HAW HAMBURG

BACHELOR THESIS
Finn-Frederik Jannsen

Hierarchical temporal memory for in-car network anomaly detection

Faculty of Computer Science and Engineering
Department Computer Science

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Finn-Frederik Jannsen

Hierarchical temporal memory for in-car network anomaly detection

Bachelor Thesis submitted as part of the Bachelor Examination
of the course *Bachelor of Science Technische Informatik*
at the Department Computer Science
of Faculty of Computer Science and Engineering
of Hamburg University of Applied Sciences

Supervisor: Prof. Dr. Franz Korf
Second Reviewer: Prof. Dr. Stephan Pareigis

Submitted on: 31. August 2021

Finn-Frederik Jannsen

Title of Thesis

Hierarchical temporal memory for in-car network anomaly detection

Keywords

HTM, Hierarchical temporal memory, Network Anomaly Detection, ML, IVN

Abstract

Automotive networks experience a transition from traditional bus systems to ethernet based communication in an attempt to optimize the communicational infrastructure of vehicles. The increased number of devices and additional interfaces in a combined network can produce new attack vectors. In order to keep such networks from being compromised, available solutions for anomaly detection and response methods need to be evaluated. This work examines the Machine Learning (ML) framework Hierarchical Temporal Memory (HTM) in terms of applicability for realtime anomaly detection by examining it's execution speed and detection performance. In addition techniques are invented that proved necessary for the system to reliably work in this environment. The framework's potential use is succesfully demonstrated on a realistic communication scenario which is interrupted by Denial of Service (DoS) attacks. Great noise robustness and detection rates can be achieved. While detection delay still amounts to a few 100 ms, better timings might be possible for less noisy input.

Finn-Frederik Jannsen

Thema der Arbeit

Hierarschischer Temporalspeicher für Anomalieerkennung in Fahrzeugbordnetzen

Stichworte

HTM, Hierarschischer Temporalspeicher, Netzwerk Anomalieerkennung, ML, IVN

Kurzzusammenfassung

Automobilnetzwerke erfahren einen Übergang von traditionellen Bussystemen zu Ethernet basierten Netzwerken mit dem Ziel, die kommunikative Infrastruktur von Automobilen zu optimieren. Die steigende Anzahl an Geräten und zusätzlichen Schnittstellen ermöglichen das Auftreten neuer Angriffsvektoren. Um das Netzwerk vor deren Effekten zu schützen, müssen verfügbare Lösungen zur Anomalieerkennung und Verteidigungsmechanismen evaluiert werden. Diese Arbeit untersucht das Machine Learning (ML) Framework Hierarchical Temporal Memory (HTM) in Bezug auf Anwendbarkeit für Echtzeit-Anomalieerkennung durch die Evaluation von Geschwindigkeit und Erkennungserfolg. Außerdem werden Mechanismen entwickelt, die für den erfolgreichen Einsatz in dieser Umgebung von Nöten sind. Der potentielle Nutzen des Frameworks wird erfolgreich anhand eines realistischen Kommunikations-Szenarios demonstriert, welches von DoS-Attacken unterbrochen wird. Gute Rauschrobustheit und Erkennungsraten können erzielt werden. Während die gemessene Erkennungsverzögerung noch wenige 100 ms beträgt, werden bessere Resultate für rauschfreieren Input erwartet.

Contents

List of Figures

List of Tables

Parameter symbols & Abbreviations

B_{str} boost strength.

D_{loc} local area density.

T_{win} window size.

f_s sampling rate.

$g.inhib$ global inhibition.

$n_{celpercol}$ cells per column.

n_{col} column count.

p_{pot} potential percentage.

r_{pot} potential radius.

seg_{max} max segments per cell.

syn_{con} synapse permanence connected.

syn_{dec} synapse inactive decrease.

syn_{inc} synapse active increase.

$tm.syn_{active}$ activation threshold.

$tm.syn_{dec}$ permanence decrease.

$tm.syn_{inc}$ permanence increase.

$tm.syn_{initperm}$ initial permanence.

$tm.syn_{max}$ max synapses per segment.

$tm.syn_{min}$ minimum threshold.

$tm.syn_{new}$ new synapse count.

CAN Control Area Network.

DoS Denial of Service.

ECU Electronic Control Unit.

FPR False Positive Rate.

HTM Hierarchical Temporal Memory.

IDS Intrusion Detection System.

IVN In-Vehicular Network.

MA moving average.

ML Machine Learning.

NAD Network Anomaly Detection.

NADS Network Anomaly Detection System.

NN Neural Network.

NUC Next Unit of Computing.

OTA Over-the-Air.

RDSE Random Distributed Scalar Encoder.

RNN Recurrent Neural Network.

SDN Software-Defined Networking.

SDR Sparse Distributed Representation.

SP Spatial Pooler.

std standard deviation.

TM Temporal Memory.

TPR True Positive Rate.

TSN Time-Sensitive Networking.

TSSDN Time-Sensitive Software-Defined Networking.

V2V Vehicle-to-Vehicle.

1 Introduction

The progression of technology in vehicles has increased the number of connected devices like Electronic Control Units (ECU), entertainment systems, sensors and actors, to provide new features. To connect all those devices, Ethernet-based In-Vehicular Networks (IVNs) appear to be suitable candidates [17]. Ethernet fulfills realtime latency and speed requirements and allows for protocol-independent transmissions. Even non-supported communication standards, such as Control Area Network (CAN), can be gated through the network using specialized hardware or software. Though CAN has been used as the bus for ECU communication for some time now, higher level control and diagnosis by autonomous systems is another reason to shift towards Ethernet. The addition of devices, some of which allow for wireless communication (e.g. for Over-the-Air (OTA) updates or Vehicle-to-Vehicle (V2V) communication), is also adding more attack vectors to the system via various interfaces [20]. Opening up the network this way calls for protection from attackers that could potentially gain access to the network and it's containing devices [20].
Software-Defined Networking (SDN) is lately used in IVNs [8] to allow for precise control over network flows. In combination with Network Anomaly Detection Systems (NADS) it can act as a powerful countermeasure to attacks and other malicious network streams (e.g. caused by defective devices). In addition to pre-defined networking rules enacted in the SDN controller, rules may need to be added or altered upon events that impact the network (e.g. addition of devices or occurence of an anomaly).
Network Anomaly Detection (NAD) can assist the SDN controller in establishing new policies. While a proper network configuration or signature based Intrusion Detection Systems (IDSs) are powerful methods against known attacks, these cannot protect against unknown exploits appearing in the future. Hence nowadays a combination of different security layers are used together, covering the potential blind spots of one another.
By analyzing the network traffic or individual flows of it, it's possible to detect abnormal behaviour and to prevent such traffic from disturbing the network [13]. Reports about network anomalies are sent to the SDN controller so that rules can be changed, poten-

tially blocking the forwarding of undesired Ethernet frames [13].
A variety of NADS already exist [3] and are typically deployed in data centres around the world. An Ethernet-based IVN however is a recent development with a different set of requirements (e.g. realtime communication) and a new set of network topologies. In addition, new Machine Learning (ML) frameworks are developed regularly and with them new potential solutions to the problem appear. This work will focus on exploring the applicability of one of these solutions (Hierarchical Temporal Memory (HTM)) in IVNs.

In the first section (s. section 2) an introduction to the 2 most fundamental topics that shape this work will take place. Here the basics of the NADS (s. 2.3) and the network (s. section 2.2) it is deployed in are explained. This is followed by a description of the problem statement and its relations to other topics and works (s. section 3). Furthermore an analysis (s. section 4) of the given environment is done to assess the required steps for creating a solution to the problem. In there the requirements (s. section 4.3) are declared, a specification of the network (s. section 4.1) is given and also which data sources are available (s. section 4.2).
After that, the design & implementation (s. section 5) of the attempted solution will be presented by explaining the architecture of the system and how it is supposed to satisfy the given requirements.
Tests are done using the proposed solution and the results are then evaluated in order to quantify the performance of the system (s. section 6). This will show how well requirements are satisfied and thus lead to the conclusion (s. section 7) about this work. Finally an outlook of the future is given, discussing further possible developments of the topic and the area in general.

2 Basics

In this chapter the network of the SecVI Demonstrator, the CoRE Group testing environment for IVN related developments, as well as the fundamental principles of the applied ML algorithm HTM will be briefly explained.

2.1 Anomalies

Anomalies can be defined slightly different depending on the data in which they appear. For anomalies within a time series however, two general categories apply: A point anomaly consists of very few data points that differ too much from the rest of the series. An example of this can be seen in figure 2.1 [12]. The second type is collective anomalies, which are built from multiple data points and deviate from the usual sequence for an extended period. Furthermore, anomalies can also be contextual. These express similar features like the rest of the time series but appear in a different context from the rest.
In this work, the detection of anomalies in time series by using common network metrics is the primary intention.

