event logs.

The resulting knowledge produced was compared with the ground-truth knowledge from the generating device's actual behaviour and that was found to be a correct explanation. In the general case, such knowledge could be deployed in an operational role with high confidence to improve individual device security and security of the network .

The XAI systems used for this work are still very rudimentary, they do not yet fulfil the complete the standard expected by Srinivasan, Bain, and Coiera, [SBC22] in which an XAI environment should to be two-way, requiring the XAI to be further capable of refuting the hypotheses of their human collaborator.

7.1 Future work

7.1.1 Volatility (Simple Good-Turing) future work

This section outlines an initial list of areas for future research. It is far from complete.

- Review volatility measurement techniques Volatility of markets, for instance, relates to real valued quantities. In this work we are interested in the volatility of nominal objects, that have a large (possibly infinite) cardinality. This would merit a literature review of existing volatility approaches and their application.
- Study independence properties of $P0$ Perform an analysis of correlation of the 6 values. Candidate techniques: ICA, PCA, simple decision trees, auto-correlation methods.
- Survey alternative volatility measures Replicate the analyses above using alternative measures.
- Analyse Destination IP addresses `id.resp_h` – Response Hosts.

- Analyse IP Originating-Responding hosts pairs `id.orig_h -- id.resp_h` – More like character bi-grams.
- Identify Low P0 internal devices `Low P0 internal devices` – These might be simple devices (e.g. IoT-like)
- Grammatical Induction `Induce behaviour models for low P0 devices` – Reverse engineer network visible behaviours for IoT devices. See section 4.
- Behavioural Complexity `Develop a scale to describe network visible behavioural complexity` – Expect that IoT devices are low on the scale, whilst traditional user workstations/laptops are high on the scale. Are there similarities with the *phi* model of consciousness[Wik22]?

7.1.2 Symbolic sequence mining (SEQUITUR) future work

What follows are potential areas of potential future work grouped into three broad topic areas: SEQUITUR usage, Visualisation, and other.

The following list of the items relate to how the system might be expanded, particularly in its use of SEQUITUR. Many items relate to ways of *auto-flattening* a sequence that is in a temporal-relational form to a temporal-propositional format[1].

- SEQUITUR k – vary the count of repeating patterns for rule formation. Experience detailed in section 4.7.1 where $k = 2$ saw that the production of rules 72 and 199 might be spurious. These rules would not have been produced if k was greater than two, whilst we expect a rule (with the same expansion as) 18[2] to be produced. This may also remove the spurious patterns relating to DNS look-ups for essentially what is the same endpoint.
- Token formation prior to SEQUITUR analysis – devise a more principled selection of fields from a temporal-relational database.
- SEQUITUR input formats – consider using plain text with end of line not allowing rule formation. Work to date has only joined text fields from the single master table (`conn`), and has not extracted fields from detailed service tables (e.g. `dns`).

[1] This is the common Data Mining data pre-processing challenge of querying a relational database to produce a single rectangular data source, to then feed into a machine learning algorithm.

[2] Each run of SEQUITUR with different parameter values may extract the same sub-sequence between runs, but label the rules differently with different rule numbers.

- `SEQUITUR` push entire event log table (as a *CSV* file) through and only use the frequently built strings to build subsequent tokens. This would take a naïve approach and treat the data as unstructured text to extract patterns to then direct token formation for a second phase of `SEQUITUR` analysis.
- `SEQUITUR` – interleave rows from `conn` to the linked service table type record (e.g. `dns`) and analyse them with `SEQUITUR`. This relates to the work done in [RHM16].

This section catalogues potential future work relating to the visualisation of the event data.

- Improved temporal clustering. The currently implemented method is a simple one-dimensional clustering with a single fixed parameter ϵ, whilst many more sophisticated techniques are available (e.g. [SNJC13])
- Build a control panel for parameters and their values. The parameters that might be initially varied are the `SEQUITUR` k and the temporal clustering ϵ.
- Add ability to overlay square brackets to selected rule numbers to the visualisation (e.g. `[ ]`). Figure 4.3 has had highlights manually added for this document. A new feature would be something that provides the same outcome, but controllable in the visualisation tool.
- GrammarViz [Gra22] is a visualisation tool that works with sequence data and utilises `SEQUITUR` analyses. Explore whether it can be used for the NetMon data.

This section catalogues other future work

- Sonification – In [Axo18] sounds are mapped to network traffic data for the purpose of network-security monitoring. The abstraction of data used in [Axo18] is low level. A higher level of abstraction could be used for sonification by using analogues of the tokens and the token map visualisation. For example each rule colour (i.e. number) and each token colour (token ID) can be mapped to particular tones and perhaps particular instrument timbres.
- Expand the data reprocessing capability by incorporating the full temporal-relational data model. The present system performs a simple selection of attributes from one table and concatenates them to form tokens. Whilst

preserving order of events, this strips away absolute timestamps and the deeper relational structures and data.

- Experiment with different data – The proof of concept described in this document has used data from on particular day from a single device. `SEQUITUR` [NMW97] claims to be particularly effective when working with longer sequences. Future work should expand the data explored with the tool.

The topic groups and topics listed in the sections above are not exhaustive.

7.1.3 XAI learning environment (`ACUITY`) future work

The work reported is a proof-of-principle empirical study. There are many ways in which the research can be expanded. Some of these are considered below.

1. **Improved tooling:** The visualisations of the `SEQUITUR` outputs from sequence mining the network event data could be integrated into the `ACUITY` workflows to enhance the ability to explain most, if not all, sequence patterns produced by a single device. Improved user interfaces of the type being explored here [DR23] will be beneficial.

2. **Expand data sources:** This work has focussed as much as possible on a single table from the event meta-data capturing system (`Zeek`). Future work may study more tables, and consider more sources of data (including emerging academic taxonomies e.g. [SSM+18]).

3. **Single IoT device complete reverse engineering:** By repeatedly inducing rules for the entire set of `SEQUITUR` provided sub-sequences will make it possible to categorise the entire observed behaviours of a single IoT device. Expanded tooling as suggested by the item above should make this easier to achieve.

4. **Explain behaviours of unknown devices:** Work to date has focused on explaining the network event behaviours of an IoT-like device where the source code of the device is available. Future work should study devices for which there is no access to its components.

5. **Export explained behaviours:** Having produced behaviour explanations, can these be codified and exported in a standard form? Existing

standards for IoT behaviour specification include *D3*[ea21a] and *MUD* [GM20]

6. **Run-time deployment of explanations:** Can the exported explained behaviours provide cyber security defence for the IoT device(s)? Is it possible to build and deploy the rules in a real-time Intrusion Detection/Protection System, hosted perhaps in a custom IoT security gateway (e.g. see: [ea21b])?

7. **Restricted power of reasoning:** Using standard logic programming deductive techniques is limited to simple deduction when there are one single consistent logical *model*. Coping with inconsistencies and missing data are common in human reasoning. Using developments in *Answer Set Programming* ([ACS+18]) and *s(CASP)* ([ACS+18]) for a more comprehensive reasoning basis will keep the formalism human comprehensible, and extend the power.

The above list is not exhaustive, but provides some future research and development directions.

Chapter 8

Acknowledgements

This work is supported by the Innovate-UK *105592: CyberStone: Collaborative Secure IOT Gateway (ManySecured)* grant.

Bibliography

[ACS+18] Joaquín Arias, Manuel Carro, Elmer Salazar, Kyle Marple, and Gopal Gupta. Constraint answer set programming without grounding. *Theory and Practice of Logic Programming*, 18(3-4):337–354, 2018.

[Axo18] Louise Axon. *Sonification for Network-Security Monitoring*. PhD thesis, Department of Computer Science, 2018.

[BC06] Lindsay Barrett and Matthew Connell. Jevons and the logic 'piano'. 1, 2005–2006.

[Boo02] Orange Book. *Trusted Computer System Evaluation Criteria: United States Government Department of Defense Standard, DoDD 5200.28-STD*. United States Government Department of Defense, 2002.

[BS95] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, 1995.

[BUG96] Francis Bacon, Peter Urbach, and John Gibson. Novum organum. *British Journal for the Philosophy of Science*, 47(1):125–128, 1996.

[CCK+00] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, and Rüdiger Wirth. *The CRISP-DM User Guide Cross Industry Process for Data Mining*. 2000. Accessed on 2022-10-19.

[DNPV22] Anastasia Dimou, Sebastian Neumaier, Tassilo Pellegrini, and Sahar Vahdati, editors. *UX Design & Knowledge Graphs - The Perfect Match - SEMANTiCS 2022 - Proceedings of the 18th International Conference on Semantic Systems, 13-15 September 2022, Vienna, Austria*, volume 55 of *Studies on the Semantic Web*. IOS Press, 2022.

[DR23] Oliver Deane and Oliver Ray. Interactive model refinement in relational domains with inductive logic programming. In *Companion Proceedings of the 28th International Conference on Intelligent User Interfaces*, pages 127–129, 2023.

[ea21a] Nicholas Allott et al. Distributed device descriptors (d3). `https://specs.manysecured.net/d3/`, 2021. Accessed 2022-10-19.

[ea21b] Nicholas Allott et al. Manysecured technical documents – smart secure collaborative gateways. `https://specs.manysecured.net/`, 2021. Accessed 2022-10-19.

[FHW16] Eibe Frank, Mark A. Hall, and Ian H. Witten. *Data Mining: Practical Machine Learning Tools and Techniques (Fourth Edition)*. Morgan Kaufmann, 2016.

[Fla12] Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, USA, 2012.

[FMPS98] Paul Finn, Stephen Muggleton, David Page, and Ashwin Srinivasan. Pharmacophore discovery using the inductive logic programming system progol. *Mach. Learn.*, 30(2–3):241–270, feb 1998.

[Gil96] Donald Gillies. *Artificial Intelligence and Scientific Method*. Oxford University Press, Oxford and New York, 1996.

[GM20] Parisa Grayeli and Blaine Mulugeta. Mitigating network-based attacks using mud – improving security of small-business and home iot devices. In *RSA Conference*, pages 41–50, 2020. Accessed 2022-10-19.

[Goo53] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40:237–264, 1953.

[Gra22] GrammarViz. Grammarviz. `https://github.com/GrammarViz2/grammarviz2_src`, 2022. Accessed on FIX ME.

[GS95] William Gale and Geoffrey Sampson. Good–turing frequency estimation without tears. *The Journal of Quantitative Linguistics*, 2:217–37, 1995.

[GVWT21] David Gunning, Eric Vorm, Jennifer Yunyan Wang, and Matt Turek. Darpa's explainable ai (xai) program: A retrospective. *Applied AI Letters*, 2(4):e61, 2021.

[(IS22] (ISC)2. Cybersecurity workforce study. `https://media.isc2.org/-/media/Project/ISC2/Main/Media/documents/research/ISC2-Cybersecurity-Workforce-Study-2022.pdf?rev=1bb9812a77c74e7c9042c3939678c196`, 2022.

[Jev12] W.S. Jevons. *The Principles of Science: A Treatise on Logic and Scientific Method*. Ebsco PsychBooks. Cosimo, Incorporated, 2012.

[KM01] Marja-Riitta Koivunen and Eric Miller. W3C semantic web activity. In *Proceedings of the Semantic Web Kick-off Seminar in Finland*, November 2001.
`http://www.w3.org/2001/12/semweb-fin/w3csw`.

[KMSS96] RD King, SH Muggleton, A Srinivasan, and MJ Sternberg. Structure-activity relationships derived by machine learning: the use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences of the United States of America*, 93(1):438—442, January 1996.

[Ko00] Calvin Ko. Logic induction of valid behavior specifications for intrusion detection. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 142–153. IEEE Computer Society, 2000.

[Kow11] Robert Kowalski. *Computational Logic and Human Thinking: How to Be Artificially Intelligent*. Cambridge University Press, 2011.

[Lif22] Vladimir Lifschitz. *What Is Answer Set Programming?* University of Texas, 2022. Accessed June 2023.

[Llo84] J.W. Lloyd. *Foundations of Logic Programming*. Symbolic computation. Springer-Verlag, 1984.

[LV08] Ming Li and Paul M.B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, New York, NY, 2008.

[MAM23] Steve Moyle, Nicholas Allott, and John Manslow. Modelling cyber defenses using s(casp). In Joaquín Arias, Sotiris Batsakis, Wolfgang

Faber, Gopal Gupta, Francesco Pacenza, Emmanuel Papadakis, Livio Robaldo, Kilian Rückschloß, Elmer Salazar, Zeynep Gozen Saribatur, Ilias Tachmazidis, Felix Weitkämper, and Adam Z. Wyner, editors, *Proceedings of the International Conference on Logic Programming 2023 Workshops co-located with the 39th International Conference on Logic Programming (ICLP 2023), London, United Kingdom, July 9th and 10th, 2023*, volume 3437 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[McC59] John McCarthy. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London, 1959. Her Majesty's Stationary Office.

[Mee15] Haroon Meer. What got us here wont get us there. In *Black-Hat Europe*, 2015.

[MFM+22] K. Megas, M. Fagan, J. Marron, P. Watrobski, , and B. Cuthill. Profile of the iot core baseline for consumer iot products. Internal report, National Institute of Standards and Technology, Gaithersburg, MD, 2022. NIST Interagency/Internal Report (NISTIR).

[MH03] Steve Moyle and John Heasman. Machine learning to detect intrusion strategies. In Vasile Palade, Robert J. Howlett, and Lakhmi Jain, editors, *Knowledge-Based Intelligent Information and Engineering Systems*, pages 371–378, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Mic88] Donald Michie. Machine learning in the next five years. In D. Sleeman and J. Richmond, editors, *Third European Workshop Session on Learning (EWSL '88)*, pages 107 – 122. Pitman, 1988.

[Moy22] Steve Moyle. *Simple Good-Turing Frequency Analysis applied to Network Monitoring for volatility indication – Internal Project Report*. 105592: CyberStone: Collaborative Secure IOT Gateway (ManySecured), February 2022.

[MR94] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19, 20:629 – 679, 1994.

[MSZ+18] Stephen H. Muggleton, Ute Schmid, Christina Zeller, Alireza Tamaddoni-Nezhad, and Tarek Besold. Ultra-strong machine learn-

ing: Comprehensibility of programs learned with ilp. *Mach. Learn.*, 107(7):1119–1140, jul 2018.

[Mug95] S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 3 and 4(13):245 – 286, 1995.

[Mug14] Stephen Muggleton. Alan turing and the development of artificial intelligence. *AI Commun.*, 27(1):3–10, jan 2014.

[NM21] C.G. Neville-Manning. Sequitur. `http://www.sequitur.info/`, November 2021.

[NMW97] C.G. Neville-Manning and I.H. Witten. Linear-time, incremental hierarchy inference for compression. In *Data Compression Conference (DCC '97)*, page 3–11, 1997.

[NTA+18] P.H. Nguyen, C. Turkay, G. Andrienko, N. Andrienko, O. Thonnard, and J. Zouaoui. Understanding user behaviour through action sequences: from the usual to the unusual. *IEEE transactions on visualization and computer graphics*, 2018.

[Ope23] OpenVAS. Openvas - open vulnerability assessment scanner. `https://openvas.org/`, June 2023.

[Org00] International Standards Organisation. *The Common Criteria for Information Technology Security Evaluation. International Standard (ISO/IEC 15408) for computer security certification.* ISO/IEC, 2000.

[Pei96] C.S. Peirce. *Elements of Logic.* Collected papers of Charles Sanders Peirce / Peirce, C. 1996.

[Plo71] Gordon D. Plotkin. *Automatic Methods of Inductive Inference.* PhD thesis, Department of Computer Science, 1971.

[Pop35] Karl R. Popper. *The Logic of Scientific Discovery.* Routledge, London, England, 1935.

[Pre22] Presto. Presto is an open source sql query engine that's fast, reliable, and efficient at scale. `https://github.com/GrammarViz2/grammarviz2_src`, October 2022.