2.2 In-Vehicular Network

The network in which the NADS should be deployed in is an IVN. Traditionally these networks consist of an ever growing number of buses and protocols. Typical members of an IVN are ECUs, which are communicating via CAN messages. Due to the variety of new standards for inter-communication (besides CAN) applied in cars, such as for navigation, management and entertainment systems, Ethernet-based networking is considered as unifying medium in recent times.
In cars however, there is a need for realtime communication when it comes to certain connections. For these connections, both delay and the resulting jitter (s. section 5.1.2)

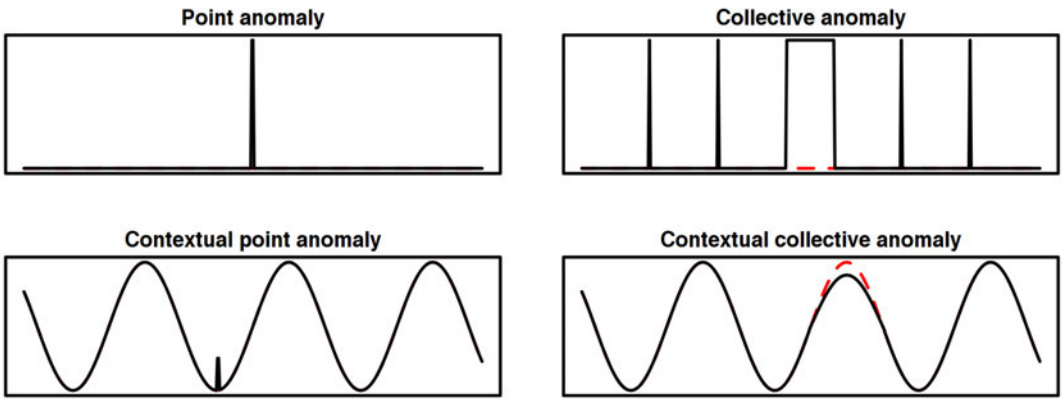

Figure 2.1: General types of anomalies in univariate time series [12].

of transmissions need to behave as expected. NAD can be used to detect violations of this behaviour. To achieve stable properties for the metrics, Time-Sensitive Networking (TSN) can provide the necessary basis. It was initially developed for the purpose of realtime audio/video-streams. TSN mostly consists of expansions to the IEEE 802.1Q standard, a priority-based VLAN networking technology, and is managed by the IEEE's TSN Task group [11]. Although the NAD is aimed towards such networks, it is developed and tested in a network using only strict priority queueing and IEEE 802.1Q VLANs as an approximation of TSN. This is due to the unavailability of switches capable of Time-Sensitive Software-Defined Networking (TSSDN), a combination of TSN and SDN, at the time of establishing the Demonstrator. Despite the current lack of capable hardware, TSSDN has been proven possible and would combine the benefits of both technologies [10].

A number of different flows are available in the provided network. A flow is described as the communication between two specific devices, such as the monitor and the camera. But it can also be narrowed down to only include traffic passing through specific ports to monitor certain applications or protocols. Network flows might be called a stream and vice versa in networking, since their described properties are very similar, if not the same.

2.3 Hierarchical Temporal Memory

HTM is a framework for sequence learning, meaning that it's processing a sequence of inputs to learn relations, and is designed after the way cortical neurons work in the neocortex [9]. It was proven that HTM can work in realtime and that it's producing good results for prediction tasks [14, 6]. Using the predictions it can serve as a base for anomaly detection in streaming data by comparing the input with the corresponding prediction [2]. Starting as a science project to mimic neurological behaviour, HTM developed into a framework with functionality close to other ML solutions. In this work a fork of the original software, called htm.core, is used [5]. It's introducing Python and C++bindings as well as improvements to the codebase, increasing the effectiveness of partial algorithms, computational speed and ease of use.
The basic functionality of HTM is described in Figure 2.2. It mainly consists of 2 parts: the Spatial Pooler (SP) and Temporal Memory (TM). Internally, Sparse Distributed Representations (SDR) are used as the main data structure for processing. An SDR consists of a bit-array, where 0 bits are considered inactive and 1 bits are considered active. SDRs are called sparse because of the low percentage of active bits that effectively represent the data and are called distributed due to the an even distribution inside the array.
The SP creates a SDR of the input, extracting spatial features and also cancelling out noise. The TM learns sequences and makes predictions for future inputs. In combination the algorithms are able to learn and extract spatiotemporal features.

2.3.1 Encoder

In HTM the data is represented by SDRs. Encoding data into a bit-array is necessary for the SP to form a SDR from (s. section 2.3.2). While this process creates a data representation much like an SDR, it might yet be lacking some of its properties (e.g. even distribution) depending on the encoder, which are manifested by the SP afterwards.
For the system to accept input, data needs to be encoded using the following basic principles [15]:
The encoder must create bit-arrays where the active bits represent the semantics of the input in a way that similar data causes an overlap of bits [15]. Furthermore, the output must always have the same size and sparsity (fraction of active bits). Therefore the most important configurable parameters are the number of active bits or percentual sparsity

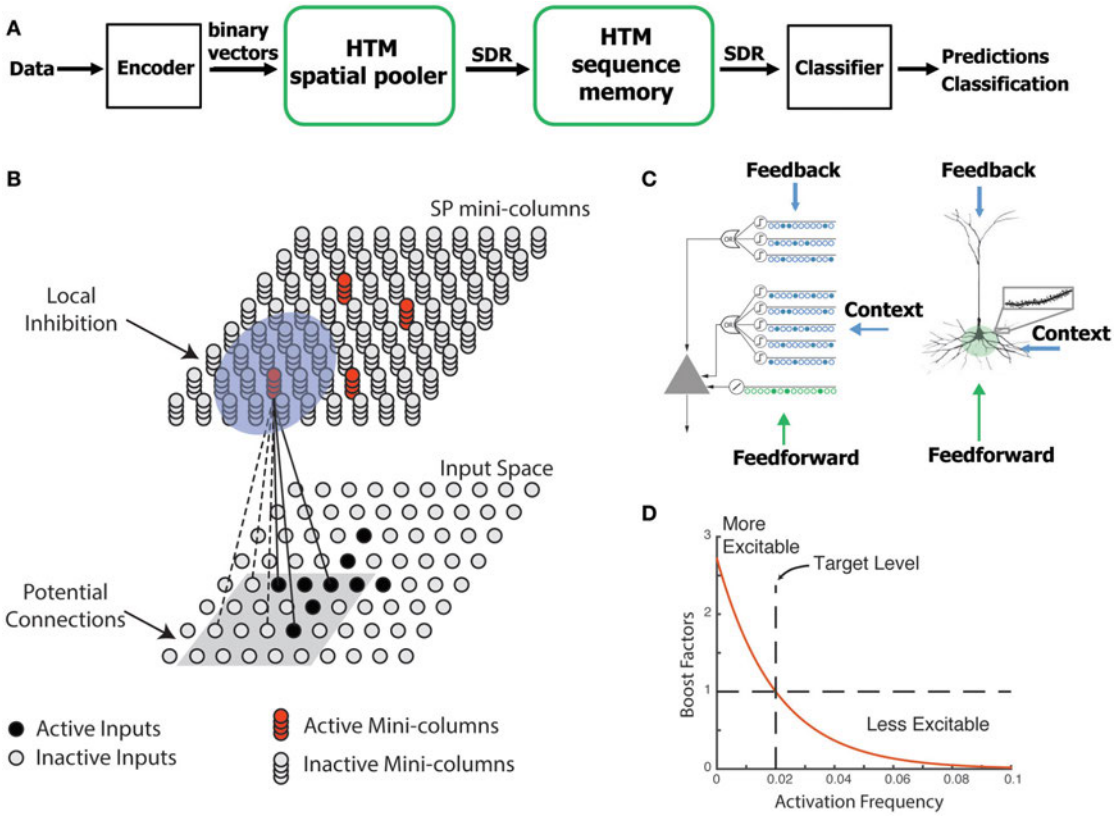

Figure 2.2: Main aspects of HTM Algorithm. **A** Data is encoded to a binary representation by using an encoder. The data is then processed by the SP, which creates a SDR that is fed into the TM (sequence memory). The TM learns and predicts sequences. An anomaly score is produced using the prediction error. **B** The SP converts the binary input into a SDR by forming synaptic connections between each mini-column and their corresponding input space (gray area). Active inputs excite columns and strengthen connection permanences while inactive ones deteriorate connections. The local inhibition (blue circle) causes only the most excited mini-columns to activate within a certain area. [7]

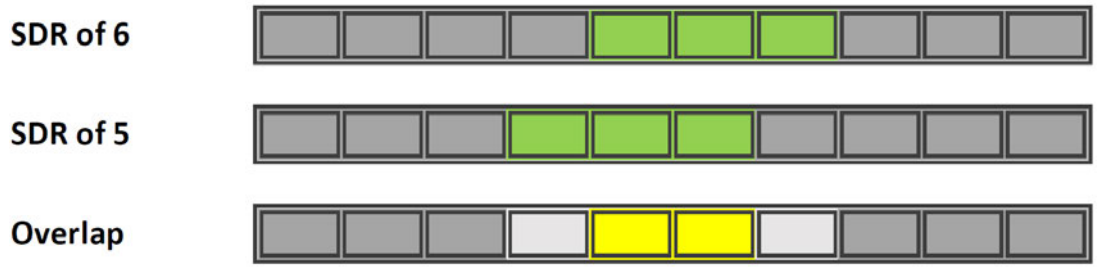

Figure 2.3: Encoding of scalar values into a 10 bit long SDR with a number of 3 active bits and a resolution of 1. Deliberately overlapping nearby values creates a meaning of similarity.

and the total number of bits used for encoding. The goal is to achieve an output similar to that of sensory organs in humans and other animals. For each data type this results in different methods of translation.
First there are scalar values, which is also the type of data this work will use. To get similarity of values to show up on the output, numbers in a narrow range should have overlapping bits in their SDRs. An example of scalar encoding can be seen in figure 2.3. The scalar encoder adds two important parameters: a minimum and maximum value. Values outside of this range can be either discarded or get interpreted into a periodic range. Additionally, an option exists to wrap values on the edge of the range around the beginning and end of the resulting array.
An alternate version of the scalar encoder is the Random Distributed Scalar Encoder (RDSE). Here values are distributed randomly using a hashing function, which also gets rid of the restricted value range in exchange for potential collisions. Which of these encoder types are used and how they're configured will be explained in section 5.1.3.
Other typical data types that can be encoded are dates, images, text, gps coordinates and many more. With more complex data the encoding methods also become more complex. A date for example can be encoded using the months, weeks and days represented as numbers with scalar encoding. These will then be added together by appendation while single parts (e.g. months) can be varied in size to increase or decrease weighting (semantic importance).

2.3.2 Spatial pooler

SDRs have been documented to be common representations of sensory feedback in a neural context [7]. Favorable properties of them include both a distinct representation of items with little to no interference, as well as allowing for a large capacity of patterns [7]. The SP is the key component for the creation of these SDR in a digital format (s. section 2.3).
Spatial pooling is described as following: Spatially similar active bits on the input patterns (encoded bit arrays) are bundled together (pooling). This can be seen in figure 2.2**B**, where the input is represented as a grid of active and inactive bits on the bottom layer and SP columns (above the input layer) are arranged to cover equally distributed regions of it. Here many active bits are condensed to just a few active columns. Just as the input is arranged in a grid-like manner from a bit-array, the same is done in reverse with the SP columns, meaning that a flattened binary representation of the columns creates a bit-array that is the SDR output of the SP. A visualization of the effect of this process is shown in figure 2.4.
Input is processed by the SP by creating connections between columns and input bits. Each connection holds a permanence value describing its' strength between 0 and 1 and is initialized with a random value in a small range around synapse permanence connected (syn_{con}). syn_{con} is a threshold for permanence, above which connections are considered synapses. These are the only connections that can contribute to the activation of a column. Initialized permanences are uniformly distributed across all connections, meaning that 50 % of all connections are synaptic in the beginning and therefore randomly connected to a unique attribute of the input. Eventually, connections between SP columns and input bits within their perceptive field are either boosted and become synapses or are deteriorated over many iterations. This is done by decreasing permanences by synapse inactive decrease (syn_{dec}) for inactive synapses and increasing them by synapse active increase (syn_{inc}) for active synapses but only when a column has been activated. By doing this, a column spezializes for particular inputs and recurring patterns are expressed more precisely and stable in the output.

Other than that, the following parameters contribute to the behaviour of the SP as well: One setting is the size of the SP itself, namely the column count (n_{col}), which describes how many columns are used for mapping the input. For each of these columns the perceptive field is described by the potential radius (r_{pot}), used to calculate a squared pool of inputs with sides of length $2 * r_{pot} + 1$, to which the column has potential connections

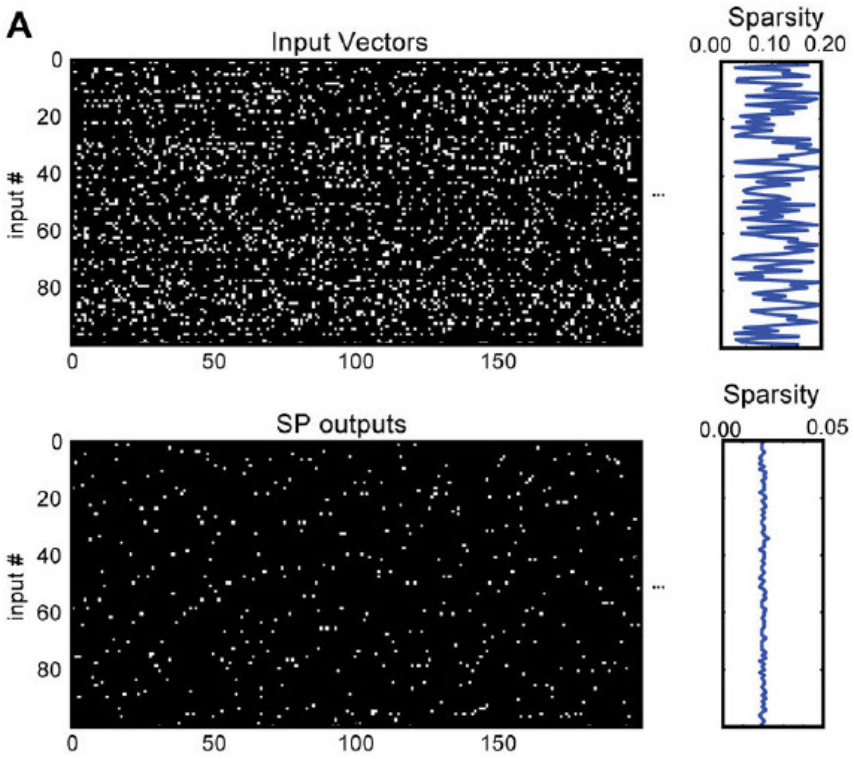

Figure 2.4: Sequence of inputs and outputs of the SP. A single line represents a single SDR where the white dots show active bits. The representation's sparsity is drastically reduced and is even stabilized along the sequence (it has become good practice to encode input with a fixed sparsity regardless). [7]

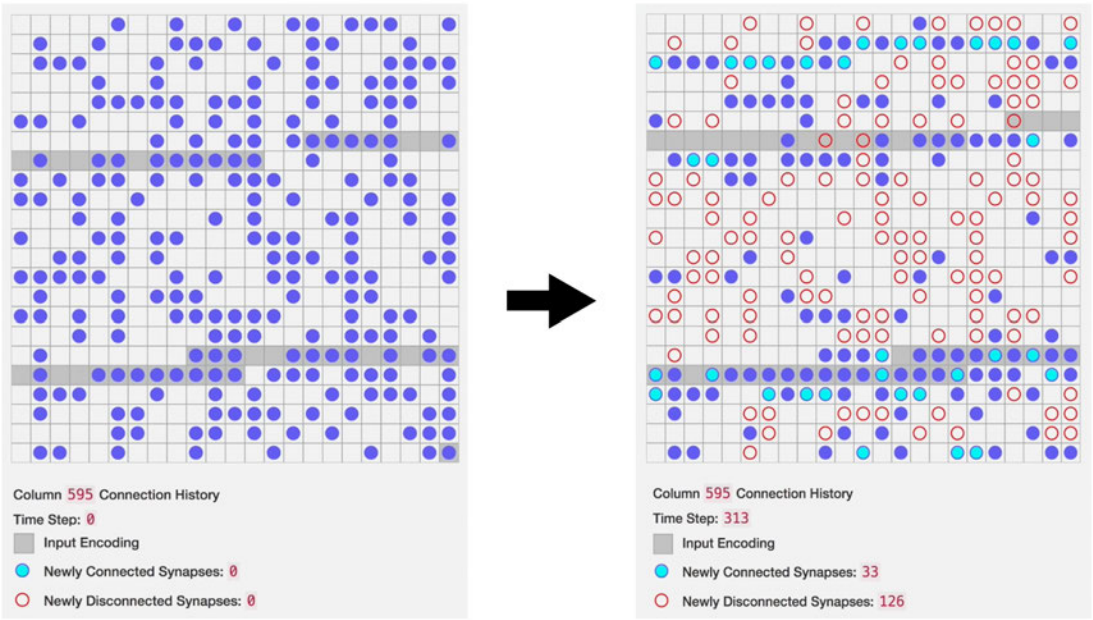

Figure 2.5: Single SP column's input connections before and after learning. Synapses were created and pruned to specialize towards specific input patterns. Example is provided by htm-school-viz[4], an interactive HTM visualization for learning purposes.

(see gray square in figure 2.2 **B**). Within this field only a maximum of potential percentage (p_{pot}) inputs, which are determined during initialization of the SP, can be connected to a column. Thereby the likelihood that each column is connected to a unique set of inputs increases. This is is important since columns usually have an overlapping r_{pot} and neighbouring columns behaving identically is not desirable.

SP columns have two metrics: The overlap score is the number of active inputs on a column's synapses and will be used for selecting the columns to activate. The active duty cycles is a count of the column's activations.
Furthermore there are two mechanisms that prevent overuse and high idle times of columns.
First, inhibition is used to only activate a limited number of columns in a certain area. It is configured by the parameter local area density (D_{loc}), stating the percentage of allowed active columns in an inhibition radius, which is in turn internally calculated using the aforementioned potential connection pools size (see blue circle in figure 2.2 **B**). As an alternative there is also an option to set the inhibition radius to be global. global inhibition ($g.inhib$) lowers execution time significantly but topological information (useful for image processing or multivariate input) is lost. Only the columns with the highest overlap scores (amount of active synapses) in that radius become activated, effectively providing a winner selection as well as creating the desired fixed sparsity.
Second, boosting is promoting columns to prevent inactivity and overuse of frequently winning columns. For each column the overlap score is multiplied by a value either above or below 1 to either increase or decrease the columns chances of activation. The factor of multiplication is determined by the number of active duty cycles in a vicinity, meaning that columns that have been activated very often get their overlap score lowered while their neighbours get it boosted, giving other columns a chance to win the selection as well. Hence similar subsets of input can cause a variety of neighbouring columns to learn from their input instead of overfitting to a single one. To allow for configuration of this behaviour the factor is further multiplied by the boost strength (B_{str}).

2.3.3 Temporal memory

TM is the second algorithm in HTM that's learning from input. As it's name implies, it learns temporal features and is therefore mainly used for prediction and consequently for anomaly detection as well.

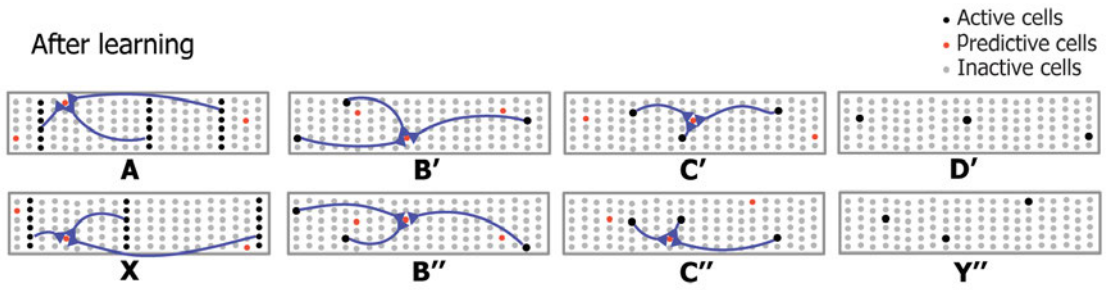

Figure 2.6: HTM high-order memory example. After learning, two sequences with different beginnings but same middle part use different cells of the columns for prediction. This way the correct ending of the sequence can be predicted out of context differentiation.[6]

The SP's SDR output is used as input by the TM for further processing. This time a number of columns, matching the number of input bits (SP's n_{col}), are forming the internal data structure. In the TM, columns are activated simply by active bits in the SP's output.
On top of that, each column contains cells per column ($n_{celpercol}$) cells. The amount of cells is configurable by adjusting $n_{celpercol}$ and will affect the memory capacity for sequences. Multiple cells per column are important for learning a large variety of sequences with different contexts. Additional cells can not only store more information, but also remember the different paths that have led to the current column in the past. This effectively creates a high-order memory by connecting different origins to the current input. An example of this effect can be seen in figure 2.6. Using a single cell for each column creates a single-order memory since the previous cell could have a large number of possible predecessors.