[RHM16] O. Ray, S. Hicks, and S. Moyle. Using ilp to analyse ransomware attacks. In *26th Int. Conf. on Inductive Logic Programming (ILP), CEUR 1865*, pages 54–59, 2016.

[RM21] O. Ray and S. Moyle. Towards expert-guided elucidation of cyber attacks through interactive inductive logic programming. In *13th International Conference on Knowledge and Systems Engineering (KSE)*. IEEE Press, 2021.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[Sam17] Geoffrey Sampson. Good–turing frequency estimation. Report, October 2017. Accessed on 2021-08-25.

[SBC22] Ashwin Srinivasan, Michael Bain, and Enrico Coiera. One-way explainability isn't the message. *arXiv*, May 2022.

[Sch20] Rolf Schwitter. Lossless semantic round-tripping in peng asp. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 5291–5293. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Demos.

[Scr10] Roger Scruton. *The uses of pessimism : and the danger of false hope*. Atlantic Books, London, 2010.

[SDFP23] Galileo Sartor, Jacinto Dávila, Alessia Fidelangeli, and Giuseppe Pisano. (re)integration of logical english and s(casp). In Joaquín Arias, Sotiris Batsakis, Wolfgang Faber, Gopal Gupta, Francesco Pacenza, Emmanuel Papadakis, Livio Robaldo, Kilian Rückschloß, Elmer Salazar, Zeynep Gozen Saribatur, Ilias Tachmazidis, Felix Weitkämper, and Adam Z. Wyner, editors, *Proceedings of the International Conference on Logic Programming 2023 Workshops colocated with the 39th International Conference on Logic Programming (ICLP 2023), London, United Kingdom, July 9th and 10th, 2023*, volume 3437 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2023.

[SNJC13] Jeffrey D. Scargle, Jay P. Norris, Brad Jackson, and James Chiang. Studies in astronomical time series analysis. vi. bayesian block representations. *The Astrophysical Journal*, 764(2):26, 2013.

[Sri17] Ashwin Srinivasan. The aleph manual. `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/`, 2017.

[SSM+18] Leslie F. Sikos, Markus Stumptner, Wolfgang Mayer, Catherine Howard, Shaun Voigt, and Dean Philp. Representing network knowledge using provenance-aware formalisms for cyber-situational awareness. *Procedia Computer Science*, 136:29–38, 2018.

[Tur37] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society, 2*, 42(1):230 – 65, 1937.

[Tur50] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[tyr21] tyrex. 1d number array clustering. stackexchange question answered jul 6 '21 at 13:02 by *tyrex*. `https://stackoverflow.com/questions/11513484/1d-number-array-clustering`, Jul 2021. Accessed on FIX ME.

[WA02] Jan Wielemaker and Anjo Anjewierden. An architecture for making object-oriented systems available from prolog. In *Workshop on Logic Programming Environments*, 2002.

[Wik22] WikiPedia. Integrated information theory. `https://en.wikipedia.org/wiki/Integrated_information_theory`, 2022. Accessed on August 2021.

[Wol23] S. Wolfram. *What Is ChatGPT Doing ... and Why Does It Work?* Wolfram Media, Incorporated, 2023.

[WSTL10] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. Swi-prolog. *arXiv*, May 2010. arXiv.1011.5332[cs.PL].

[Zee21] Zeek. The zeek network security monitor. `https://zeek.org/`, June 2021.

[Ziz14] Slavoj Zizek. *Event: a philosophical journey through a concept.* Melville House, Brooklyn, NY, 2014.

Appendices

.1 Data Understanding

Much of the analysis focuses on the connection log. An example record and details of the meta data fields available in the `conn` log are shown in Table 1 (below).

.2 Appendices relating to SGT Volatility

.2.1 Analyses and observations

Column Name	Data type	Example	Selected descriptions
ts	timestamp	2020-12-31 23:59:22	Timestamp of connection
uid	string	CSuVAz2v3MtaiPkaja	Unique ID (key)
id.orig_h	string	10.0.100.55	Source IP address
id.orig_p	bigint	51475	Source IP port
id.resp_h	string	192.29.32.24	Destination IP address
id.resp_p	bigint	80	Destination IP port
proto	string	tcp	Protocol
service	string	http	Service type
duration	string	0 days 00:01:00.052789000	
orig_bytes	bigint	282	Originator message size
resp_bytes	bigint	510	Response message size
conn_state	string	SF	
local_orig	boolean	TRUE	
local_resp	boolean	FALSE	
missed_bytes	bigint	0	
history	string	ShADadFf	
orig_pkts	bigint	6	
orig_ip_bytes	bigint	534	
resp_pkts	bigint	4	
resp_ip_bytes	bigint	682	
tunnel_parents	string	(empty)	
year	smallint	2020	
month	smallint	12	
day	smallint	31	

Table 1: Column names, data types, and examples of fields from the Zeek `conn` metadata log. NB (1) Some values have been altered to ensure confidentiality. For a complete description of fields see: `https://docs.zeek.org/en/master/logs/conn.html` . NB (2) The `year`, `month`, and `day` fields are additional helper fields provided independently from Zeek to improve database storage and indexing.

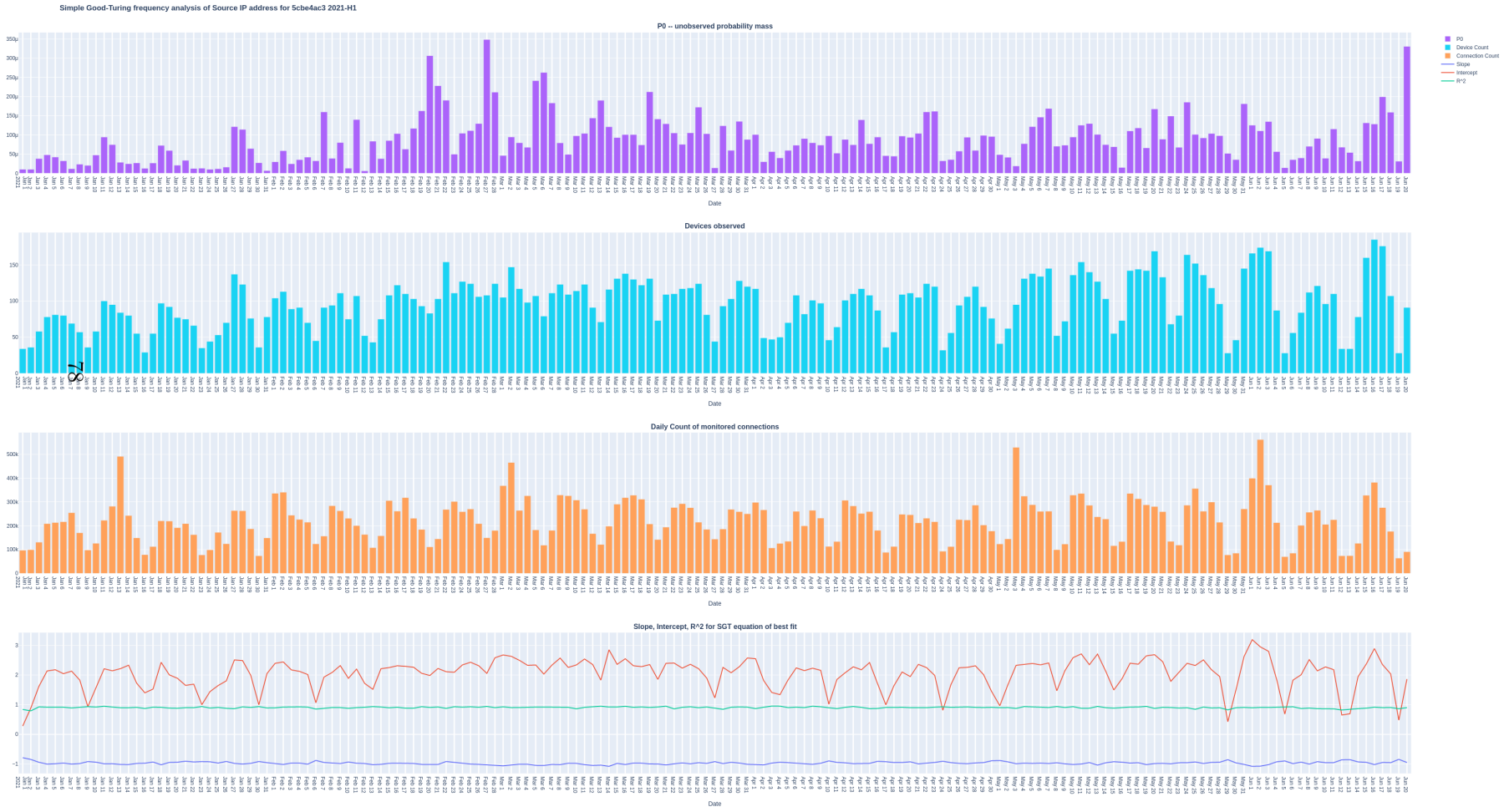

Figure 1: Simple Good-Turing frequency analysis of Originating Host IP addresses for 5cbe4ac3 2021-H1 – by day. Top panel – purple bars – $P0$ SGT probability mass of unseen addresses (a proxy for volatility). Panel 2 – cyan bars – number of connected devices. Panel 3 – orange bars – number of monitored connections. Panel 4 – model fit parameters – intercept (red), slope (blue), r^2 (green).

Rank	Total Daily Connections	Year	Month	Day	Weekday
1	561,742	2021	6	2	Wednesday
2	529,238	2021	5	3	Monday
3	491,148	2021	1	13	Wednesday
4	465,476	2021	3	2	Tuesday
5	399,347	2021	6	1	Tuesday
:	:	:	:	:	:
:	:	:	:	:	:
2239	74,096	2021	6	13	Sunday
240	73,670	2021	6	12	Saturday
241	73,384	2021	1	30	Saturday
242	69,937	2021	6	5	Saturday
243	63,710	2021	6	19	Saturday

Table 2: The 5 most frequent and the five least frequent daily number of connections observed. Note the difference in weekdays and weekends.

The daily data is plotted in Figure 1 – Simple Good-Turing frequency analysis of the Originating Hosts (Source IP addresses) observed by the network monitoring system with serial number `5cbe4ac3` during the period 2021-H1.

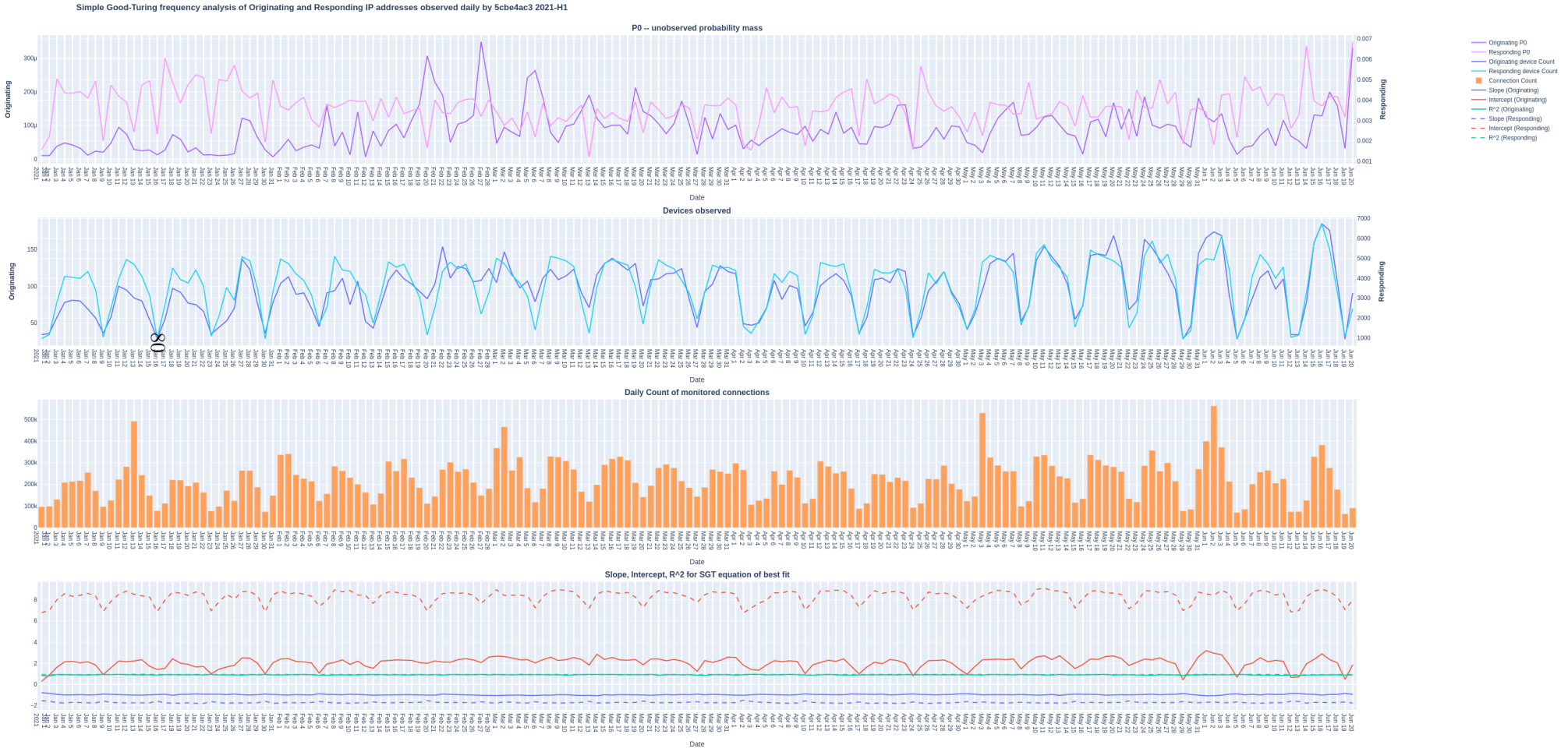

Figure 2: Simple Good-Turing frequency analysis of individual Originating IP addresses and individual Responding IP addresses for 5cbe4ac3 2021-H1 – by day. Top panel – $P0$ SGT probability mass of unseen addresses (a proxy for volatility) – Originating IP addresses (purple); Responding IP addresses (pink). Panel 2 – Daily count of IP addresses – Originating IP addresses (dark blue); Responding IP addresses (light blue). Panel 3 – orange bars – number of monitored connections. Panel 4 – model fit parameters – intercept (red), slope (blue), r^2 (green) – Model for Originating IP addresses (solid lines); Model for Responding IP addresses (dashed lines).