Each cell of a column can hold max segments per cell (seg_{max}) number of dendritic segments, which in return can hold up to max synapses per segment ($tm.syn_{max}$) number of synapses. These synapses will, throughout learning, be created between segments of cells of different columns, which effectively provides the remembrance of sequences in the TM. Just like the synapses of the SP, a permanence describes the connection strength and is initialized with initial permanence ($tm.syn_{initperm}$). Whenever a column is activated, one of its segments will be chosen to learn, depending on the fact if the column was predicted or is bursting (see below). A general rule is that only segments with at least minimum threshold ($tm.syn_{min}$) number of active synapses are eligible for learning. However, on a learning segment, there will be active and inactive synapses, both of which will get their permanences changed by permanence increase ($tm.syn_{inc}$) for active

and permanence decrease ($tm.syn_{dec}$) for inactive ones. In addition, new synapse count ($tm.syn_{new}$) number of synapses will be created and connected to previous active cells, when a segment is learning.
Regarding the main algorithm of the TM, two key methods make up its functionality: predicting and bursting.
A prediction means that for each iteration there are cells in some columns, that are excited into a predictive state. This functionality requires dendritic segments being connected to previously active cells through synapses. Segments will activate when atleast activation threshold ($tm.syn_{active}$) number of synapses connected to it are active. The creation of dendritic segments is part of bursting (see next paragraph). These predecessor cells have been active before and have essentially seen the predicted cells being active afterwards. How this process works can be seen in figure 2.7.
Now, as to what happens when there is no predicted cell in an activated column, either in the beginning or somewhere else in the sequence:
Bursting will activate all cells of a column, whenever it is activated by input but contains no cell in a predictive state. With all cells being active now, a winner cell is selected, which is usally the one with the least segments for even saturation. Dendritic segments are connected to known winner or correctly predicted cells of the previous iteration through new synapses. In figure 2.8 an example is showing how this process happens over 2 iterations of unknown inputs.

An anomaly score is calculated directly by the TM at the end of each iteration and depends on how many columns activate and contain a cell in a predictive state. Calculation of this score is done by dividing the number of bursting columns by the total number of active columns. A high share of correctly predicted columns will create a score close to 0, while a lot of bursting will take it closer to 1.

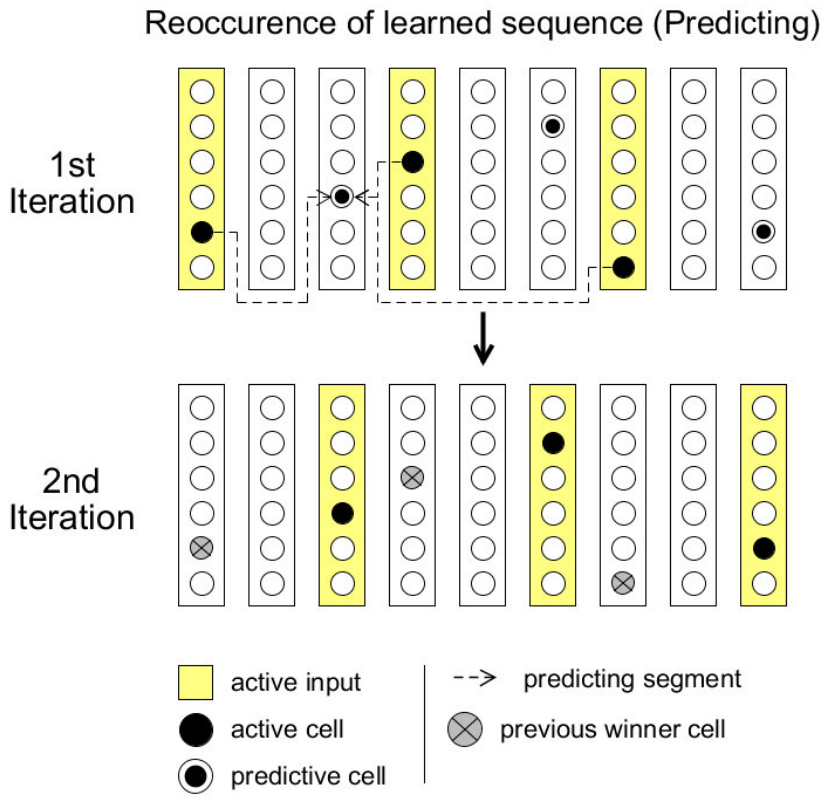

Figure 2.7: TM Predicting: When processing a previously **learned sequence** (see right example), each active cell will activate its synapses to stimulate other cells. Cells that get stimulated by many synapses and thus reach a defined permanence threshold will enter a predictive state. This means that they are part of columns expected to be active next, as they have been previously in the same sequential context. Columns that have been incorrectly predicted will raise the anomaly score. Cells can predict several other cells but for clarity reasons only few segments are highlighted.

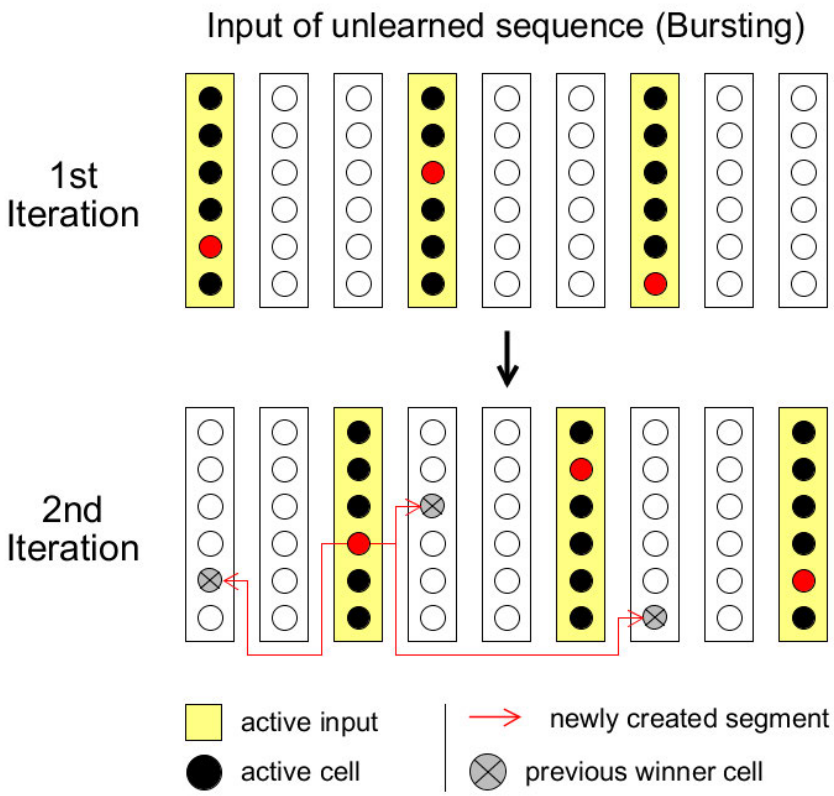

Figure 2.8: TM Bursting: When an **unlearned input** (see left example) is seen active for the first time, bursting takes place and activates all cells within that column. Usually a winner cell is selected and connected to previous active or winner cells (in case of bursting), forming a synapse. Though for the first input no connections can be formed as there are no previous winner cells. Bursting creates many segments but for clarity reasons only few segments are highlighted.

3 Problem Statement & Related Work

Inter-vehicular networks are growing in size and complexity with the addition of new devices and functionality. Like many technological advancements, this growth will eventually lead to new attack vectors and other problem sources. The system of an autonomous or software-assisted car can have a direct impact on the health of any living being in its vicinity, both in the car and in the surroundings of it. That means that the system must rely on safety-critical technologies (e.g. automatic brakes).
These technologies need to be secured so that the safety-aspect is not compromised by external tampering or malfunctional network traffic. Many advancements have been made by the research community working towards this goal, but there is not yet any solution that covers all security aspects. Cybersecurity is usually a never ending race between newly developed security solutions and newly developed workarounds of these. Hence it is crucial to continue developing new systems to ensure the highest possible security that can be achieved at the very moment.
Concerning the security of IVN, there are a number of solutions out there that intend to cover different aspects. One attempt at achieving anomaly detection in IVNs is proposed by Philipp *et al.* [13] and describes ingress control in an IEEE 802.1Q standardized network, which enacts link-layer anomaly detection. IEEE 802.1Qci, an amendment within the aforementioned standard, defines methods for per-stream filtering and policing. In [13] it is shown that proper traffic shaping and prevention of flows that violate the established rules can result in a good reaction to anomalous traffic. Nevertheless it is also demonstrated that by itself this method is not enough since it's showing some false negative results.
Another work related to the security of an IVN is the traffic analysis in V2X application-level gateways proposed by Szancer [18]. By definition it is not a solution targeted at IVN since its main goal is to prevent malicious traffic entering the vehicle by analyzing V2X traffic on the application layer. However, for the IVN to be secure, the traffic entering from the outside needs to be secured as well, as it is entering the IVN after passing the V2X gateway.

Using the same framework as this paper (HTM), Wang *et al.* tried to achieve better security of IVNs. The team is taking a different approach compared to this work in a sense that the analyzed data consists of CAN packet data fields, while prediction and comparison of single fields is used to evaluate abnormality. This means that only CAN traffic is monitored. Meanwhile this paper will base it's data on metrics that are important to the link-layer of an Ethernet network, which can also be measured for gated CAN traffic.
Furthermore, a survey [16] shows that, regarding NAD in connected vehicles, safety is often underrepresented in many works and ideas to countermand attacks are often missing. Therefore the search for the best solutions in terms of delay and reliability is important to eventually reach levels that can be considered safety compliant.
Looking at the works in this area of research, it becomes clear that several aspects of the network need to be secured separately and that a combination of different solutions is critical to cover the system as a whole. As a result it's been determined that this work will investigate the ML framework called HTM, which has not yet been applied as a NADS in this kind of network topology, in order to find out whether it is a suitable addition to the various NAD techniques or not. HTM has, in theory, a high affinity for this particular setting since it is proven to be well-suited for analyzing streaming data [2], especially to find anomalies while also dealing with noise at the same time [14].
Another advantage of HTM is that it's capable of online unsupervised learning. This means that in the case of any changes to the network, e.g. by a software update or alternation of the hardware setup, it is possible to adapt to the changes by allowing it to learn new patterns over a short time, whereas other frameworks such as Recurrent Neural Networks (RNN) would have to be trained beforehand.

4 Analysis

To establish a working NADS it is important to investigate the environment it will be placed in later. Finding out what the key aspects of the network are, as well as what input is available for monitoring is crucial for the development of the system, as it will implicate what requirements may be possible to be achieved. Having this focus will help getting the most out of the detection rates and speed and also point out what kind of anomalies could be detected with the planned system.

4.1 Network of the Demonstrator

The network in which the NADS will be developed is hosted at the HAW Hamburg by the CoRE working group and is called the SecVI Demonstrator. Its internal layout can be seen in figure 4.1. It consists of an SDN capable switch that is managed by a SDN controller. Using VLANs the network is split into a front and rear part with an interconnect. Furthermore the CAN network is split into a left and right portion in both the front and rear part, while each one is connected through a CAN gateway to the main network. Besides the CAN network, on which a recorded playback is being transmitted, there is also a simulated camera in the rear part that is streaming a video to a monitor in the front part of the network. Then there are several Next Units of Computing (NUC) or Raspberry PI 4 based computing units intended for a number of different purposes. Their main purpose is to provide platforms for developing monitoring and defense systems as part of research projects. Traffic passing through port 1 to 12 of the SDN switch is mirrored to NADS1 (core-NUC3) while traffic going through port 13 to 24 is sent to NADS2 (core-NUC4). The traffic arriving at these devices can be captured to create a dataset for learning and testing. For this paper a personal computer will be used to test the proposed NADS on parts of that dataset.
Regarding the technical specifications of the network, as mentioned in section 2.2, it is designed to follow the IEEE 802.1Q standard with the addition of strict priority queueing.

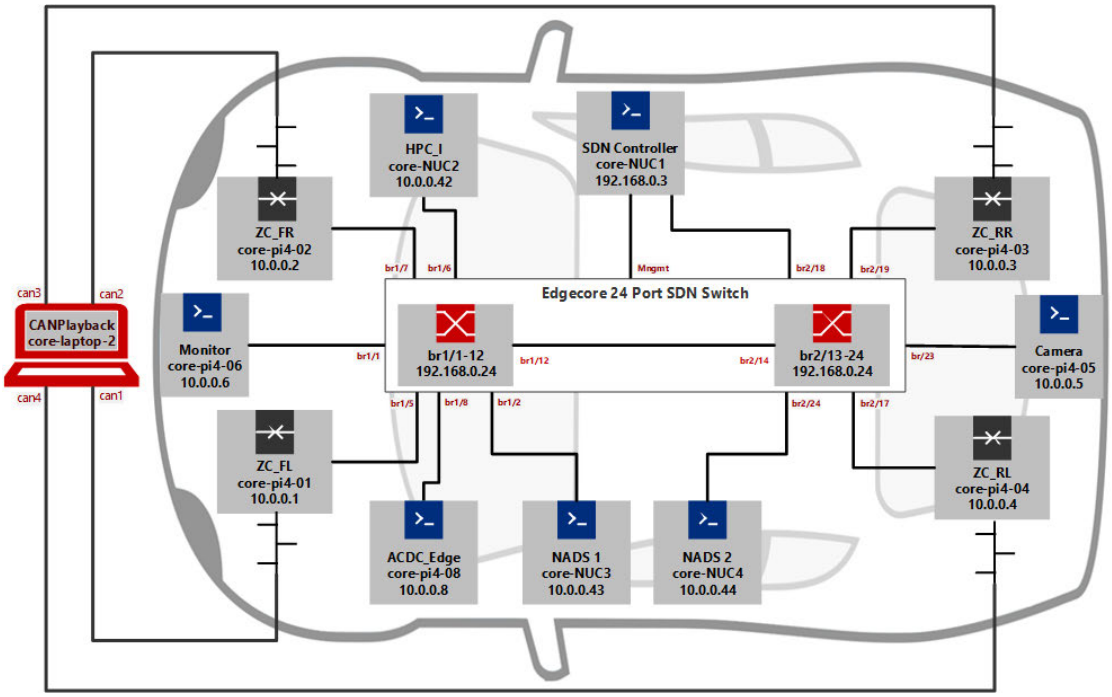

Figure 4.1: Network topology of SecVI Demonstrator.

This will not fulfill the TSN specifications but they have a great number of design rules in common. What this means is that a certain amount of stability in high priority traffic can be expected.
In this work, the flow that will be monitored is made up of transmissions from the camera to the monitor.

4.2 Data sources

Network traffic is mirrored from one half of the car to the NADS responsible for that part of the network. Here it can be preprocessed into a format that will be analyzed later on. Due to the time sensitive nature of the traffic and the advertised abilities of HTM to extract spatiotemporal features from time series, it seems meaningful to transform the traffic into metrics that express these properties. Metrics that count towards carrying such properties are jitter, avg. gap between frames, bandwidth and frequency. An example measurement of these with a resolution of 1 ms can be seen in figure 4.2. Since two of these metrics carry information about the timings (jitter and avg. gap) and the

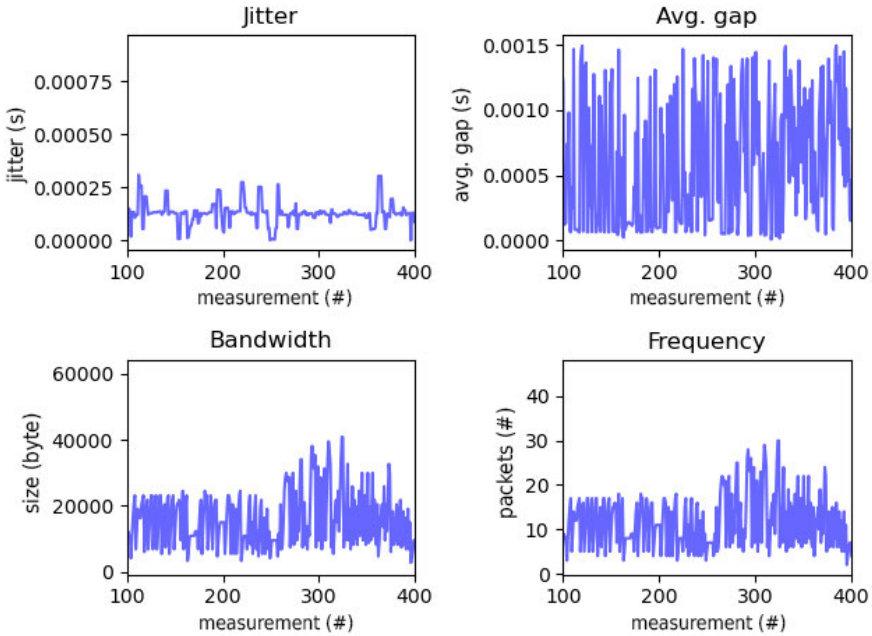

Figure 4.2: Example metric measurement sample of videostream from simulated camera to monitor, sorted by iteration.

other two about the volume of the traffic (bandwidth and frequency), analyzing a combination of those metrics in unison could provide a great method of finding correlations and thereby detecting violations of those as well. This is called cross-validation and it can also increase the certainty of a result in some situations.

4.3 Requirements

As this application will not be part of any functional system in the car, but is mainly purposed for monitoring, it doesn't need to meet many requirements. A lot of common network participators of an IVN need to fulfill either safety- or realtime-related requirements. At the very least these must not be disturbed by this application, which is mostly ensured by mirroring the traffic to a separate device for analysis.
Nevertheless, for the NADS to be practical, the following requirements have been defined with limitations and possibilities in mind:

- Detection should be as close to realtime as possible in order to support rapid countermeasures.

- Detection should identify anomalous behaviour without seeing it beforehand in order to be applicable to unknown attacks.
- The system should not need to rely on human operators while running, making it a better fit for autonomous cars.

5 Design & Implementation

A foundation of the system can now be laid out after analyzing the environment it will be used in. During the design phase there are a few things to consider in order to make the system capable. The main aspect that would be advisable to achieve next to good detection rates is to get timings as low as possible. Having a NADS react as fast as possible in a TSN is a given considering the impact, a flawed security potentially has in a safety-critical situation. This means that parallelization of time consuming work should be done where it's possible, next to optimizing the code for high efficiency and avoiding unnecessary operations. Therefore the applications is split up into 4 asynchronously communicating components.

5.1 Components

Components are divided by simple tasks that need to be executed in the workflow:
In figure 5.1, each component and their interface to the next one is displayed as a data flow diagram. The Traffic capture component (s. section 5.1.1) provides packets to the Metric Preprocessor (s. section 5.1.2), where jitter, avg. gap, bandwidth and frequency are calculated. This data is handed over to the HTM algorithm as a tuple, where it is encoded, pooled and is then finally processed by the TM (s. section 5.1.3). As the final step, the anomaly score is given to the anomaly reporter, where a decision is made whether an anomaly report should be sent or not. Each component has its own thread and is buffering incoming data using thread-safe queues, while HTM internally executes 3 sequential processing steps in a single thread. All parameters for the components that have and will be mentioned are declared in a single configuration file (s. section 5.1.6).