Figure 3: Simple Good-Turing frequency analysis summary of individual internal IP addresses observed daily by 5cbe4ac3 2021-H1 – rank ordered by mean P0 value for the period. The device with lowest average volatility is the left-most data-point; and the device with the highest average volatility is the right-most data-point. Top Panel – blue bars – the period average $P0_{conn}$ for 2021-H1 – each bar represents a single `id.orig_h` (e.g. an IP address); Panel 2 – red bars – the period standard deviation of $P0_{conn}$; Panel 3 – green bars – the number of days the `id.orig_h` is active in the period (maximum = 171 days).

id.orig_h	Days Active	P0 sum	P0 mean	P0 std
10.0.0.173	171	0.076522	0.000447	0.000039
fe80::7c16:c0df:389d:51a6	171	0.091728	0.000536	0.000273
10.0.0.145	171	0.139165	0.000814	0.001308
10.0.0.137	171	0.167736	0.000981	0.000598
fe80::1ae8:29ff:feb6:5737	171	0.210277	0.001230	0.001937
205.196.6.75	1	0.500000	0.500000	0.000000
74.122.190.78	1	0.500000	0.500000	0.000000
169.254.209.247	4	2.200000	0.550000	0.057735
fe80::4f6:f8b2:c79:975e	2	1.100000	0.550000	0.070711
169.254.179.210	1	0.600000	0.600000	0.000000

Table 3: The five lowest and the five highest ranked devices by their period average $P0_{conn}$ 2021-H1.

id.orig_h	Days Active	P0 sum	P0 mean	P0 std	MAC Address	Device Manufacturer	Type of Device Hypothesis
10.0.0.137	171	0.167736	0.000981	0.000598	001daabfb300	DrayTek Corp.	Router
10.0.0.145	171	0.139165	0.000814	0.001308	b827eb5b2331	Raspberry Pi Foundation	brAIn-box
10.0.0.173	171	0.076522	0.000447	0.000039	00d04b961b40	LA CIE GROUP S.A.	NAS
fe80::fa0d:60ff:fe4a:c270	131	6.429776	0.049082	0.026372	F80D604AC270	CANON INC.	Printer
fe80::92e7:c4ff:fe68:144a	116	18.407727	0.158687	0.115621	90E7C468144A	HTC Corporation	Possible Smart Phone

Table 4: Identification of particular devices by their network hardware address, and their associated volatility statistics for the period 2021-H1.

```
diagnostics.20210620_0500.log.gz Diagnostics report for 5cbe4ac3-6178-
     508c-b9e2-dc44bb8025aa
5cbe4ac3-6178-508c-b9e2-dc44bb8025aa
 :
 05:02:02 up 24 days,  5:17,  0 users,  load average: 1.63, 1.52, 1.93
Linux 5cbe4ac3-6178-508c-b9e2-dc44bb8025aa 4.14.79-v7+ #1159 SMP Sun
    Nov 4 17:50:20 GMT 2018 armv7l GNU/Linux
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.0.145  netmask 255.255.255.0  broadcast 10.0.0.255
        inet6 fe80::23c0:cb3f:a6ba:9af2  prefixlen 64  scopeid 0x20<link>
        ether b8:27:eb:5b:23:31  txqueuelen 1000  (Ethernet)
        RX packets 1645057923  bytes 1458532012969 (1.3 TiB)
        RX errors 0  dropped 902  overruns 0  frame 0
        TX packets 65895898  bytes 84445104451 (78.6 GiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Listing 1: The relevant contents of the diagnostics file for the *brAIn-box*.

Looking up the information for device `10.0.0.8` with MAC `74:83:c2:4c:00:f9` returns the following information:

```
 MAC address to search for or name of manufacturer:

Number of database entries: 44 974
The MAC address "74:83:c2:4c:00:f9" has been assigned by the IEEE to:
MAC-Segment:  74:83:C2:00:00:00 - 74:83:C2:FF:FF:FF (MA-L)
Vendor:   Ubiquiti Networks Inc.
Address:  2580 Orchard Pkwy
San Jose CA 95131
US
```

Listing 2: Manufacturer MAC address information for 10.0.0.8 having a high observed volatility of all the devices.

.2.2 Ignoring Timestamps

Do absolute timestamps of logged events (messages) matter? Does the inter-arrival duration between logged events (messages) matter?

`SEQUITUR` has no notion of timestamps, only the ordering of the tokens in the sequence, from which to induce grammars and extract patterns.

Some hope that there is useful information in the sequences even without the timestamps can be found in the research from Warwick[NTA+18], where they were mining sequences of user behaviour for patterns of good and bad behaviour. Their approach was to disregard timestamps and utilize the token arrival ordering.

Alternative sequence extraction might be compared to the technique produced by Katsumi Inoue LFIT for learning state transitions.

.2.3 Weekday variance of P0

In all three cases where P0 is measured on a daily basis: **originating hosts** (Table 5), **responding hosts** (Table 6), and **originating-responding host pairs** (Table 7), there is a predominance in lower P0 values occurring on week-end days.

Note that for all of the tables that follow the Weekdays are only approximate. The original data was recorded with 'UTC' as the timestamp (i.e. as the clock ticks at Greenwich Mean Time). The actual network being monitored was in 'GMT+11'. This means that a record timestamped with '08:00:00' (potentially before the start of a business day) was recorded at '19:00:00' local time (potentially after the close of a business day).

.2.4 Ideas

- Operational definition of IoT devices – As the term *Master-Slave* seems no longer in vogue, perhaps a device for which an App is used to interact with the device. The App is a proxy term for *Master*, and the IoT device is a proxy term for *Slave*. In Tango, these roles would be (in English) Leader and Follower (or in Spanish - marcar and responder). Alternatively, from the philosopher Roger Scruton, we could use *Master* and *Subordinate* [Scr10]. Or maybe – *Master* and *Servant*.

.3 Appendices relating to `SEQUITUR`

.3.1 Simple `SEQUITUR` Example

The following example is adapted from `http://www.sequitur.info/`. The aim of this example is to show how the compositional hierarchies are constructed and the representation of the grammar expansion-rules.

Rank	P0	Weekday	Year	Month	Day
1	0.000349	Saturday	2021	2	27
2	0.000331	Sunday	2021	6	20
3	0.000306	Saturday	2021	2	20
4	0.000263	Saturday	2021	3	6
5	0.000241	Friday	2021	3	5
6	0.000228	Sunday	2021	2	21
7	0.000212	Friday	2021	3	19
8	0.000211	Sunday	2021	2	28
9	0.000199	Thursday	2021	6	17
10	0.00019	Monday	2021	2	22
11	0.00019	Saturday	2021	3	13
12	0.000185	Monday	2021	5	24
13	0.000183	Sunday	2021	3	7
14	0.000181	Monday	2021	5	31
15	0.000172	Thursday	2021	3	25
16	0.000169	Friday	2021	5	7
17	0.000168	Thursday	2021	5	20
18	0.000163	Friday	2021	2	19
19	0.000162	Friday	2021	4	23
20	0.00016	Thursday	2021	4	22
:	:	:	:	:	:
:	:	:	:	:	:
224	0.000025	Wednesday	2021	2	3
225	0.000024	Friday	2021	1	8
226	0.000021	Wednesday	2021	1	20
227	0.00002	Saturday	2021	1	9
228	0.000019	Monday	2021	5	3
229	0.000016	Tuesday	2021	1	26
230	0.000015	Sunday	2021	5	16
231	0.000014	Saturday	2021	6	5
232	0.000014	Saturday	2021	3	27
233	0.000013	Wednesday	2021	2	10
234	0.000013	Saturday	2021	1	23
235	0.000013	Saturday	2021	1	16
236	0.000012	Friday	2021	1	22
237	0.000012	Thursday	2021	1	7
238	0.000012	Monday	2021	1	25
239	0.00001	Friday	2021	1	1
240	0.00001	Sunday	2021	1	24
241	0.00001	Saturday	2021	1	2
242	0.000007	Sunday	2021	1	31
243	0.000006	Friday	2021	2	12

Table 5: The 20 most highest daily P0 values and the 20 lowest daily P0 values for the **originating hosts**.

Rank	P0	Weekday	Year	Month	Day
1	0.006856	Sunday	2021	6	20
2	0.006653	Monday	2021	6	14
3	0.006062	Sunday	2021	1	17
4	0.005703	Tuesday	2021	1	26
5	0.005668	Sunday	2021	4	25
6	0.005238	Thursday	2021	1	21
7	0.00515	Sunday	2021	6	6
8	0.005091	Friday	2021	1	22
9	0.005039	Sunday	2021	4	18
10	0.005035	Sunday	2021	1	3
11	0.005008	Sunday	2021	1	24
12	0.005003	Wednesday	2021	5	26
13	0.004988	Sunday	2021	1	31
14	0.004934	Friday	2021	1	15
15	0.004931	Friday	2021	1	8
16	0.004928	Monday	2021	1	25
17	0.004878	Sunday	2021	5	9
18	0.004861	Monday	2021	1	18
19	0.004748	Wednesday	2021	1	20
20	0.004747	Thursday	2021	1	14
:	:	:	:	:	:
:	:	:	:	:	:
224	0.002348	Saturday	2021	1	23
225	0.002327	Saturday	2021	1	16
226	0.002252	Monday	2021	5	3
227	0.002208	Saturday	2021	4	17
228	0.002195	Saturday	2021	3	6
229	0.002194	Saturday	2021	1	2
230	0.002159	Saturday	2021	6	5
231	0.002074	Saturday	2021	3	27
232	0.002057	Saturday	2021	5	22
233	0.002002	Saturday	2021	1	9
234	0.001921	Saturday	2021	1	30
235	0.001847	Saturday	2021	5	29
236	0.001828	Friday	2021	4	2
237	0.001803	Wednesday	2021	6	2
238	0.001755	Saturday	2021	4	10
239	0.001658	Saturday	2021	2	20
240	0.001578	Friday	2021	1	1
241	0.001565	Saturday	2021	4	24
242	0.001559	Saturday	2021	4	3
243	0.001215	Saturday	2021	3	13

Table 6: The 20 most highest daily P0 values and the 20 lowest daily P0 values for the **responding hosts**.

Rank	P0	Weekday	Year	Month	Day
1	0.02148	Monday	2021	6	14
2	0.021393	Sunday	2021	6	20
3	0.01918	Sunday	2021	1	31
4	0.018944	Tuesday	2021	1	26
5	0.017068	Wednesday	2021	6	9
6	0.016999	Thursday	2021	3	25
7	0.016967	Sunday	2021	1	17
8	0.016916	Wednesday	2021	1	27
9	0.016895	Friday	2021	1	22
10	0.01689	Thursday	2021	6	17
11	0.016863	Sunday	2021	2	14
12	0.016769	Thursday	2021	5	6
13	0.016735	Thursday	2021	6	10
14	0.016676	Monday	2021	1	18
15	0.016676	Thursday	2021	5	13
16	0.016629	Wednesday	2021	6	16
17	0.016548	Thursday	2021	1	28
18	0.016508	Thursday	2021	1	21
19	0.016502	Wednesday	2021	5	26
20	0.016453	Tuesday	2021	2	16
:	:	:	:	:	:
:	:	:	:	:	:
224	0.007899	Saturday	2021	1	23
225	0.007828	Sunday	2021	6	13
226	0.007613	Saturday	2021	6	19
227	0.0076	Monday	2021	5	3
228	0.007566	Saturday	2021	2	13
229	0.007524	Saturday	2021	1	2
230	0.00744	Wednesday	2021	1	13
231	0.007426	Saturday	2021	3	27
232	0.007357	Saturday	2021	6	12
233	0.00732	Saturday	2021	4	10
234	0.007154	Saturday	2021	1	30
235	0.006817	Friday	2021	1	1
236	0.006691	Saturday	2021	4	17
237	0.006447	Saturday	2021	5	1
238	0.00638	Saturday	2021	5	22
239	0.006277	Saturday	2021	6	5
240	0.005915	Saturday	2021	5	29
241	0.005719	Saturday	2021	3	13
242	0.005061	Saturday	2021	4	24
243	0.004794	Friday	2021	4	2

Table 7: The 20 most highest daily P0 values and the 20 lowest daily P0 values for the **originating-responding host pairs**.

Consider the input string: `abcabdabcabd`

We can execute the `SEQUITUR` run-time within a unix-like terminal as follows:

```
$ echo "abcabdabcabd" | ~/gits/sequitur/c++/sequitur -prk2tr
```

Listing 3: Running `SEQUITUR`

```
0 -> 1 1 \n
1 -> 2 c 2 d      abcabd
2 -> a b      ab
```

Listing 4: `SEQUITUR` output

The output of the "compression" of the string is shown in listing 4. The numbers preceding the `->` symbol are expansion *rule numbers*. *Rule 0* is the distinguished **start rule S0** and represents the compressed input string.

Each expansion rule specifies how to replace the elements of the compressed input string in a way that exactly reconstructs the original input string.

Rule `0` – the start rule S0 – instructs us to expand by `->` replacing `1` with the expansion produced by rule 1 followed by replacing `1` with the expansion produced by rule 1 (a second time) followed by an end-of-line character `\n`.

Rule `1` instructs us to expand by `->` replacing `2` with the expansion produced by rule 2 followed by the character `c` followed by the expansion produced by rule 2 (a second time) followed by the character `d`. Notice that rule 1 also has the full sub-string (`abcabd`) that the rule fully expands to as a helpful comment.

Rule `2` instructs us to expand by `->` replacing it with the character `a` followed by the character `b`. Notice that rule 2 also has the full sub-string (`ab`) that the rule fully expands to as a helpful comment.

The expansion process is summarised in table 8 (below)

Rule Number	Rule Body	Expansion
0	rule(1) rule(1)	Original input
1	rule(2) `c` rule(2) `d`	`abcabd`
2	`a b`	`ab`

Table 8: Expansion rule grammar from input string "abcdabcd".

The process for producing integer tokens from NetMon connection is illustrated with an example in figure 5 .

The first 10 and last 10 `token_uids` are shown in listing 5.

```
token_uid
350344446
```

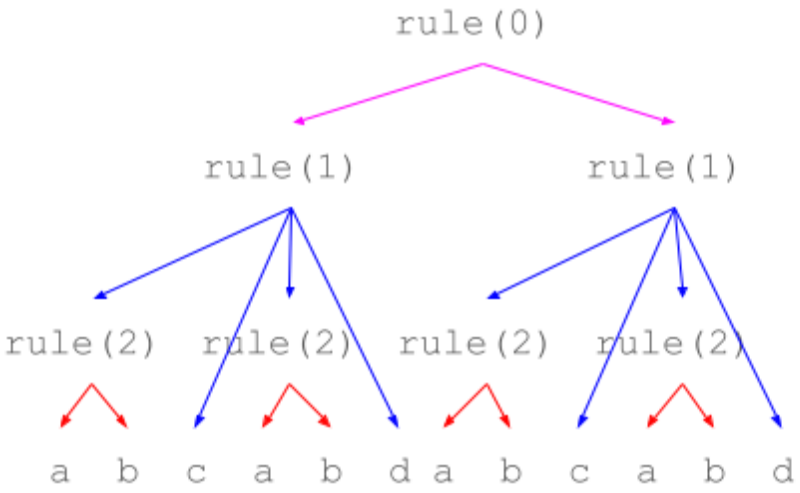

Figure 4: An illustrative example of the reconstruction of the compressed input sequence from the SEQUITUR produced grammar structures. Start at the start rule 0, recursively replace each rule with its elements and each element until only terminals remain.

```
271311686
100000001
100000001
271311686
271311686
350344446
271311686
100000001
271311686

<3593 token_uids not shown>

271311686
271311686
271311686
271311686
271311686
100000001
100000001
840284070
```

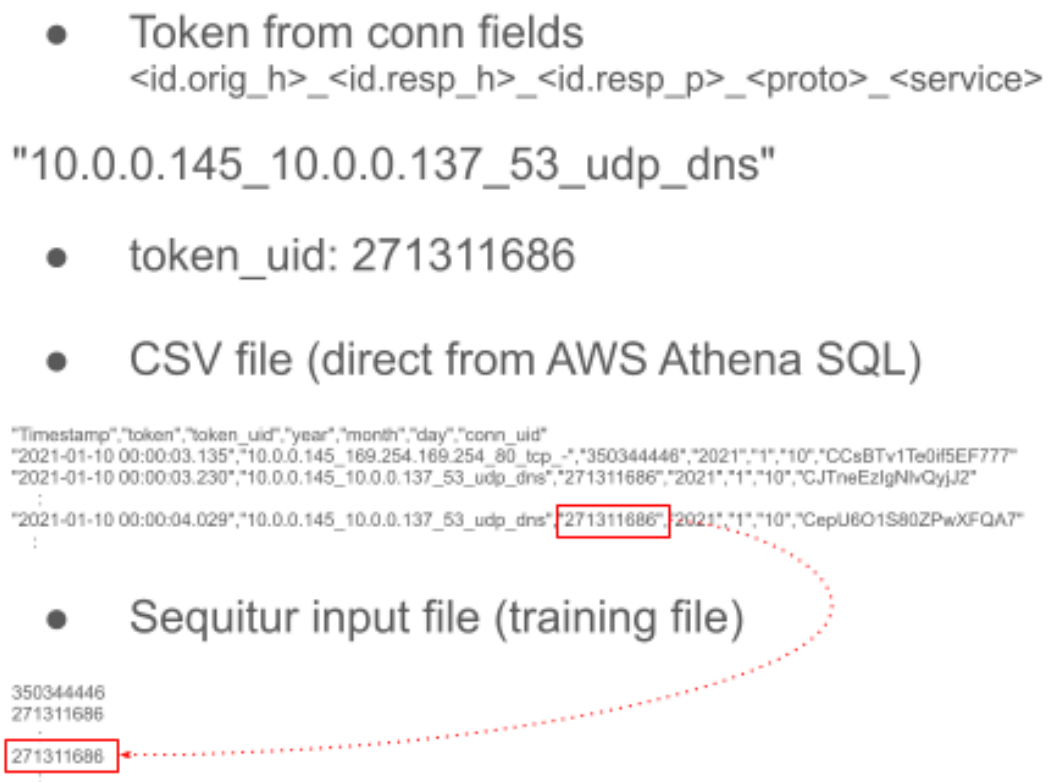

Figure 5: Example data transformations to produce `token_id`s from Zeek `conn` NetMon event data as an input file ready for extracting patterns using `SEQUITUR`.

```
271311686
271311686
```

Listing 5: Example `SEQUITUR` input file. Top 10 `token_uid`s and bottom 10 `token_uid`s.

```
$ tail -n +2 10-0-0-145_2021-01-10tx.csv \
| cut -d '"' -f 6 \
| /home/sam/gits/sequitur/c++/sequitur -pdrtTk 2 \
> 10-0-0-145_2021-01-10tx.seq
```

Listing 6: Running `SEQUITUR` on the input CSV file `10-0-0-145_2021-01-10tx.csv` to produce a sequence grammar file `10-0-0-145_2021-01-10tx.seq`.

```
$ head 10-0-0-145_2021-01-10tx.seq
0 -> 1 2 3 3 4 5 6 7 8 9 10 11 12 13 14 14 15 16 [840284070] 9 17
    18 19 20 21 21 21 22 16 23 24 25 1 8 26 26 27 28 [840284070] 29
     30 31 32 33 10 34 20 35 11 31 36 36 36 33 37 38 39 35 40 41
    [213948788] [213948788] [271311686] [680399771] [680399771] 42
    42 43 [513489017] [513489017] 35 37 43 35 2 44 44 45
    [840284070] 40 [840284070] 42 [271311686] 11 29 44 46 47 45 48
    49 50 19 51 52 52 52 53 54 55 56 49 19 57 58 58 59 60 61 59 59
    59 62 63 64 61 65 65 66 67 68 17 69 66 66 70 71 [382930009] 64
    24 72 [271311686] 72 73 73 74 75 76 24 77 78 79 75 80 81 57
    [271311686] 78 82 83 67 51 2 69 84 70 85 43 50 86 87 88 84 89
    76 [840284070] 77 10 90 80 25 91 [100000001] 20 90 92 48 2 93
    87 94 95 83 96 97 92 8 98 34 50 17 99 100 55 43 86 13 99 101
    102 103 81 104 [350344446] 69 2 105 105 96 101 43 103 8 93 2 43
     [271311686] 30 77 11 68 106 106 50 55 37 12 [628100651]
    [866509023] 107 12 [382930009] 103 107 2 34
1 -> 17 108   2 (2) [350344446] [271311686] [100000001] [100000001]
     [271311686] [271311686] [350344446] [271311686] [100000001]
2 -> [271311686] [271311686]  20  (109) [271311686] [271311686]
3 -> 109 109  2 (2) [100000001] [271311686] [100000001] [100000001]
     [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001] [100000001] [271311686]
    [100000001] [100000001] [271311686] [100000001] [100000001]
    [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001] [100000001] [271311686]
    [100000001] [100000001] [271311686] [100000001] [100000001]
    [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001] [100000001] [271311686]
    [100000001] [100000001] [271311686] [100000001]
4 -> 110 110  3 (17)  [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001]
5 -> [100000001] [271311686]  4 (162) [100000001] [271311686]
6 -> 30 111   2 (3) [847213017] [847213017] [271311686] [100000001]
     [100000001] [271311686] [100000001] [100000001] [271311686]
    [100000001]
7 -> 19 [840284070]   2 (3) [100000001] [271311686] [271311686]
    [840284070]
8 -> 112 [100000001]  15  (71)  [100000001] [271311686] [100000001]
     [100000001]
9 -> 113 7  2 (2) [100000001] [271311686] [100000001] [100000001]
    [271311686] [271311686] [271311686] [271311686] [271311686]
    [100000001] [100000001] [271311686] [271311686] [840284070]
```

Listing 7: First 10 lines of sequence grammar file `10-0-0-145_2021-01-10tx.seq`.

```
$ tail 10-0-0-145_2021-01-10tx.seq
121 -> 124 [840284070]  2 (4) [100000001] [271311686] [100000001]
    [100000001] [271311686] [271311686] [271311686] [271311686]
    [271311686] [100000001] [100000001] [840284070]
122 -> 2 108  2 (7) [271311686] [271311686] [350344446] [271311686]
     [100000001]
123 -> 50 [271311686]   11  (65)  [271311686] [100000001]
    [100000001] [271311686]
124 -> 48 68  2 (6) [100000001] [271311686] [100000001] [100000001]
     [271311686] [271311686] [271311686] [271311686] [271311686]
    [100000001] [100000001]
125 -> 120 37   5 (86)  [100000001] [100000001] [271311686]
    [100000001] [100000001] [271311686] [100000001] [100000001]
    [271311686]
126 -> 96 96  3 (8) [100000001] [100000001] [271311686] [100000001]
     [100000001] [271311686] [100000001] [100000001] [271311686]
    [100000001] [100000001] [271311686] [100000001] [100000001]
    [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001] [100000001] [271311686]
    [100000001] [100000001] [271311686] [100000001] [100000001]
    [271311686]
127 -> 129 129  2 (4) [271311686] [100000001] [100000001]
    [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001] [100000001] [271311686]
    [100000001] [100000001] [271311686] [100000001] [100000001]
    [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001]
128 -> 4 4  2 (8) [100000001] [271311686] [100000001] [100000001]
    [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001]
129 -> 104 104  2 (8) [271311686] [100000001] [100000001]
    [271311686] [100000001] [100000001] [271311686] [100000001]
    [100000001] [271311686] [100000001] [100000001]
508 symbols, 130 rules 20984 total space
```

Listing 8: Last 10 lines of sequence grammar file `10-0-0-145_2021-01-10tx.seq`.

.3.2 The SEQUITUR Grammar Rules

The SEQUITUR grammar rules have the following syntax[1] shown in figure 6:

[1] Provided SEQUITUR is run with the parameters `rtTk 2`. For a full list of SEQUITUR options and parameters see listing 12.

```
Rule: <Rule Number> '->' <ID> <ID> <Rule Count> '(' <Input Count> ')'
                                                  <Sub-sequence expansion>
ID: <Rule Number> | <Token ID>

Token ID: '[' <9 Digit Integer> ']'

Rule Count: Integer  % The number of times the rule is used in the
                     % entire grammar

Input Count: Integer % The number of times the rule is used in the
                     % raw input sequence

Sub-sequence expansion: <space><Token ID>*

space: ' '
```

Figure 6: BNF grammar to parse rules generated by the SEQUITUR sequence compression system.

Hierarchical SEQUITUR grammar navigator

The *Hierarchical* SEQUITUR *grammar navigator* is the north-west panel and enables browsing of the SEQUITUR grammar produced from the input sequence.

An example using the input sequence file 10-0-0-145_2021-01-10.csv is shown in figure 7. The hierarchical navigator uses a *file browser* metaphor – *directories* represent elements that are in the compressed input sequence (start rule **S0**) produced by SEQUITUR; whilst a *file* within a directory represents the sub-sequence expansion of grammar rules.

Root node

Selecting the *Root node* of the *Hierarchical* SEQUITUR *grammar navigator* provides details in the North-east panel showing the raw output from the SEQUITUR processing of the input sequence. This includes:

1. Start rule S0: the 0 rule that contains a representation of the compressed original sequence.

2. Hierarchical Rules, their frequency in the training data; their full expansion back to source token_ids.

3. Statistics about the SEQUITUR run (appear at the end and are not shown in the adjacent image).

This information is illustrated in figure 8. Recall that integers enclosed in `[ ... ]` represent the raw token ID in the input sequence, while the plain integers represent the expansion grammar rule number.

Token ID nodes

Selecting a *Token ID node* – as a *file icon* – from the *Hierarchical* `SEQUITUR` *grammar navigator* provides details in the North-east panel showing the the following information about the token and its occurrence in the input sequence.

1. Token ID (e.g. `271311686`).
2. The underlying data for the token (e.g. `10.0.0.145 10.0.0.137 (name_not_found) 53 udp dns`).
3. Frequency of the token's occurrence in the training sequence (e.g. `1358`).
4. The position(s) in the training sequence that the token occurs (e.g. `positions: [2,5,6,8,10,11, ... ,3608,3612,3613]`).

This is illustrated in figure 9.

`SEQUITUR` **Grammar Rules**

Selecting a `SEQUITUR` *Rule number* – as a *directory icon* – from the *Hierarchical* `SEQUITUR` *grammar navigator* provides details in the North-east panel showing the the following information about the `SEQUITUR` generated expansion rule and its occurrence in the input sequence.

1. Rule ID and its definition: e.g. `Rule: [head(18),token(271311686),rule(162)]`[2]
2. Full expansion of the sequence of token ids that the rule represents. E.g. `Expansion: [271311686,847213017,847213017,271311686]`
3. Statistics relating to the rule and the expansion. E.g. `Length: 4 Frequency in training: 21/3613 Frequency in the grammar: 19A`
4. A table of expansion token IDs and the underlying data for the tokens. E.g.
   ```
   271311686:  10.0.0.145 10.0.0.137 (name_not_found) 53 udp dns
   847213017:  10.0.0.145 3.209.99.56 (ec2-3-209-99-56.compute-1.amazonaws.com) 22 tcp -
   847213017:  10.0.0.145 3.209.99.56 (ec2-3-209-99-56.compute-1.amazonaws.com) 22 tcp -
   271311686:  10.0.0.145 10.0.0.137 (name_not_found) 53 udp dns
   ```

This is illustrated in figure 10.

[2]This is the internal representation of `SEQUITUR`'s rule representation of `18 -> [271311686] 162`, an example of the form `<head> -> <body 1> <body 2>`.

```
# Adapted from: https://stackoverflow.com/questions/11513484/1d-number-array-
    clustering
eps = 0.2

points = [0.1, 0.31,  0.32, 0.45, 0.35, 0.40, 0.5, 0.7, 0.91 ]

clusters = []
points_sorted = sorted(points)
curr_point = points_sorted[0]
curr_cluster = [curr_point]
for point in points_sorted[1:]:
    if point <= curr_point + eps:
        curr_cluster.append(point)
    else:
        clusters.append(curr_cluster)
        curr_cluster = [point]
    curr_point = point
clusters.append(curr_cluster)
print(clusters)

# Python 3.8.10 (/usr/bin/python3)
# >>> %Run 1DArrayClustering.py
# [[0.1], [0.31, 0.32, 0.35, 0.4, 0.45, 0.5, 0.7], [0.91]]
# >>>
```

Listing 9: 1D Array Clustering – simple *Python* illustration code. A sequence of real values is partitioned into clusters on the basis of adjacent elements being no more than ϵ apart.

.3.3 Visualisation of input sequence

A colour-based depiction of the original input sequence, overlaid on the `SEQUITUR` grammar rules is displayed on the southern panel of the visualisation tool. This provides a method of visualising recurring sub-sequences (`SEQUITUR` rules) – only considering the order of events. Some additional adornments are added that refer to the precise timestamps of the events. These are discussed in more detail below.

Start of the hour markers

For example, figure 12 identifies the token (35034446 at position 945 out of 3613) to be the first token in the input sequence to occur after 06:00 UTC.

Simple 1-Dimensional Clustering

Examples of clustering segment markers

.3.4 NetMon Case Study supporting information

Examples of tokens and timestamps relating to Rule 18

Refer to table 9 below.

.3.5 Code fragments

```
with

"tokens" as
```

ID	Rule	N	Date	Token 271311686	Token 847213017	Token 847213017	Token 271311686
1	18	1	2021-01-10	00:10:01	00:10:01	00:10:01	00:10:05
2	18	2	2021-01-10	01:10:01	01:10:01	01:10:01	01:10:04
3	72	1	2021-01-10	02:10:02	02:10:02	02:10:02	02:10:05
4	18	3	2021-01-10	03:10:01	03:10:01	03:10:02	03:10:05
5	18	4	2021-01-10	04:10:02	04:10:02	04:10:02	04:10:05
6	18	5	2021-01-10	05:10:01	05:10:01	05:10:01	05:10:05
7							
8							
9	199	1	2021-01-10	08:10:01	08:10:01	08:10:02	08:10:05
10	18	6	2021-01-10	09:10:01	09:10:01	09:10:01	09:10:05
11	18	7	2021-01-10	10:10:01	10:10:02	10:10:02	10:10:05
12	18	8	2021-01-10	11:10:01	11:10:01	11:10:02	11:10:05
13	72	2	2021-01-10	12:10:01	12:10:02	12:10:05	12:10:05
14	18	9	2021-01-10	13:10:08	13:10:08	13:10:08	13:10:12
15	18	10	2021-01-10	14:10:01	14:10:01	14:10:01	14:10:05
16	199	2	2021-01-10	15:10:02	15:10:02	15:10:02	15:10:06
17	18	11	2021-01-10	16:10:01	16:10:01	16:10:01	16:10:05
18	18	12	2021-01-10	17:10:01	17:10:01	17:10:02	17:10:05
19	18	13	2021-01-10	18:10:02	18:10:02	18:10:02	18:10:05
20	18	14	2021-01-10	19:10:01	19:10:01	19:10:01	19:10:05
21	18	15	2021-01-10	20:10:01	20:10:01	20:10:01	20:10:05
22	18	16	2021-01-10	21:10:01	21:10:01	21:10:01	21:10:05
23	18	17	2021-01-10	22:10:01	22:10:01	22:10:01	22:10:05
24			2021-01-10	23:00:01	23:00:01	23:00:02	

Table 9: Tokens and timestamps relating to Rule 18. *Rule N* details the rule number and the sequential instance of the rule, and has the token sub-sequence represented by rule 18. In the case of record IDs 3 and 13, these are related to occurrences for rule 72 which has rule 18 as a prefix. In the case of record IDs 9 and 16, these are related to occurrences for rule 19 which also has rule 18 as a prefix. Records 7, 8, and 9 might be where rule 18 tokens might be expected, but the `SEQUITUR` analysis does not extract them into the hierarchical compositional grammar.

```
(
SELECT
"ts",
concat("id.orig_h", '_', "id.resp_h", '_', cast("id.resp_p" as
    varchar), '_', "proto", '_', "service")
as "token",
"uid"
FROM "5cbe4ac3_6178_508c_b9e2_dc44bb8025aa"."conn"
where
year in (2021) and
month in (01) and
day in (10, 11)
and
(
 "id.orig_h" = '10.0.0.145'
 or
"id.resp_h" = '10.0.0.145'
)
order by "ts" asc
)

SELECT
"ts" as "timestamp",
"token",
SUBSTR(CAST(abs(from_big_endian_64(xxhash64(CAST("token" AS
    varbinary)))) as varchar), 1, 9) as "token_uid",
year("ts") as "year",
month("ts") as "month",
day("ts") as "day",
"uid" as "conn_uid"
from "tokens"
```

Listing 10: SQL code for the generation of string tokens.