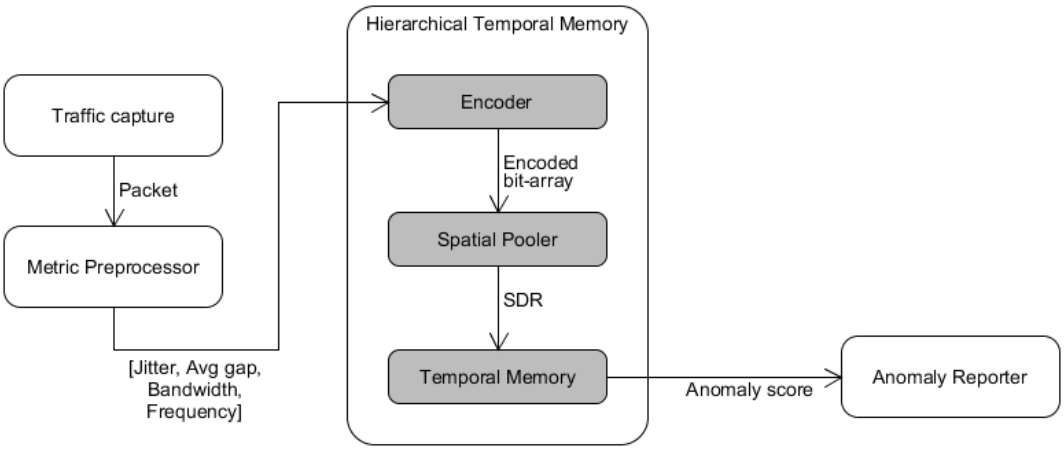

Figure 5.1: The systems data flow diagram, also showing the participating components. HTM is a single component consisting of 3 parts.

5.1.1 Traffic capture

Traffic capturing is done by using the python library `pyshark`, which is a python wrapper for `tshark`, a common network sniffing application. The component is providing the traffic to other components by implementing a simple callback for each captured packet. In case only a certain connection should be monitored, there is an option to pass a capture filter as an argument. By doing that, streams between devices can be analyzed separately for better results and better identification of the problem source.

5.1.2 Metric preprocessor

The preprocessor is preparing the metrics for the HTM algorithm. On each packet it is calling methods that calculate bandwidth, frequency, jitter and the average gap between packets. Packets that have been captured contain a lot of information, including arrival time, length (size), headers, payloads and many more. For calculation of above mentioned metrics, just two data fields are of importance:
Arrival time, which is given in seconds and is including decimals, reaching a precision of 100ns on our machines. Length, which is the total packet size including all headers, given in bytes.
A window size (T_{win}) of choice (e.g. 1, 10 or 30 ms) is used for metric calculation periods. Both a rolling window with a configurable sampling rate (f_s) and a static window have

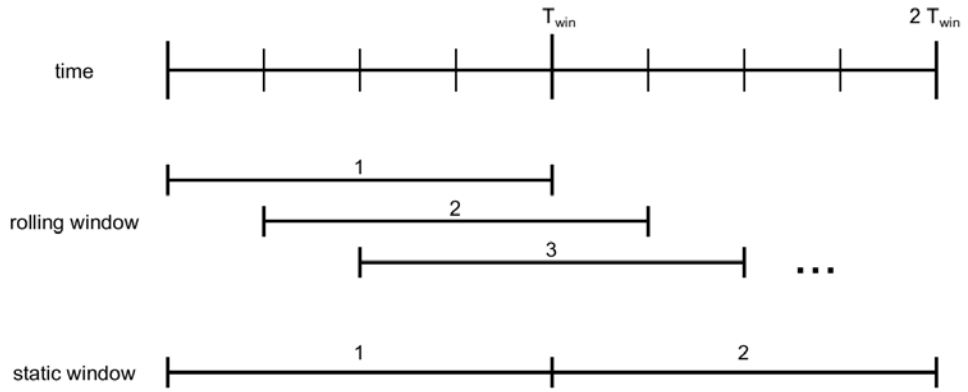

Figure 5.2: Rolling window with f_s= 4 and static window

been implemented to compare their effectiveness in observing such periods (s. figure 5.2 for window type reference). While a static window provides a fixed deadline where data can be discarded, the rolling window had to be implemented using a queue. This queue holds pairs of data with timestamps, is being filled with incoming data and gets it's oldest entries outside of the measurement period removed regularly.

During these periods, each packet size is added up, resulting in the bandwidth:

$$x_{bw} = \sum_{n=1}^{m} size_n$$

where m is the number of packets and thereby the packet frequency.
Avg. gap and jitter are metrics that depend on the time between incoming packets. By adding up all inter-packet delay and dividing it by the number of packets, the avg. gap is calculated:

$$x_{a.gap} = (\sum_{n=1}^{m} delay_n)/m$$

Lastly, jitter is calculated by the following method, using the minimum and maximum delay that occured:

$$x_{jit} = delay_{max} - delay_{min}$$

and is describing the variance of the delays during the period.
Once calculated, the four values are pushed into a thread-safe queue as an Array, which will be accessed by the HTM component next.

5.1.3 Hierarchical temporal memory

Aside from configuration (s. section 2.3), HTM is fairly easy to use. First of all the encoder, SP and TM are initialized using a set of parameters during startup.
During processing, the encoder prepares the data of each iteration, followed by calling a compute method on the SP, creating an SDR of the encoded data. Lastly, another compute method is called on the TM with this SDR. With that an anomaly score is produced by the TM (s. section 2.3.3), which can now be used for further processing (s. section 5.1.3). A quick breakdown of this process is displayed in figure 5.1.

Anomaly window

An Anomaly window will be established as an interpretation of the anomaly score. This is done because the anomaly score is naturally noisy, especially when using a high resolution for metric sampling. By applying a smoothing technique, such as using a moving average (MA) of the score and adding a threshold, it could be possible to provide some noise reduction. A major downside of doing so is an increase in delay and also potential false negatives. Hence two variants for windows, using either a simple threshold or a combination with MA, are implemented and compared against each other to determine a reliable method.
Once the anomaly score crosses above the threshold, an anomaly window will begin for a configured duration. The state of the observed flow counts as anomalous during an active window. In addition, occurences of scores above threshold during this state will prolong the window for another full duration.
An implementation of this higher-level analysis is a method to counter unwanted behaviour due to the use of high resolutions, its effects will be explored in section 6.4. In there, both threshold and minimum duration will be tested for reasonable configurations. The general goal is to reduce false positives and to cover whole durations of anomalies to provide a stable anomaly state.
The calculation and decision making for anomaly windows is done by the same thread that's doing the HTM calculations because the result needs to be available before the next iteration (s. section 5.1.4).

Encoder

In this work, HTM input consists of scalar values. Therefore the scalar encoders will be used going forward. Though, as described in section 2.3.1, there are two methods of scalar encoding available: default encoding using a limited value range and RDSE.
RDSE uses MurmurHash3, a widely used non-cryptographic hashing function for lookups, to map each value to a number of buckets equal to the number of active bits for representation (s. algorithm 1). Both encoder types have been implemented to monitor the impact on the results.

Algorithm 1 RDSE mapping

```
output = bit-array[size]
index = int(input * resolution) // starting index based off of input without fractionals
for offset=0,1,2,...,activeBits do
    hash_buffer = index + offset      ▷ map similar values until a number of buckets equal to the amount of activeBits has been found
    bucket = MurmurHash3(hash_buffer, seed)
    bucket = bucket % size // move buckets to available range
    output[bucket] = 1 // use bucket as index for activated bit
end for
```

Before encoding, it might provide an advantage to normalize input values depending on their behaviour (e.g. high variance). A normalizer class has been added for such purposes, which provides the variables necessary for normalization. It can be configured to simply provide a maximum value to divide the input by, but is also able to take input values over a time period to calculate standard deviation (std), mean or max. The latter method can either finish after observing that period or continue updating with after an interval of choice.
This means that the following basic normalization options can be used:

$$x_{norm} = \frac{x}{x_{max}} \quad \text{and} \quad x_{norm} = \frac{x - mean}{std}$$

5.1.4 SP & TM

As mentioned in section 5.1.3, the SP & TM are easy to use and implement. For each of them there is a parameter class, that must be populated with the settings mentioned in section 2.3.2 and 2.3.3. A single exception is that the column dimensions of the TM are not configured separately and must be the same as the SP (s. section 2.3.3 for reasoning).

Then both SP & TM are created by calling their constructors and passing their respective parameter class instances. Though a great number of single task methods are provided, both components have a compute method which combines the typical processing steps for HTM. These methods take a boolean, enabling or disabling learning, as well as the input (encoder output for SP & SP output for TM) as parameters.
One more notable built-in feature of HTM is storing and loading SP & TM as a whole, allowing for persistence of learned structures and also pre-learning and distribution of the model.

However, the high resolutions used in this work are causing a large number of iterations, more than is seen in previous HTM application examples. This led to a potential improvement for this system, which is the temporary blocking of learning in the TM during an anomaly window. A Denial of Service (DoS) attack for example can last a few seconds atleast, spanning over thousands of iterations which would be learned as reoccuring sequences by the TM. The effect of this behaviour and its countermethod will be explored in section 6. Note that this is only important for the TM since the SP is only learning and providing representations of incoming data. The technique would also not apply to the first anomalous input which would lower it's effect on systems using longer measurement periods and therefore less iterations overall.