```
 Results
  timestamp token uid year  month day conn_uid
1 2021-01-09 23:10:01.489 10.0.0.145_3.209.99.56_22_tcp_- 847213017
     2021  1 9 CNhRGcgA051QufQd7
2 2021-01-10 00:00:03.135 10.0.0.145_169.254.169.254_80_tcp_-
    350344446 2021  1 10  CCsBTv1Te0if5EF777
3 2021-01-10 00:00:03.230 10.0.0.145_10.0.0.137_53_udp_dns
    271311686 2021  1 10  CJTneEzIgNlvQyjJ2
4 2021-01-10 00:00:03.262 10.0.0.145_52.95.132.200_443_tcp_-
    410067030 2021  1 10  Cz5cJ74Snj7XkFICng
5 2021-01-10 00:00:03.293 10.0.0.145_52.95.132.200_443_tcp_-
    410067030 2021  1 10  CpepdIPD7uWbJ6dqg
```

```
6 2021-01-10 00:00:04.028 10.0.0.145_10.0.0.137_53_udp_dns
    271311686 2021  1 10  Czwm0Q1AedgYgkRIe
7 2021-01-10 00:00:04.029 10.0.0.145_10.0.0.137_53_udp_dns
    271311686 2021  1 10  CepU6O1S80ZPwXFQA7
8 2021-01-10 00:00:04.145 10.0.0.145_169.254.169.254_80_tcp_-
    350344446 2021  1 10  C134zD4U9Zk4N0eTp9
9 2021-01-10 00:00:04.362 10.0.0.145_10.0.0.137_53_udp_dns
    271311686 2021  1 10  C74I376AD8ChxNoIa
10  2021-01-10 00:00:04.364 10.0.0.145_52.95.132.200_443_tcp_-
    410067030 2021  1 10  CdJxTo3FDQBJFKmjei
```

Listing 11: 10 sample records from the generation of string tokens.

.3.6 SEQUITUR options

```
usage: sequitur -cdpqrtTuz -k <K> -e <delimiter> -f <max symbols> -
    m <memory_limit>

-p    print grammar at end
-d    treat input as symbol numbers, one per line
-c    compress
-u    uncompress
-m    use this amount of memory, in MB, for the hash table (default
     1000)
-q    quiet: suppress progress numbers on stderr
-r    reproduce full expansion of rules after each rule
-t    print rule usage in the grammar after each rule
-T    print rule usage in the input after each rule
-z    put rule S in file called S, other rules on stdout as usual
-k    set K, the minmum number of times a digram must occur to form
     rule
      (default 2)
-e    set the delimiter symbol. Rules will not be formed across (i.
    e.
      including) delimiters. If with -d, 0-9 are treated as numbers
-f    set maximum symbols in grammar (memory limit). Grammar/
    compressed output
      will be generated once the grammar reaches this size
```

Listing 12: SEQUITUR command line options.

.4 ACUITY Appendix

.4.1 Data Understanding

The data has been collected by the Zeek[Zee21] Intrusion Detection and Network Monitoring System which logs metadata extracted from the observed network communications. Only the communications that passes through the network monitoring device are logged for analysis[3]. Zeek stores captured meta-data about network events in a relational model, with interlinked tables. The data is transferred into a relational database (`Presto` [Pre22]) for ease of retrieval. The map of the metadata schema is shown in Figure 2.2. The master table – `conn` – keeps track of all connections observed.

An example record and details of the meta data fields available in the master `conn` log are shown in Table 1. The example shows an `http` (service field) connection between a client at `10.0.100.55:51475` (`id.orig_h:id.orig_p` fields) and a server at `192.29.32.24:80` via the `tcp` protocol, initiated at `2020-12-31 23:59:22` (ts field). Zeek maintains a unique uid (field), `CSuVAz2v3MtaiPkaja`, which cross references detailed records from the `http` table (not shown).

Many of the Zeek tables have a similar initial set of columns with `ts`, `uid`, `id.orig_h`, `id.orig_p`, `id.resp_h`, and `id.resp_p` recurring in many tables.

For the experimental work we focused mainly on data extracted from fields of the `conn` table[4]. The fields and sample values are shown in table 1. For event records pertaining to network connections of the particular device at IP address 10.0.0.145, the fields `id.orig_h`, `id.resp_h`, `id.resp_p`, `proto`, and `service`, are projected into a concatenated string – referred to as a **token**. This string-based token is then hashed to a (unique) 9 digit integer to produce a **token_id**. This data preparation process is described in more detail below in section .4.2.

.4.2 Data Preparation

In this section we document the data transformations and production of the background knowledge for the ILP runs. The background knowledge is categorised in to three broad areas

- Raw Zeek `conn`ection table event records;

[3] An eavesdropper can only record what it observes. Encryption and wireless communications are not explicitly monitored by our device.

[4] Future work may analyse log tables other than `conn`.

- Extracted Zeek `dns` and `dhcp` table event records;
- Sequence pattern data extracted by the `SEQUITUR` system (cite some previous report) including a sequence grammar and supporting data elements.
- Hand produced `Prolog` code providing logical relationships between entities.

Each of these areas will be discussed below, but first we will be guided by the main driver file in `ACUITY`, the `.b` file.

.4.3 exp_seq.b: The background knowledge file

We wish to interactively 'explain the sequences' and the main coordinating driver file is held in: `~/gits/AI-Data-Science/trunk/src/acuity/experiments/10.0.0.145/data/ exp_seq.b`

Determinations

Determinations describe the heads and bodies of clauses that `ACUITY` can utilize when building hypotheses.

```
:- determination(exp_seq/1, rule_length/2).
:- determination(exp_seq/1, rule_start_zeek_uuid/2).
:- determination(exp_seq/1, rule_start/2).
:- determination(exp_seq/1, has_rule_token_hash/3).
:- determination(exp_seq/1, event/3).
:- determination(exp_seq/1, next_to/2).
:- determination(exp_seq/1, has_port_service/2).
:- determination(exp_seq/1, has_service/2).
:- determination(exp_seq/1, has_dest_name/2).
:- determination(exp_seq/1, has_dest_suffix/2).
:- determination(exp_seq/1, beaconing_every_hour/2).
:- determination(exp_seq/1, cloud_storage_connection/2).
```

Listing 13: `exps_eq.b` Determinations code fragment.

Modes

Modes describe the predicates and their type maps that `ACUITY` can utilize when building hypotheses.

```
% The head atom
:- mode(1, exp_seq(+rule_id)).
% Body literals
:- mode(1, rule_start_zeek_uuid(+rule_id, -zeek_uuid)).
```

```
5  :- mode(1, rule_start(+rule_id, -seq_pos)).
6  :- mode(*, rule_has_token_hash(+rule_id, -token_hash, -seq_pos)).
7  :- mode(*, rule_event(+rule_id, -zeek_uuid, -date_time, -token_hash
       )).
8  :- mode(*, next_to(+zeek_uuid, -zeek_uuid)).
9  :- mode(*, ahead(+zeek_uuid, -zeek_uuid, #offset)).
10 :- mode(*, behind(+zeek_uuid, -zeek_uuid, #offset)).
11 :- mode(*, has_port_service(+zeek_uuid, -port_service)).
12 :- mode(*, has_port_service(+zeek_uuid, #port_service)).
13 :- mode(*, has_service(+zeek_uuid, -service)).
14 :- mode(*, has_service(+zeek_uuid, #service)).
15 :- mode(*, has_dest_name(+zeek_uuid, -dest_name)).
16 :- mode(*, has_dest_name(+zeek_uuid, #dest_name)).
17 :- mode(*, has_dest_suffix(+zeek_uuid, -dest_suffix)).
18 :- mode(*, has_dest_suffix(+zeek_uuid, #dest_suffix)).
19 :- mode(1, beaconing_every_hour(+rule_id, #minute_hand)).
20 :- lazy_evaluate(beaconing_every_hour/2). %NB
21 :- mode(1, cloud_storage_connection(+zeek_uuid, #storage_name)).
22 :- mode(1, rule_length(+rule_id, -integer)).
23 :- mode(1, rule_length(+rule_id, #integer)).
```

Listing 14: `exp_seq.b` Modes code fragment.

Notice that some of the modes seem similar, but have a difference in the form of the typing/variablising decorations. In line 11 of listing 14 the final argument is `-port_service` while the similar mode on line 12 is `#port_service`[5]. This indicates that we are interested in both chaining a variable of the original value of `port_service` as well as the actual value of `port_service` in our hypothosese.

See appendix .4.8 for examples of the atoms that are generated using the mode declarations and the loaded background knowledge.

Settings

Settings determine how the underlying `ALEPH` system is configured.

```
1  :- set(clauselength, 10).
2  :- set(depth, 50000).
3  :- set(i,7).   % At 5 useful literals start to appear in bottom
4  :- set(nodes, 10000).
5  :- set(verbosity, 0).
6  :- set(openlist, 1000).  % beam size for search
```

Listing 15: `exp_seq.b` Settings code fragment.

[5]A `-` type produces an output variable from a constant, while a `#` type leaves the original constant in place.

The most noteworthy of the settings is `openlist` which specifies the size of the `beam-width` in the greedy search. `ALEPH`'s default value of `inf` does not allow the search to proceed very deeply into the lattice without exhausting the search node limit before any useful hypotheses are proposed, given the size of the most specific clause[6]. We set the `openlist` of candidates to be further explored to `10,000`.

.4.4 Zeek connection records

The intrusion detection system, Zeek, produces the metadata records for every network event observed. In this work, we are primarily interested[7] in using the most common event type, the `conn`ection event.

An example of the connection record is shown in table 1 above. For the explanatory learning, we are only interested in the subset of fields: `ts`, `id.orig_h`, `id.resp_h`, `id.resp_p`, `proto`, and `service`. These fields are concatenated together into a token, which is then hashed to a 9 digit integer for input to the sequence pattern system (see section .4.5 below).

The Zeek connection records are formatted into predicates describing the raw tokens as shown in listing 16.

```
?- raw_token(Date, TokenString, TokenHash, Year, Month, Day,
    ZeekUUID).
Date = date(2020, 12, 31, 23, 10, 1.9990000719999999, 0, 'UTC', -),
TokenString = '10.0.0.145_3.209.99.56_22_tcp_-',
TokenHash = 100233664,
Year = 2020,
Month = 12,
Day = 31,
ZeekUUID = 'CEwenXksBNnRoFxt8' ;
Date = date(2021, 1, 1, 0, 0, 2.213999986, 0, 'UTC', -),
TokenString = '10.0.0.145_10.0.0.137_53_udp_dns',
TokenHash = 363783709,
Year = 2021,
Month = Day, Day = 1,
ZeekUUID = 'CMeo4R3uciQAGKgzB' ;
Date = date(2021, 1, 1, 0, 0, 2.223999977, 0, 'UTC', -),
TokenString = '10.0.0.145_10.0.0.137_53_udp_dns',
TokenHash = 363783709,
Year = 2021,
```

[6]For the sample most specific clause, (see listing 41 in the Appendix) the size is 55 body literals.

[7]This is because all network interactions, regardless of protocol or service, have a Zeek connection event.

```
Month = Day, Day = 1,
ZeekUUID = 'CEnbbZ1DG74U0JLv22' ;
Date = date(2021, 1, 1, 0, 0, 3.124000072, 0, 'UTC', -),
TokenString = '10.0.0.145_10.0.0.137_53_udp_dns',
TokenHash = 363783709,
Year = 2021,
Month = Day, Day = 1,
ZeekUUID = 'CocP1sjYRXwZtMUxg' .
```

Listing 16: `raw_token`s generated from Zeek `conn`ection event records code fragment.

Two further Zeek record types are utilized, to provide more information about the remote hosts that are being connected to – the `dns` and `dhcp` events. Samples are shown in the listings below.

```
dhcp(timestamp(2021, 1, 1, 11, 16, 46, 895000000), '
    CxjvB54q3IN1RaZqqf', '10.0.0.145', 68, '10.0.0.137', 67, 'b8
    :27:eb:5b:23:31', '10.0.0.145', '1 days 00:00:00.000000000',
    3014660360, 2021, 1, 1).
dhcp(timestamp(2021, 1, 1, 23, 16, 47, 215000000), '
    CQE4TWBNYVpy9U9nb', '10.0.0.145', 68, '10.0.0.137', 67, 'b8:27:
    eb:5b:23:31', '10.0.0.145', '1 days 00:00:00.000000000',
    3326895937, 2021, 1, 1).
dhcp(timestamp(2021, 1, 1, 23, 16, 47, 215000000), '
    CQE4TWBNYVpy9U9nb', '10.0.0.145', 68, '10.0.0.137', 67, 'b8:27:
    eb:5b:23:31', '10.0.0.145', '1 days 00:00:00.000000000',
    3326895937, 2021, 1, 1).
dhcp(timestamp(2021, 1, 1, 11, 16, 46, 895000000), '
    CxjvB54q3IN1RaZqqf', '10.0.0.145', 68, '10.0.0.137', 67, 'b8
    :27:eb:5b:23:31', '10.0.0.145', '1 days 00:00:00.000000000',
    3014660360, 2021, 1, 1).
dhcp(timestamp(2021, 1, 1, 11, 16, 46, 895000000), '
    CxjvB54q3IN1RaZqqf', '10.0.0.145', 68, '10.0.0.137', 67, 'b8
    :27:eb:5b:23:31', '10.0.0.145', '1 days 00:00:00.000000000',
    3014660360, 2021, 1, 1).
dhcp(timestamp(2021, 1, 1, 23, 16, 47, 215000000), '
    CQE4TWBNYVpy9U9nb', '10.0.0.145', 68, '10.0.0.137', 67, 'b8:27:
    eb:5b:23:31', '10.0.0.145', '1 days 00:00:00.000000000',
    3326895937, 2021, 1, 1).
dhcp(timestamp(2021, 1, 1, 23, 16, 47, 215000000), '
    CQE4TWBNYVpy9U9nb', '10.0.0.145', 68, '10.0.0.137', 67, 'b8:27:
    eb:5b:23:31', '10.0.0.145', '1 days 00:00:00.000000000',
    3326895937, 2021, 1, 1).
```

Listing 17: Imported Zeek `dhcp` event records.