5.1.5 Anomaly reporter

When an anomaly has been detected, it needs to be reported by the anomaly reporter. During an anomaly window, a packet is broadcasted regularly on the network with the following contents:

- Destination MAC (broadcast): FF:FF:FF:FF:FF:FF
- EtherType = 0xFFAD
- Payload:
  - Code (64 bit): 1 (Anomalous behaviour) or 3 (Heartbeat)
  - Sequence number (64 bit): Number of report
  - Report ID (String - UTF8): "AnomalyHTM_", followed by an instance name

In addition the reporter will send a heartbeat every 10 seconds. This lightweight protocol has been defined for anomaly reporting in the SecVI Demonstrator network and can be recognized by the SDN controller.
When such broadcasts arrive at the SDN controller, network flows could be reconfigured to block or reroute specific traffic in response to the anomaly. Furthermore, by using unique instance names for the NADS, problem sources (devices or flows) can be narrowed down efficiently.

5.1.6 Run configuration

An instance of this NADS is set up using a configuration file with a Dictionary, containing all parameters for the various components. An example of it can be seen in listing 5.1, including a few comments about important settings. For an explanation of SP & TM configuration values refer to section 2.3.2 and 2.3.3.

Listing 5.1: Run configuration for the NADS

```
"enc": {    # All encoders listed in this part will be auto. created
    "jitter": {     # Each encoder has a meta and param section
        "meta": {   # In meta various top-level settings are described
            "type": "SCALAR", "col": "jitter", "normalize": False,
            "store": True       # Option to store measurements for tests
        },
        "param": {  # settings directly used in encoder initialization
            "size": 300, "sparsity": 0.1, "minimum": 0.0, "maximum": 0.0015
        }
    },
    "bandwidth": {
        "meta": {
            "type": "RDSE", "col": "bandwidth", "normalize": True,
            "constant_max": 500, "store": True
        },
        "param": {
            "resolution": 0.1, "size": 300, "sparsity": 0.1
        }
    }
},
"application": {
    "mode": "SIM_LIVE",     # Simulated live analysis (by loading a pcap)
    "model": "",            # Option to reload a previously trained model
    "resolution_shift": 3,  # Resolution configured by decimal shifting and
```

```
    "resolution_mult": 1.5, # multiplication, T_win = 1s*1.5/1000 = 1.5ms
    "max_polling": 2,       # Max sampling frequency f_s for rolling windows
    "window_metrics": False,    # True = static window, False = rolling window
    "l_iterations": 800,    # Learning time in seconds
    "l_iterations_min": 140,    # Enforced startup learning time
    "stop": 801,            # Finishing time
    "seconds": True,        # Base calculations on seconds, not iterations
    "interface": "",        # Interface to monitor during live tests
    "pcap": "networklog_medium_video.pcap", # Pcap to load for simulation
    "report_interface": "Ethernet",     # Interface to report anomalies on
    "learn": True,          # True = learning, False = inference only
    "always_learn": False,  # Enforce learning during anomaly windows
    "padding": False,       # Fill gaps occuring in measurement with 0 values
    "anom_thresh": 0.005,   # MA threshold for anom window
    "anom_thresh_max": 0.1,     # SCORE threshold, for immediate anom window
    "num_above_thresh": 1500    # Anom window duration in measurement periods
},
"sp": {"boostStrength": 1.5,            # B_str
        "columnCount": 128,             # n_col
        "localAreaDensity": 0.4,        # D_loc
        "potentialRadius": 32,          # r_pot
        "potentialPct": 0.5,            # p_pot
        "globalInhibition": False,      # g.inhib
        "synPermActiveInc": 0.05,       # syn_inc
        "synPermConnected": 0.1,        # syn_con
        "synPermInactiveDec": 0.008},   # syn_dec
"tm": {"activationThreshold": 5,        # tm.syn_active
        "cellsPerColumn": 2,            # n_celpercol
        "initialPerm": 0.21,            # tm.syn_initperm
        "maxSegmentsPerCell": 126,      # seg_max
        "maxSynapsesPerSegment": 126,   # tm.syn_max
        "minThreshold": 3,              # tm.syn_min
        "newSynapseCount": 64,          # tm.syn_new
        "permanenceDec": 0.1,           # tm.syn_dec
        "permanenceInc": 0.1}           # tm.syn_inc
```

6 Evaluation

In this chapter the evaluation of the most important aspects of the system will take place. This includes a look at the detection rates of the algorithm, the temporal behaviour and also the tuning of HTM parameters. For proper evaluation, anomaly types need to be defined and recreated in test runs if possible. Due the high impact on detection results of many settings for input and HTM itself, focus of this evaluation will not be to test as many scenarios as possible, but rather to increase the performance of the system. Additionally, the lack of real-world data sets of a yet emerging environment, namely in-car ethernet networking, provides another challenge for finding suitable test scenarios. Therefore, the main part of the evaluation will be the exposure of strengths and weaknesses of the algorithm in our given setting, as well as the trade-off measures that needed to be taken for better results.

6.1 Anomalies

Concerning the anomalies it needs to determined what types the system will likely detect and which ones are likely to be overlooked. The main factor for distinguishment is the impact each type has on the input of this system. For this system, 4 metrics are considered (s. section 4.2) to provide the data for anomaly detection: jitter, avg. gap between frames, bandwidth and frequency. By monitoring these metrics, the system is performing link-layer anomaly detection. This means that it inherits the typical strengths and weaknesses that come with this category.
Based on this, the following causes could be expected to create detectable anomalies:

- DoS attacks cause a packet flood with the intention to disrupt the communication between devices by exceeding hardware capacities. An attack of this type has a high chance of impacting all of the above metrics.

- Man-in-the-middle attacks can, depending on the type, introduce additional delay when altering packets.
- There is also a chance that a device will malfunction in any way and cause irregular network flow. One example is when a device is throttling performance due to overheating or a process is taking up all resources of a device. Another example could be a defect sensor sending malformed packets.

On the other hand there are anomaly sources that might leave little to no traces in the above metrics instead:

- Application-level attacks that change the content of the communication but don't alter it's behaviour.
- Shutting down the communication of a participant of a flow (metrics are only collected on active communication).
- Many other anomalies can occur that can only be detected by different input.

It's common for anomaly detection systems to specialize for certain features, as trying to cover too many areas at once lowers success rates and increases false positives. Instead, weaknesses can be covered by combining multiple detection systems and techniques (e.g. Application-level gateway[18], Watchdog etc.), each specialized for different anomaly types.

6.2 Testing methodology

In order to properly measure the performance of the system, a testing environment with reproducable properties is necessary. Rather than running the same configuration on the SecVI Demonstrator and risking deviations of behaviour of some sort, testing will be done on recordings of the traffic. This way the scenario will stay the same over many tests, leaving only randomness inside the HTM algorithm (e.g. random SP initialization) as factors that could alter the behaviour of the NADS between runs.
Fortunately `pyshark` provides all required tools: instead of live processing it's also possible to store the captured traffic into a file and load it again later. Though a replay doesn't provide the data in the same speed as in reality (usually it's faster), the arrival time of each packet can be used to perform time-based calculations like metric calculation and duration of anomaly windows. Hence setting up a test case consists of recording the

flow that is to be analyzed, launching a simulated attack (such as DoS) and storing the capture file for later use.
Time is measured from the beginning of a capture. During tests and captures no countermeasures will be taken, as it is not part of this work. In order to validate the results, each parameter set presented in this work has been subjected to at least 10 test runs.
Finally, test runs have been performed on a computer running Windows 10, using the following components:

- CPU: AMD Ryzen 3600
- RAM: 32 GB DDR4-3200
- GPU: AMD Radeon RX6800
- Mainboard: MSI MS-7B86
- HDD: Western Digital WDS100T2B0C-00PXH0

For this NADS's performance however, only the CPU and RAM have any impact, with the only exceptions being edge cases, such as insufficient RAM and paging as a result.

6.2.1 Test scenario: Videostream DoS-attack

Communication between the camera and monitor of the Demonstrator (s. figure 4.1) will serve as a base for the first test. The camera sends a continuous stream of video data to the monitor.
The traffic of this network flow has been captured for a total duration of little over 800 seconds or 13:20 minutes. During the capture, three DoS attacks are launched from another device in the network at different points in time to serve as anomaly sources. Target of the attacks is the monitor, while the malicious packets contain a spoofed IP-address which belongs to the camera. Spoofing the IP address is important to actually inject the packets, since the SDN controller will already block unknown routes between devices.
The exact beginning and ending time of the attacks, according to arrival of the first and last DoS packets, are as follows:

- Attack 1 starts at 176.794 s, ends at 209.466 s, lasting 32.672 s.
- Attack 2 starts at 542.717 s, ends at 580.432 s, lasting 37.715 s.

- Attack 3 starts at 654.861 s, ends at 687.517 s, lasting 32.656 s.

In figure 6.1 all four input values are displayed in temporal arrangement. Here measurements are taken over a period of 100 ms using a *rolling* window, which is sampled every 5 ms during continuous packet flow.
Figure 6.2 contains the same measurement but with a *static* window of length 1.5 ms. This highlights key differences between the window types in a sense that the expression of flow behaviour in the metrics varies greatly, depending on the choice of measurement configuration.
Therefore both input methods will be investigated in their detection efficiency and other performance aspects in the following paragraphs. Noticable consistency in measurements can be seen during the attacks on all metrics except jitter, while normal behaviour appears volatile in comparison. In theory, HTM should be able to adapt to the high variance of the flow and be raising the anomaly score when it's missing.