```
dns(timestamp(2021, 1, 1, 18, 0, 2, 974000000), 'CXSLA7yIMCQOGG87c', '10.0.0.145', 54335, '10.0.0.137', 53, udp, 46203, 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws.com', '3225.000000', 0, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 2, 974000000), 'CXSLA7yIMCQOGG87c', '10.0.0.145', 54335, '10.0.0.137', 53, udp, 49144, 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws.com,52.95.132.49', '3222.000000,3.000000', 0, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 3, 343000000), 'C604GU1pyzmkw3Od27', '10.0.0.145', 54233, '10.0.0.137', 53, udp, 9618, -, 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 3, 346000000), 'CVHoQ42GPmNRdEMvFc', '10.0.0.145', 57010, '10.0.0.137', 53, udp, 56057, -, 0, -, 0, -, 3, 'NXDOMAIN', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 3, 525000000), 'CZixg24AyHanfuHgPb', '10.0.0.145', 42709, '10.0.0.137', 53, udp, 60680, 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws.com', '3224.000000', 0, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 3, 525000000), 'CZixg24AyHanfuHgPb', '10.0.0.145', 42709, '10.0.0.137', 53, udp, 31295, 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws.com,52.95.132.49', '3224.000000,2.000000', 0, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 4, 335000000), 'Ca2yBo2mEtCwDIFyg', '10.0.0.145', 52280, '10.0.0.137', 53, udp, 18482, -, 0, -, 0, -, 3, 'NXDOMAIN', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 4, 335000000), 'Cjk3SZ10F3oJD9cRci', '10.0.0.145', 43963, '10.0.0.137', 53, udp, 17521, -, 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 5, 275000000), 'CaHdmErJjiZCCr0Jk', '10.0.0.145', 57479, '10.0.0.137', 53, udp, 46883, 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws.com,52.95.132.88', '3223.000000,3.000000', 0, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 5, 235000000), 'CaHdmErJjiZCCr0Jk', '10.0.0.145', 57479, '10.0.0.137', 53, udp, 23504, 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws.com', '3222.000000', 0, 2021, 1, 1).
dns(timestamp(2021, 1, 1, 18, 0, 2, 974000000), 'CXSLA7yIMCQOGG87c', '10.0.0.145', 54335, '10.0.0.137', 53, udp, 46203, 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0, -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws
```

```
    .com', '3225.000000', 0, 2021, 1, 1).
12 dns(timestamp(2021, 1, 1, 18, 0, 2, 974000000), 'CXSLA7yIMCQOGG87c'
    , '10.0.0.145', 54335, '10.0.0.137', 53, udp, 49144, 'ad6f88-5
    cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0,
     -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws
    .com,52.95.132.49', '3222.000000,3.000000', 0, 2021, 1, 1).
13 dns(timestamp(2021, 1, 1, 18, 0, 3, 343000000), 'C604GU1pyzmkw3Od27
    ', '10.0.0.145', 54233, '10.0.0.137', 53, udp, 9618, -, 0, -,
    0, -, 0, 'NOERROR', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
14 dns(timestamp(2021, 1, 1, 18, 0, 3, 346000000), 'CVHoQ42GPmNRdEMvFc
    ', '10.0.0.145', 57010, '10.0.0.137', 53, udp, 56057, -, 0, -,
    0, -, 3, 'NXDOMAIN', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
15 dns(timestamp(2021, 1, 1, 18, 0, 3, 525000000), 'CZixg24AyHanfuHgPb
    ', '10.0.0.145', 42709, '10.0.0.137', 53, udp, 60680, 'ad6f88-5
    cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0,
     -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws
    .com', '3224.000000', 0, 2021, 1, 1).
16 dns(timestamp(2021, 1, 1, 18, 0, 3, 525000000), 'CZixg24AyHanfuHgPb
    ', '10.0.0.145', 42709, '10.0.0.137', 53, udp, 31295, 'ad6f88-5
    cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0,
     -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws
    .com,52.95.132.49', '3224.000000,2.000000', 0, 2021, 1, 1).
17 dns(timestamp(2021, 1, 1, 18, 0, 4, 335000000), 'Ca2yBo2mEtCwDIFyg'
    , '10.0.0.145', 52280, '10.0.0.137', 53, udp, 18482, -, 0, -,
    0, -, 3, 'NXDOMAIN', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
18 dns(timestamp(2021, 1, 1, 18, 0, 4, 335000000), 'Cjk3SZ10F3oJD9cRci
    ', '10.0.0.145', 43963, '10.0.0.137', 53, udp, 17521, -, 0, -,
    0, -, 0, 'NOERROR', 0, 0, 0, 0, 0, -, -, 1, 2021, 1, 1).
19 dns(timestamp(2021, 1, 1, 18, 0, 5, 275000000), 'CaHdmErJjiZCCr0Jk'
    , '10.0.0.145', 57479, '10.0.0.137', 53, udp, 46883, 'ad6f88-5
    cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0,
     -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws
    .com,52.95.132.88', '3223.000000,3.000000', 0, 2021, 1, 1).
20 dns(timestamp(2021, 1, 1, 18, 0, 5, 235000000), 'CaHdmErJjiZCCr0Jk'
    , '10.0.0.145', 57479, '10.0.0.137', 53, udp, 23504, 'ad6f88-5
    cbe4ac3-6178-508c-b9e2-dc44bb8025aa.s3.amazonaws.com', 0, -, 0,
     -, 0, 'NOERROR', 0, 0, 0, 1, 0, 's3-ap-southeast-2-w.amazonaws
    .com', '3222.000000', 0, 2021, 1, 1).
```

Listing 18: Imported Zeek `dns` event records.

.4.5 Sequence Patterns

The first phase in the extraction of sequence patterns from network monitor data is to produce 9 digit numeric tokens identifiers that the `SEQUITUR` implementation

can utilize. A text token, produced by concatenating field values from `conn` table records is produced. For example, the token is made from conn fields as follows: `<id.orig_h>_<id.resp_h>_<id.resp_p>_<proto>_<service>`. One illustrative example is "`10.0.0.145_10.0.0.137_53_udp_dns`".

The text token is then hashed to a 9 digit number. E.g. the text token above produces token identifier `271311686`. A sequence of token identifiers, one per line, is provided to `SEQUITUR` in a file as input, from which it generates an hierarchical grammar with repeating subsequences.

The `SEQUITUR` hierarchical grammar produces a compressed start sequence (`S0`) and a set of grammar rules. All of this is part of the background information made available for learning. It is codified (in `Prolog`)as follows[8][9].

```
seq_data:symbols(1742).
seq_data:rules(366).
seq_data:s0([token(100233664),rule(1),rule(2),rule(3), ... ,rule
    (261),rule(86),rule(338),token(363783709),token(116817571)]).
```

Listing 19: Sequitur grammar – S0 code fragment. Note that the compressed start rule S0 has been summarized. The original contains 945 elements (representing 24 hours of events for the device).

Individual hierarchical rules generated by `SEQUITUR` and the sequence of event hash identifiers (or expansion) that the rule is made from are shown below.

```
   :
seq_data:rule(8,[head(8),token(363783709),rule(316)]).
seq_data:grammar_frequency(8,19).
seq_data:training_frequency(8,20).
seq_data:rule_expansion
    (8,[363783709,100233664,100233664,363783709]).
  :
```

Listing 20: Sequitur grammar – Rules code fragment relating to a single rule. The total number of rules is 366.

.4.6 Custom Background Knowledge

In this section we describe the custom background knowledge used as part of the explanatory learning task. One can think of this as helper functions that

[8]The example fragments are generated from a sample of one days data from device with IP address `10.0.0.145` recorded on 2021-01-01 which includes data from 3232 events.

[9]From `/home/sam/gits/AI-Data-Science/trunk/src/acuity/experiments/10.0.0.145/data/raw/1dde2934-cece-4fa2-3340-a98686edcff3.pl`

provide data enrichments, and fit sub models, that can be used in the explanation process.

Data Enrichments

One example of *data enrichments* is shown below.

```
% E.g.
% DestSuffix = 's3-ap-southeast-2-w.amazonaws.com',
user:has_dest_suffix(Uuid, DestSuffix) :-
    has_dest_name(Uuid, DestName),
    atomic_list_concat(L, '.', DestName),
    reverse(L, [DotCom, Name | _]),
    atomic_list_concat([Name, DotCom], '.', DestSuffix).
```

Listing 21: Extracting the destination hosts domain suffix.

Sub model fitting

This section describes the run-time fitting of a sub-model when seeking an explanatory model. It relates to the temporal clustering of particular event types. First we give some background necessary to understand the reason for needing to fit a sub-model.

Devices in a distributed computing environment are often arranged in an hierarchical order, where some devices are subordinate to others. We will use the rather old-fashioned terminology of *master* and *slave* to describe the roles of the devices. The *master* device provides direction and coordination by issuing *commands* to the *slave* devices. The *slave* device performs it duties whilst *seeking* and acting on commands from the *master* device.

Common tactics for receiving commands include *push* and *pull* messaging design patterns. In a *push* setting, the *slave* device persistently listens for messages pushed to it from the *master*. In a *pull* setting, the *slave* device regularly contacts the *master(s)* to determine if there are pending commands that it should act on.

A simple *pull* mode implementation has the *slave* device checking (also known as polling) for new commands at some fixed interval (say every 10 minutes). Cyber defenders know this behaviour as **beaconing** as it is often utilized by malware which is managed by a *Command and Control* system (C2). Beaconing is not always sinister as it is also used in many benign system architectures. In this work we build and utilize libraries to detect beaconing (if it exists), and determine what the beaconing frequency is. The determination of the beaconing frequency requires a **sub-model** to be fitted.

First we use a one-dimensional clustering system to identify time points on a

number line, around which other time stamps cluster. This is packaged into a `date_spectra` library.

```
:- module(date_spectra, [
              one_d_epsilon_clustering/3,   % +[SortedListOfNumbers
    ], +Epsilon, -[Clusters]
              segment_timestamps/3, % +[SortedKVPairs], +Epsilon,
    -[Segments]
              clusters_statistics/2, % +[Clusters], -Stats
              minutes/2, % +[DateTimes], -[Minutes]
              hours/2 % +[DateTimes], -[Hours]
          ]).
  :
```

Listing 22: Beaconing detection date spectra library code fragments.

The `date_spectra` library is then used for the explanatory learning, in which **Lazy Evaluation**[10] is used to **fit the clustering sub-model** to identify beaconing, and its related frequency (should it exist).

```
% ! Beaconing -- the regular phoning home as part of a command and
% control (C2) system.
%
% The simplest C2 system is to have the controlled device poll its
% master at a set time-of [day/hour].
%
% Here we use a linear clustering based on event's minute hands.
% A very restrictive definition for beaconing will be that if the
% event's minute hands cluster into a single(!) cluster AND
% the standard deviation is less than or equal to a minute.

minute_hand_clustering_std_threshold(1.0).

% Definition to use during lazy evaluation
% Identified as a call during lazy evaluation by the first argument
% being [P, N]
user:beaconing_every_hour([Ps, _Ns], RoundMinuteHand) :-
    !,
    rules_first_events_date_times(Ps, Ds),
    minutes(Ds, Ms),
    one_d_epsilon(Epsilon),
    one_d_epsilon_clustering(Ms, Epsilon, [C]),  %% NOTE: Single
    cluster only
    clusters_statistics([C], [Stat]),
    Stat = MinuteHand/Std/_ClusterSize,
    minute_hand_clustering_std_threshold(StdThreshold),
```

[10] Refer to section 3.12 in the `ALEPH` manual [Sri17] for more on Lazy Evaluation.

```
    Std =< StdThreshold,
    RoundMinuteHand is round(MinuteHand).

% definition to use during normal evaluation
user:beaconing_every_hour(RuleID, RoundMinuteHand) :-
    number(RuleID), number(RoundMinuteHand), !,
    rule_start_time(RuleID, DateTime1),
    DateTime1 = date(_Year, _Month, _Day, _Hour, RoundMinuteHand,
    _Seconds, _Offset, _TZ, _),
    true.

% definition to use during construction of bottom clause
user:beaconing_every_hour(RuleID, RoundMinuteHand) :-
    number(RuleID),
    rule_start_time(RuleID, DateTime1),
    DateTime1 = date(_Year, _Month, _Day, _Hour, RoundMinuteHand,
    _Seconds, _Offset, _TZ, _),
    true.
```

Listing 23: Beaconing detection Lazy Evaluation code fragments.

Convenience predicates

```
user:ahead(N1, N2, 0) :-
    N1 == N2,
    !.

user:ahead(N1, N2, 1) :-
    next_to(N1, N2).

user:ahead(N1, N3, Dist) :-
    next_to(N1, N2),
    ahead(N2, N3, D23),
    Dist is D23 + 1.

user:behind(N2, N1, D21) :-
    ahead(N1, N2, D21),
    !.
```

Listing 24: Convenience predictates relating to how far apart are two Zeek events.

.4.7 Data File

This Appendix provides some documentation and commentry for the primary data input file.

~/gits/AI-Data-Science/trunk/src/acuity/experiments/
10.0.0.145/data/10.0.0.145.data

Note that all code is Prolog syntax.

```
:- module('10_0_0_145_data', [
              load_data/0,
              load_csv/0,
              load_exported_zeek_tables/0,
              reset_data/0,
              sample_ntp_seq/1
          ]).

:- use_module(library(apply)).
:- use_module(library(csv)).
:- use_module(library(lists)).

:- use_module('/home/sam/gits/AI-Data-Science/trunk/src/acuity/
    experiments/10.0.0.145/data/raw/1dde2934-cece-4fa2-3340-
    a98686edcff3.pl').
:- use_module('/home/sam/gits/AI-Data-Science/trunk/src/sequitur/
    experiment/hand_prototype/prolog/reverse_lookup.pl').
:- use_module('/home/sam/gits/AI-Data-Science/trunk/src/acuity/
    experiments/10.0.0.145/background/date_spectra.pl').

:- use_module(library(debug)).
:- debug(header).
:- nodebug(process_seq_data).
:- debug(reset).
:- nodebug(process_event_stream).
:- debug(load_raw_tokens).
```

Listing 25: Prolog code fragment 1.

.4.8 Modes – Atom Samples

This appendix provides samples of atoms generated from the mode language. The first code listing demonstrates how to load the source data and related libraries into Prolog, while the subsequent listings provide examples of atoms generated.

```
sam@neon:~/gits/AI-Data-Science/trunk/src/acuity/experiments
    /10.0.0.145/data$ swipl -s 10.0.0.145.data
Welcome to SWI-Prolog (threaded, 64 bits, version 8.5.4-34-
    g8c2cd4a78)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software
    .
```

```
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- load_data.
% load_raw_tokens complete
true .
```

Listing 26: Loading the data into `Prolog`.

```
?- rule_start_zeek_uuid(8, Zeek_UUID).
Zeek_UUID = 'C1dmkX20dLMu7gskM1' ;
Zeek_UUID = 'Cv9XjhZ6IV1PnXW96' ;
Zeek_UUID = 'CJtVyY1seh3HAj3BBc' ;
Zeek_UUID = 'Cox9r33gxKbQ948BT9' ;
Zeek_UUID = 'CeBhbq34RmDtr7haDh' ;
Zeek_UUID = 'Ce0P3F28gw0YLhhk0i' ;
Zeek_UUID = 'CrPCDrlkbwdEYrRF2' ;
Zeek_UUID = 'CVVKba4gDDeSod2mmd' ;
Zeek_UUID = 'Cpb0do3HJmakUYXkvi' ;
Zeek_UUID = 'CSPFs5aRE072e7Asb' ;
Zeek_UUID = 'C4wfBipm6I4TBT6b9' ;
Zeek_UUID = 'CcOBGk3lgrqN0qs0nl' ;
Zeek_UUID = 'Cu0jzm4a2VVIlJO2M' ;
Zeek_UUID = 'CbGXXd2PbRwFrpX2I2' ;
Zeek_UUID = 'Cvwi3o2CfVAZ6BN437' ;
Zeek_UUID = 'CCo9d02SMM0vTMmlN7' ;
Zeek_UUID = 'C17FoMQXgTetT2Mpb' ;
Zeek_UUID = 'C1qlt844jJ8qO4WC75'.
```

Listing 27: Examples of `mode(1, rule_start_zeek_uuid(+rule_id, -zeek_uuid)).` atoms.

```
?- rule_start(8, Seq_Pos).
Seq_Pos = 21 ;
Seq_Pos = 71 ;
Seq_Pos = 122 ;
Seq_Pos = 177 ;
Seq_Pos = 228 ;
Seq_Pos = 295 ;
Seq_Pos = 346 ;
Seq_Pos = 402 ;
Seq_Pos = 960 ;
Seq_Pos = 1118 ;
Seq_Pos = 1274 ;
```

```
Seq_Pos = 1533 ;
Seq_Pos = 1835 ;
Seq_Pos = 2000 ;
Seq_Pos = 2153 ;
Seq_Pos = 2452 ;
Seq_Pos = 2743 ;
Seq_Pos = 3051.
```

Listing 28: Examples of `mode(1, rule_start(+rule_id, -seq_pos)).` atoms.

```
?- rule_has_token_hash(8, Token_hash, Seq_Pos).
Token_hash = 363783709,
Seq_Pos = 1 ;
Token_hash = 100233664,
Seq_Pos = 2 ;
Token_hash = 100233664,
Seq_Pos = 3 ;
Token_hash = 363783709,
Seq_Pos = 4.
```

Listing 29: Examples of `mode(*, rule_has_token_hash(+rule_id, -token_hash, -seq_pos)).` atoms.

```
?- rule_event(8, Zeek_UUID, Date_Time, Token_hash).
Zeek_UUID = 'C1dmkX20dLMu7gskM1',
Date_Time = date(2021, 1, 1, 0, 10, 1.8970000740000001, 0, 'UTC',
    -),
Token_hash = 363783709 ;
Zeek_UUID = 'CWsc9F11AsQOiMyt6g',
Date_Time = date(2021, 1, 1, 0, 10, 1.917999982, 0, 'UTC', -),
Token_hash = 100233664 ;
Zeek_UUID = 'CVq5uNisuvbiTQ7Kh',
Date_Time = date(2021, 1, 1, 0, 10, 2.144000053, 0, 'UTC', -),
Token_hash = 100233664 ;
Zeek_UUID = 'C7OGCR3E7qqOdmd0Z4',
Date_Time = date(2021, 1, 1, 0, 10, 5.5720000259999996, 0, 'UTC',
    -),
Token_hash = 363783709 ;
Zeek_UUID = 'Cv9XjhZ6IV1PnXW96',
Date_Time = date(2021, 1, 1, 1, 10, 1.163000106, 0, 'UTC', -),
Token_hash = 363783709 ;
Zeek_UUID = 'Czj3pl449Tt4Wwf1a7',
Date_Time = date(2021, 1, 1, 1, 10, 1.196000099, 0, 'UTC', -),
Token_hash = 100233664 ;
Zeek_UUID = 'CH9MPd1V2WrONP1fK4',
Date_Time = date(2021, 1, 1, 1, 10, 1.424000024, 0, 'UTC', -),
```

```
Token_hash = 100233664 ;
Zeek_UUID = 'CRoM6o2T3AUlF2NVy6',
Date_Time = date(2021, 1, 1, 1, 10, 5.032000064, 0, 'UTC', -),
Token_hash = 363783709 ;
Zeek_UUID = 'CJtVyY1seh3HAj3BBc',
Date_Time = date(2021, 1, 1, 2, 10, 1.539999961, 0, 'UTC', -),
Token_hash = 363783709 ;
Zeek_UUID = 'CrQ34u4me7zLeHqYe1',
Date_Time = date(2021, 1, 1, 2, 10, 1.5620000360000001, 0, 'UTC',
    -),
Token_hash = 100233664 ;
Zeek_UUID = 'Cwtna93bk3eSnyuodj',
Date_Time = date(2021, 1, 1, 2, 10, 1.7939999100000001, 0, 'UTC',
    -),
Token_hash = 100233664 ;
Zeek_UUID = 'CgA3ET15faj3KuUoGi',
Date_Time = date(2021, 1, 1, 2, 10, 5.424000024, 0, 'UTC', -),
Token_hash = 363783709 ;
```

Listing 30: Examples of `mode(*, rule_event(+rule_id, -zeek_uuid, -date_time, -token_hash))`. atoms.

```
?- next_to('C1dmkX20dLMu7gskM1', Zeek_UUID).
Zeek_UUID = 'CWsc9F11AsQOiMyt6g'.
```

Listing 31: Example of `mode(*, next_to(+zeek_uuid, -zeek_uuid))`. atoms.

```
?- ahead('C1dmkX20dLMu7gskM1', Zeek_UUID, Offset).
Zeek_UUID = 'CWsc9F11AsQOiMyt6g',
Offset = 1 ;
Zeek_UUID = 'CVq5uNisuvbiTQ7Kh',
Offset = 2 ;
Zeek_UUID = 'C7OGCR3E7qqOdmd0Z4',
Offset = 3 ;
Zeek_UUID = 'CllSR51bfVb035a79',
Offset = 4 ;
Zeek_UUID = 'Cs7sgs1zt09TjMLwb',
Offset = 5 .
```

Listing 32: Examples of `mode(*, ahead(+zeek_uuid, -zeek_uuid, #offset))`. atoms.

```
?- behind('C1dmkX20dLMu7gskM1', Zeek_UUID, Offset).
Zeek_UUID = 'Cdshqp2k69HH6dXYUc',
```

```
Offset = 1.
```

Listing 33: Examples of `mode(*, behind(+zeek_uuid, -zeek_uuid, #offset)).` atoms.

```
?- has_port_service('C1dmkX20dLMu7gskM1', Port_service).
Port_service = dns.
```

Listing 34: Examples of `mode(*, has_port_service(+zeek_uuid, -port_service))` and `mode(*, has_port_service(+zeek_uuid, #port_service))` atoms.

```
?- has_service('C1dmkX20dLMu7gskM1', Service).
Service = dns.
```

Listing 35: Examples of `mode(*, has_service(+zeek_uuid, -service))` and `mode(*, has_service(+zeek_uuid, #service))` atoms.

```
?- has_dest_name('CEwenXksBNnRoFxt8', Dest_name).
Dest_name = 'ec2-3-209-99-56.compute-1.amazonaws.com'.
```

Listing 36: Examples of `mode(*, has_dest_name(+zeek_uuid, -dest_name))` and `mode(*, has_dest_name(+zeek_uuid, #dest_name))` atoms.

```
?- has_dest_suffix('CEwenXksBNnRoFxt8', Dest_suffix).
Dest_suffix = 'amazonaws.com'.
```

Listing 37: Examples of `mode(*, has_dest_suffix(+zeek_uuid, -dest_suffix))` and `mode(*, has_dest_suffix(+zeek_uuid, #dest_suffix))` atoms.

```
?- beaconing_every_hour(8, Minute_hand).
Minute_hand = 10 .
```

Listing 38: Examples of `mode(1, beaconing_every_hour(+rule_id, #minute_hand))` atoms.

```
?- cloud_storage_connection('CXSLA7yIMCQOGG87c', Storage_name).
Storage_name = 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa' .
```

Listing 39: Examples of `mode(1, cloud_storage_connection(+zeek_uuid, #storage_name))` atoms.

```
?- rule_length(8, Integer).
Integer = 4.
```

Listing 40: Examples of `mode(1, rule_length(+rule_id, -integer))` and `mode(1, rule_length(+rule_id, #integer))` atoms.

```
    exp_seq(A) :-
    [1]     rule_length(A,4),
    [2]     rule_start_zeek_uuid(A,B),
    [3]     rule_start(A,C),
    [4]     beaconing_every_hour(A,10),
    [5]     next_to(B,D),
    [6]     has_port_service(B,E),
    [7]     has_port_service(B,dns),
    [8]     has_service(B,F),
    [9]     has_service(B,dns),
    [10]     has_dest_name(B,G),
    [11]     has_dest_name(B,name_not_found),
    [12]     next_to(D,H),
    [13]     has_port_service(D,I),
    [14]     has_port_service(D,ssh),
    [15]     has_service(D,J),
    [16]     has_service(D,-),
    [17]     has_dest_name(D,K),
    [18]     has_dest_name(D,'ec2-3-209-99-56.compute-1.amazonaws.
    com'),
    [19]     has_dest_suffix(D,L),
    [20]     has_dest_suffix(D,'amazonaws.com'),
    [21]     next_to(H,M),
    [22]     has_port_service(H,I),
    [23]     has_port_service(H,ssh),
    [24]     has_service(H,J),
    [25]     has_service(H,-),
    [26]     has_dest_name(H,K),
    [27]     has_dest_name(H,'ec2-3-209-99-56.compute-1.amazonaws.
    com'),
    [28]     has_dest_suffix(H,L),
    [29]     has_dest_suffix(H,'amazonaws.com'),
    [30]     next_to(M,N),
    [31]     has_port_service(M,E),
    [32]     has_port_service(M,dns),
    [33]     has_service(M,F),
    [34]     has_service(M,dns),
    [35]     has_dest_name(M,G),
    [36]     has_dest_name(M,name_not_found),
    [37]     cloud_storage_connection(M,'ad6f88-5cbe4ac3-6178-508c-
```

```
    b9e2-dc44bb8025aa'),
39  [38]    next_to(N,O),
40  [39]    has_port_service(N,P),
41  [40]    has_port_service(N,https),
42  [41]    has_service(N,J),
43  [42]    has_service(N,-),
44  [43]    has_dest_name(N,Q),
45  [44]    has_dest_name(N,'s3-ap-southeast-2-w.amazonaws.com'),
46  [45]    has_dest_suffix(N,L),
47  [46]    has_dest_suffix(N,'amazonaws.com'),
48  [47]    next_to(O,R),
49  [48]    has_port_service(O,P),
50  [49]    has_port_service(O,https),
51  [50]    has_service(O,J),
52  [51]    has_service(O,-),
53  [52]    has_dest_name(O,Q),
54  [53]    has_dest_name(O,'s3-ap-southeast-2-w.amazonaws.com'),
55  [54]    has_dest_suffix(O,L),
56  [55]    has_dest_suffix(O,'amazonaws.com').
```

Listing 41: Example `bottom clause` generated by saturating an example (of rule 8).

.4.9 Explaining `SEQUITUR` sequence rule 8

This section documents a proof-of-principle experiment using `ACUITY` to explain one of the subsequences extracted by `SEQUITUR` from network monitoring data of device at IP address `10.0.0.145` on 2021-01-01.

The `SEQUITUR` subsequence exemplar we will study is grammar `Rule 8`[11] Rule 8 contains 4 elements with token hashes(see listing 29) `[363783709, 100233664, 100233664, 363783709]`), and occurs 18 times in the twenty-four hour period under study.

By selecting rule 8 as the exemplar, `ALEPH` produces a most specific clause of 55 body literals (see listing 41). Even with this modest most specific clause anchoring the search space, there are a lot of clauses to try. The upper bound for the number of possible single clause explanations is 2.40×10^{32} (see appendix .4.10 for the calculation). Often ILP systems have multiple positive and negative examples that can be used in navigating the search space, but in the use case here, we are interested in explaining *a single positive example only* (no negative

[11] Many of the data relating to Rule 8 are provided as illustrative examples in the appendix in section .4.

examples at the start). It is the domain expert's interactions that constrain the search space and help the system provide an acceptable explanation.

Interactive Explanation generation

- **Initialisation:** Start `ACUITY` and load the exp_seq.b, and all subordinate files and data.

```
1
2 sam@neon:~/gits/acuity/src/acuity$ swipl -s acuity.pl
3
4
5 A L E P H
6 Version 6
7 Last modified: Sun Jan 1 12:00:00 UTC 2017
8
9 Manual: http://www.comlab.ox.ac.uk/oucl/groups/machlearn/Aleph
      /index.html
10
11 TO DO: Add Reset capability to ACUITY
12 and trigger it at initialisation.
13
14
15 Welcome to SWI-Prolog (threaded, 64 bits, version 8.5.4-34-
      g8c2cd4a78)
16 SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free
      software.
17 Please run ?- license. for legal details.
18
19 For online help and background, visit https://www.swi-prolog.
      org
20 For built-in help, use ?- help(Topic). or ?- apropos(Word).
21
22 ?- cd('/home/sam/gits/AI-Data-Science/trunk/src/acuity/
      experiments/10.0.0.145/').
23 true.
24
25 ?- read_all('exp_seq').
26 % load_raw_tokens complete
27 [cannot open] [exp_seq.f]
28 [cannot open] [exp_seq.n]
29 true .
```

- **Select example to be explained** `exp_seq(8)` and allow the first hypothesis to be generated.

```
1 ?- induce(incremental).
```

```
2
3     Options:
4         -> "ok." to accept default example
5         -> Enter an example
6         -> ctrl-D or "none." to end
7
8     Response [default:none] ?
9 |: exp_seq(8).
10 [sat] [1]
11 [exp_seq(8)]
12
13 [bottom clause]
14 [literals] [57]
15    :
16    :
17 [hypothesis]
18 exp_seq(A).
```

The over general clause `exp_seq(A)` (*Every sequence is interesting*) is returned.

- **Rebut the overgeneral clause (and retry previous example)** This time, `ACUITY` suggests the over general clause `exp_seq(A) :- rule_length(A,B).` (*Every sequence that has a rule length is interesting* is returned.

- **Define the over-general clause body literal to be redundant (and retry previous example)** Redundant definition is performed by adding the following to the background knowledge `redundant(_Clause, rule_length(A, B)) :- var(B).` and setting `set(check_redundant, true).`. This produces a (slightly) more specific clause `exp_seq(A) :- rule_length(A,4).`

  Checking the `extent` of the proposed hypothesis shows there are thirty `SEQUITUR` rules that are 4 elements long.

```
1 |: extent.
2 Extent of proposed hypothesis is [exp_seq(1),exp_seq(8),
      exp_seq(9),exp_seq(18),exp_seq(55),exp_seq(56),exp_seq(60)
      ,exp_seq(71),exp_seq(86),exp_seq(87),exp_seq(109),exp_seq
      (115),exp_seq(127),exp_seq(135),exp_seq(140),exp_seq(150),
      exp_seq(164),exp_seq(166),exp_seq(182),exp_seq(202),
      exp_seq(205),exp_seq(227),exp_seq(232),exp_seq(253),
      exp_seq(261),exp_seq(268),exp_seq(270),exp_seq(279),
      exp_seq(311),exp_seq(332)]
3 [done]
```

  More work is required to produce an acceptable explanation for rule 8.

- **Add a negative example (and retry previous example)** We add `exp_seq(1)` as a negative example.

```
    Response?
|: overgeneral because not(exp_seq(1)).

[added new negative example]
```

This time the hypothesis returned relates to beaconing.

```
[hypothesis]
ntp_seq(A) :-
   beaconing_every_hour(A,10).
```

- **Constrain the hypotheses to contain interesting literals** We would like some more detail in the explanation and choose to specify some hypothesis constraints that have attracted our attention from the most specific clause – in particular insisting that body literals with identifiers 14 and 18 **must** appear in any hypothesis.

```
    [14]     has_port_service(D,ssh),
      :
    [18]     has_dest_name(D,'ec2-3-209-99-56.compute-1.
    amazonaws.com'),
```

```
[hypothesis]
ntp_seq(A) :-
   rule_start_zeek_uuid(A,B), next_to(B,C), next_to(C,D),
    has_port_service(D,ssh),
   has_dest_name(D,'ec2-3-209-99-56.compute-1.amazonaws.com').
```

The extent of this proposed hypothesis only *covers* three `SEQUITUR` rules this time.

```
|: extent.
Extent of proposed hypothesis is [exp_seq(8),exp_seq(191),
    exp_seq(215)]
[done]
```

- **Add a furthur constraint for hypotheses** Beaconing is a particularly intersting behaviour, so we insist that `ACUITY` tries to include it in any hypothesis it considers.

```
      :
    [4]     beaconing_every_hour(A,10),
      :
```

- **Accepting the hypothesis** The suggested explanatory hypothesis that has been co-produced by the machine learning system and the domain expert in the context of the data and background knowledge is provided below.

```
[hypothesis]
exp_seq(A) :-
   rule_start_zeek_uuid(A,B), next_to(B,C), next_to(C,D),
   has_port_service(D,ssh),
   has_dest_name(D,'ec2-3-209-99-56.compute-1.amazonaws.com'),
   beaconing_every_hour(A,10).
```

- **The accepted *Theory*** A *theory* is a set of acceptable hypotheses, plus the examples and any constraints.

```
[theory]

[Rule 1] [Pos cover = 1 Neg cover = 0]
exp_seq(A) :-
   rule_start_zeek_uuid(A,B), next_to(B,C), next_to(C,D),
   has_port_service(D,ssh),
   has_dest_name(D,'ec2-3-209-99-56.compute-1.amazonaws.com'),
   beaconing_every_hour(A,10).


[positives]
exp_seq(8).

[negatives]
exp_seq(sk_1_1_rule_id).
exp_seq(1).


true .
```

Listing 42: Explanatory Theory of rule 8 learned with `ACUITY`.

.4.10 Estimating the size of the search space

Following Muggleton 1995:

$P(T)$ is the number of predicate symbols in the logic program T.

$\mathcal{C}(T)$ is the number of constants in T.

$V(T)$ is the maximum number of variables in any clause in T.

a is the maximum arity of any predicate symbol in T.

l is the maximum cardinality of the body of any clause in T.
$|T|$ is the number of clauses in T.

- $\mathcal{A}(T)$ is the set of atoms that can be constructed with predicate symbols, constants and variables from T. $|\mathcal{A}(T)| \leq P(T).(\mathcal{C}(T) + V(T))^a$.
- $\mathcal{C}(T)$ is the set of definite clauses which can be made up with heads chosen from $\mathcal{A}(T)$ and bodies having at most l atoms chosen from $\mathcal{A}(T)$.

$$\mathcal{C}(T) \leq |\mathcal{A}(T)| \left(\begin{array}{c} |\mathcal{A}(T)| \\ \\ l \end{array} \right)$$

For the sample bottom clause shown in listing 41 we have the following:
The set of predicate symbols is {`rule_length/2, rule_start_zeek_uuid/2, rule_start/2, beaconing_every_hour/2, next_to/2, has_port_service/2, has_service/2, has_dest_name/2, has_dest_suffix/2, cloud_storage_connection/2`} with cardinality 10.
$P(T) = 10$

The set of variables is {A, ..., Q} with cardinality 17.
$V(T) = 17$

The set of constants is { `4, 10, dns, name_not_found, ssh, -, 'ec2-3-209-99-56.compute-1.amazonaws.com', 'amazonaws.com', 'ad6f88-5cbe4ac3-6178-508c-b9e2-dc44bb8025aa', https, 's3-ap-southeast-2-w.amazonaws.com'` } with cardinality 11.
$\mathcal{C}(T) = 11$

The maximum arity a of any of the predicate symbols is 2.
The upper bound on the number of atoms that can be constructed is

$$\begin{aligned} |\mathcal{A}(T)| &\leq P(T).(\mathcal{C}(T) + V(T))^a \\ &\leq 10.(11 + 17)^2 \\ &\leq 7840 \end{aligned}$$

Clause length l is set to 10 (see the setting from listing 15).

For this situation we are only interested in clauses with a single head `exp_seq/2` which reduces the formula for $\mathcal{C}(T)$ to:

$$\begin{aligned}
\mathcal{C}(T) &\leq \left(\begin{array}{c} |\mathcal{A}(T)| \\ \\ l \end{array} \right) \\
&\leq \left(\begin{array}{c} 7840 \\ \\ 10 \end{array} \right) \\
&\leq 2.40 \times 10^{32}
\end{aligned}$$

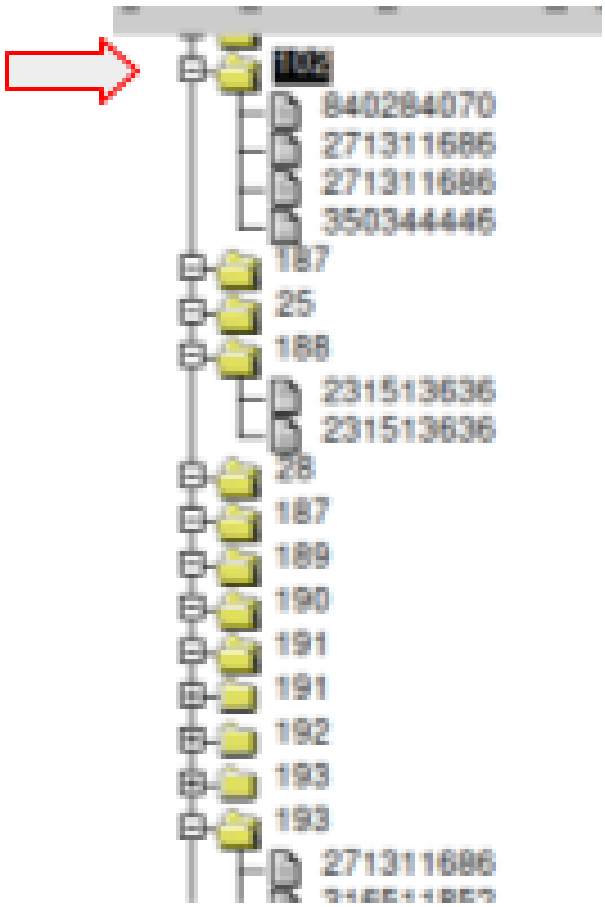

Figure 7: Simple `SEQUITUR` sequence viewer – Hierarchical `SEQUITUR` grammar navigator. Using a *file-browser* metaphor, *folders* represent grammar rules, *files* represent event `token_ids`. The vertical sequence of the expanded folder tree maps directly to the original input sequence of `token_ids`.

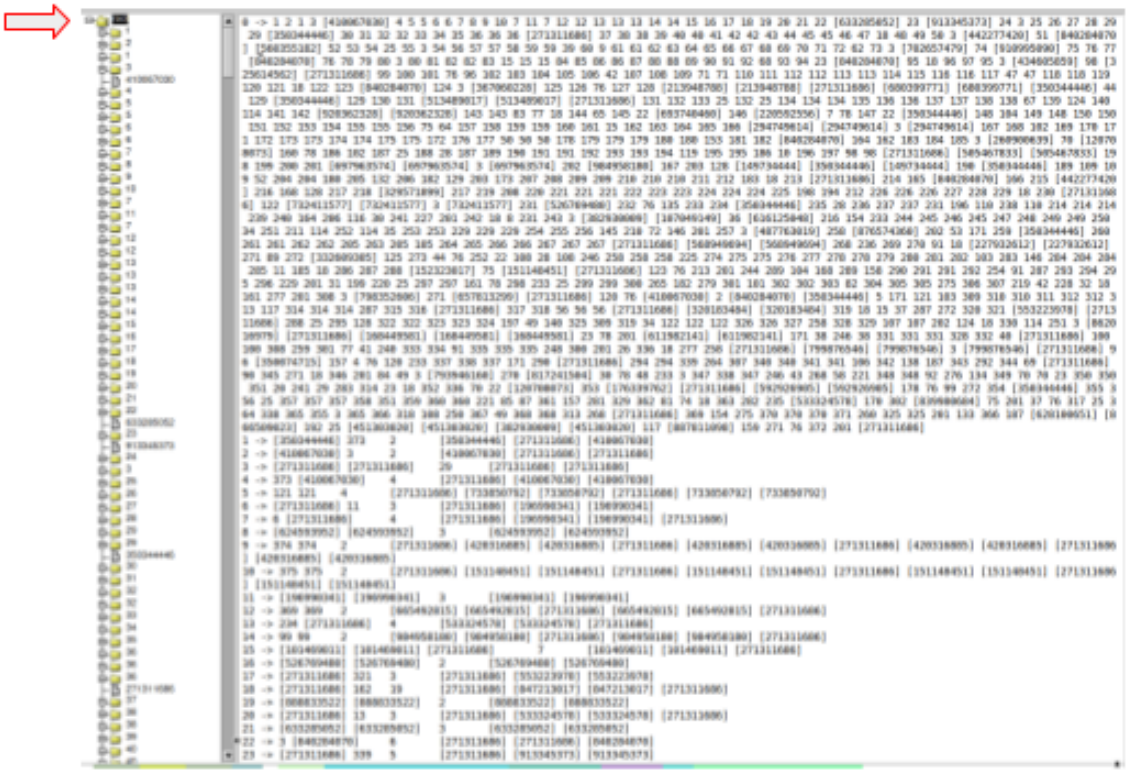

Figure 8: Simple SEQUITUR sequence viewer – the start rule **S0**. This displays the raw compression output generated by SEQUITUR with the start rule S0 being the first rule in hierarchical grammar. This shows the output of SEQUITUR compressing NetMon data collected of an IoT-like device.

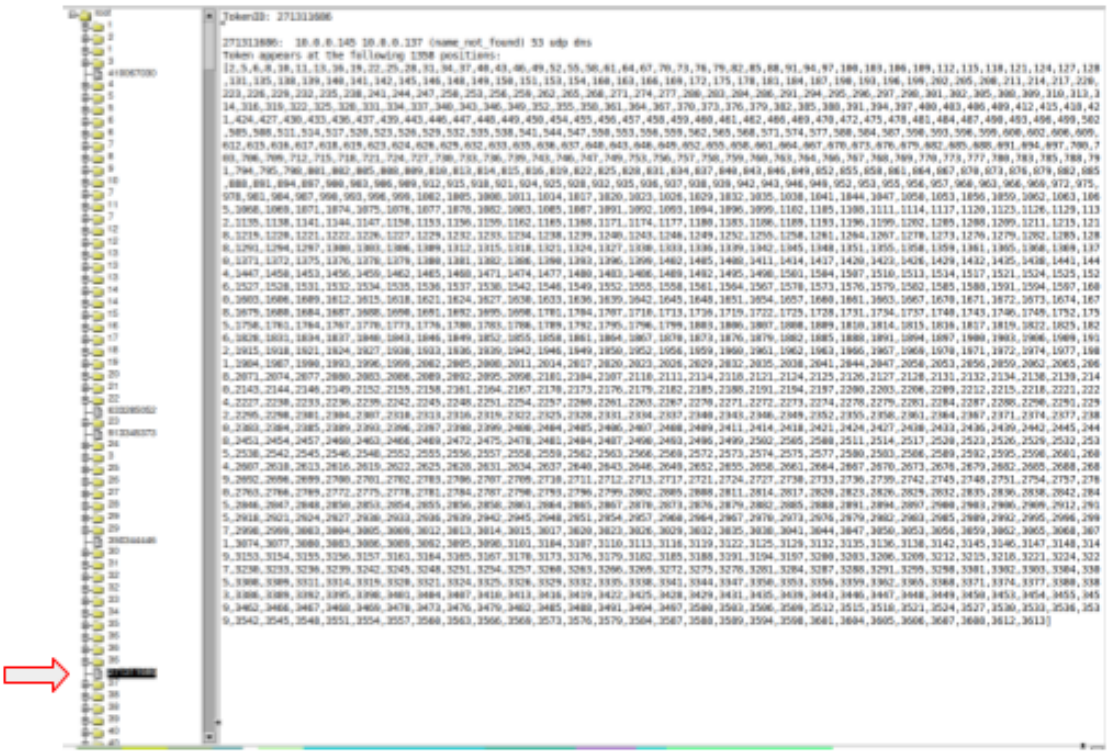

Figure 9: Simple SEQUITUR sequence viewer – an example token ID node.

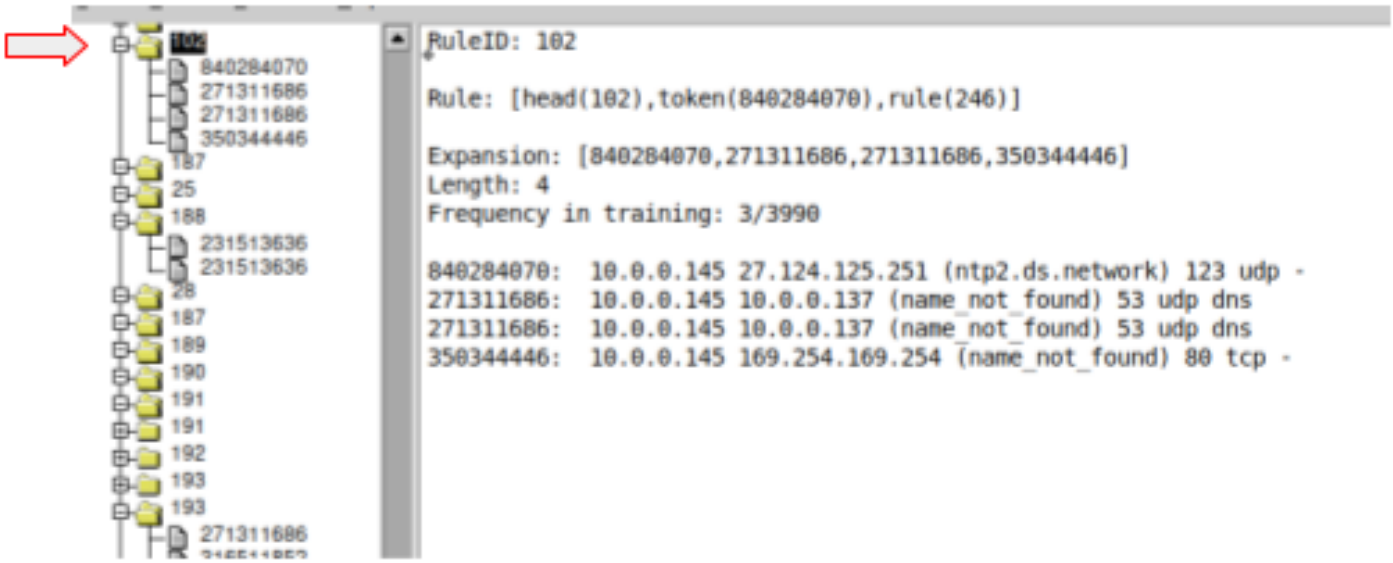

Figure 10: Simple `SEQUITUR` sequence viewer – an example rule node. For rule 18, this displays the *Rule ID*, *full expansion*, *statistics*, and underlying string tokens.

Figure 11: Simple SEQUITUR sequence viewer. Individual event tokens are represented by coloured squares. The sequence of events flows left-to-right, top-to-bottom. Underlayed on the event tokens are coloured rectangles representing recurring sub-sequence patterns extracted by SEQUITUR rules. Tool tips provide detail information for selected event tokens. Adornments relating to absolute timestamps can be added (e.g. which token is nearest to the start of an hour). Note that the purple highlighted borders are manually drawn, to support the narrative in the text.

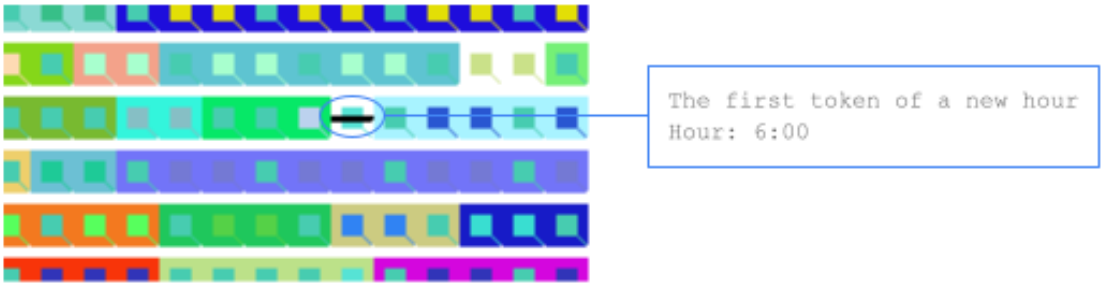

Figure 12: Simple SEQUITUR sequence viewer – an example hour marker. The black strike-through indicates the first token tile of the hour – for this example the hour is 06:00 UTC, and the token represents the 945^{th} event after midnight UTC.

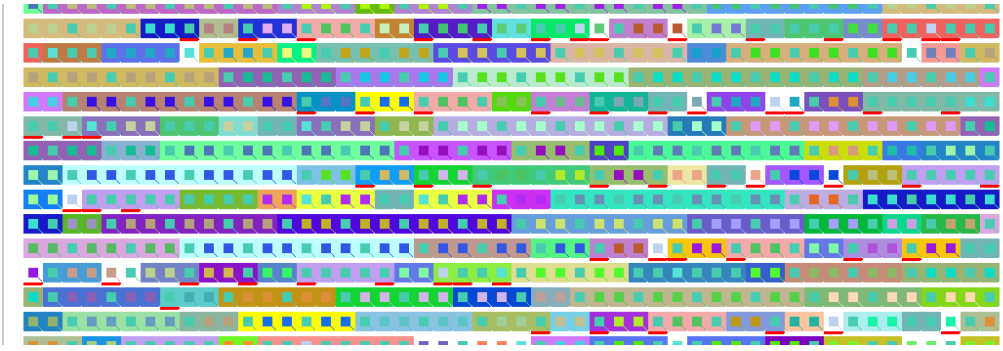

Figure 13: Simple SEQUITUR sequence viewer – examples of segment markers. Event token tiles underlined with a red line indicate the beginning of a sequence of event tokens in which each pair of elements of the sequence are not more than 5 seconds apart. The time between an underlined token tile and its predecessor is greater than 5 seconds.