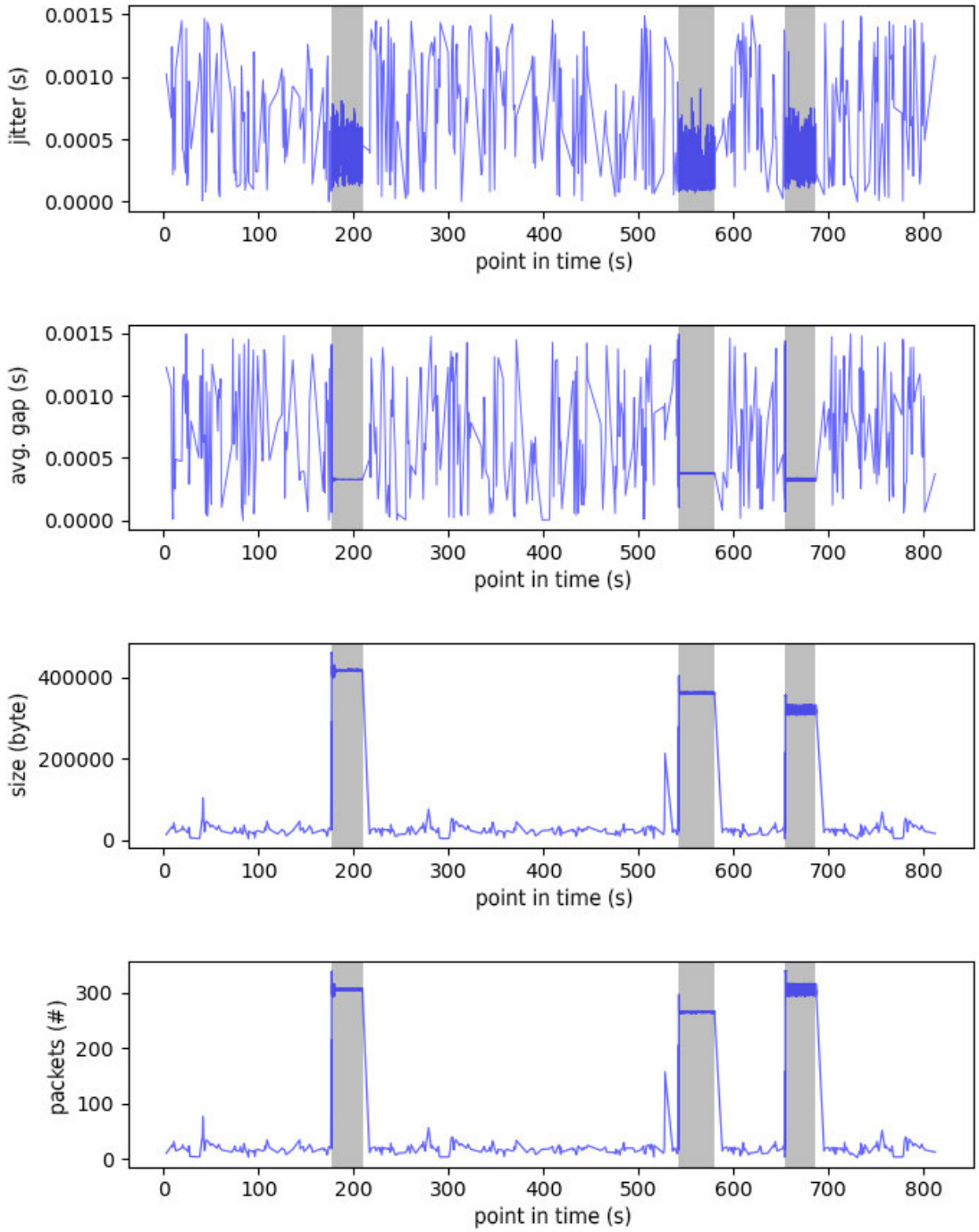

Figure 6.1: Input Measurements for Test: Videostream DoS-attack. The following metrics are shown from top to bottom: jitter, avg. gap, bandwidth and frequency. One measurement consists of a maximum of 100 ms of data, using a *rolling* window with $f_s = 20$. Attack-related measurements have a gray shaded background.

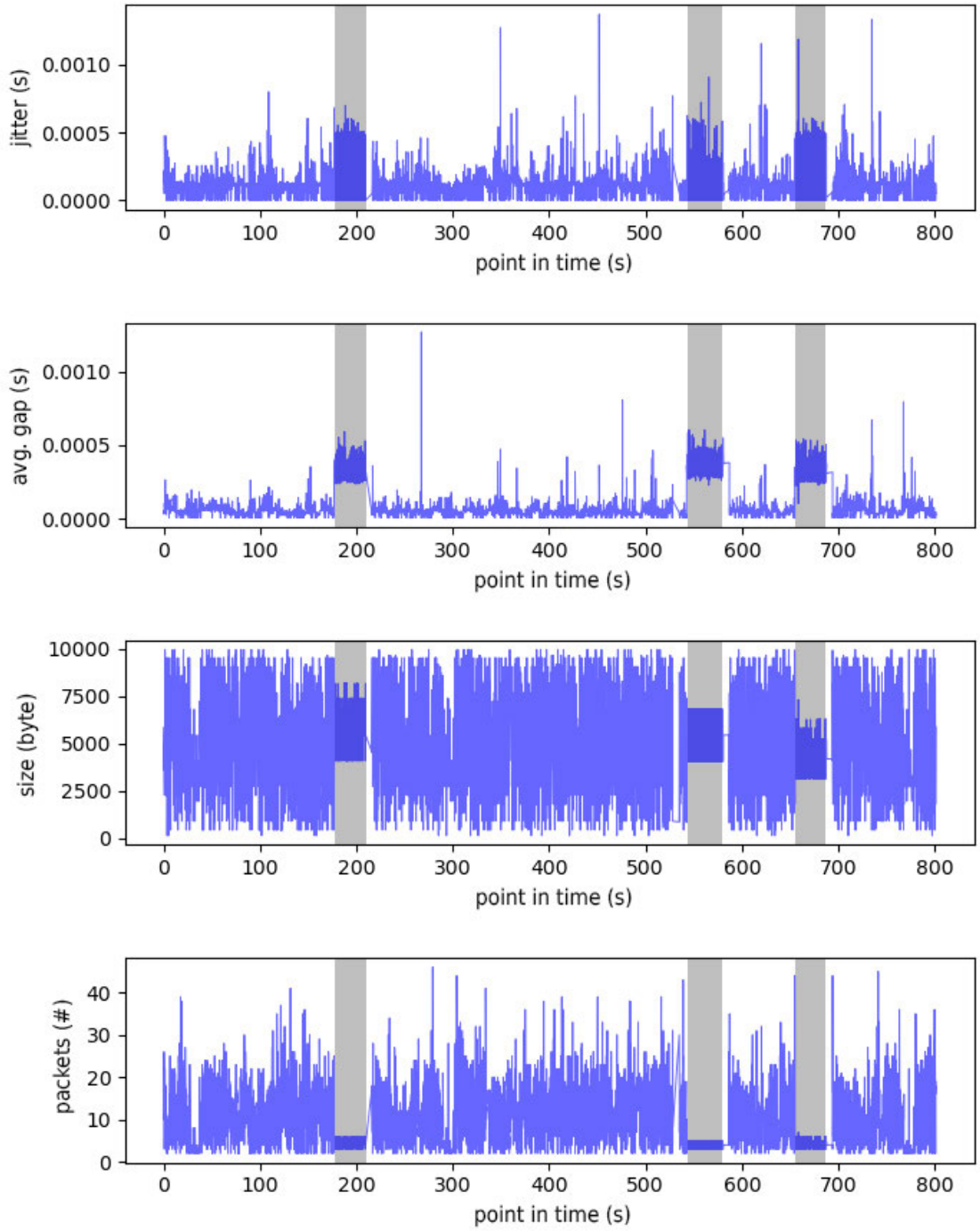

Figure 6.2: Input Measurements for Test: Videostream DoS-attack. The following metrics are shown from top to bottom: jitter, avg. gap, bandwidth and frequency. One measurement consists of a maximum of 1.5 ms of data, using a *static* window. Attack-related measurements have a gray shaded background.

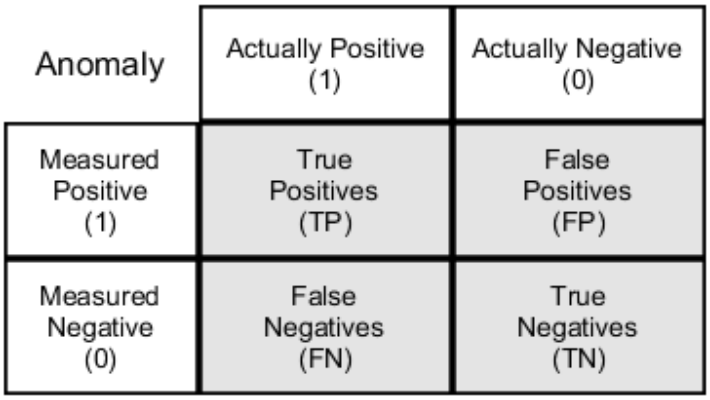

Figure 6.3: Confusion matrix based on actual and measured anomalies (anomaly window).

6.3 Detection rates

Anomaly detection systems are evaluated by their detection performance, which is typically measured by the True Positive Rate (TPR), False Positive Rate (FPR) or other performance metrics. In general, the calculation of these rates is a comparison between actual anomalies and the detection on their respective input. Anomaly windows have been chosen as an interpretation of the anomaly score produced by the TM and will therefore provide the base for detection rates in this work.

Out of the available performance metrics, the following methods have been chosen for this work's evaluation and are based on the confusion matrix in figure 6.3:

- Coverage quality during anomalies: TPR, defined by the total time of positive anomaly detection during anomaly occurence:

$$TPR = \frac{TP}{TP + FN}$$

- False positive detection outside of expected anomalies: FPR, defined by the total time of positive anomaly detection during normal behaviour.

$$FPR = \frac{FP}{FP + TN}$$

SP parameters	value
B_{str}	**3**
n_{col}	128
D_{loc}	0.4
r_{pot}	64
p_{pot}	0.2
syn_{inc}	0.12
syn_{con}	0.2
syn_{dec}	0.008
$g.inhib$	**true**

TM parameters	value
$n_{celpercol}$	6
seg_{max}	256
$tm.syn_{active}$	10
$tm.syn_{initperm}$	0.21
$tm.syn_{max}$	256
$tm.syn_{min}$	8
$tm.syn_{new}$	128
$tm.syn_{dec}$	0.1
$tm.syn_{inc}$	0.1

Jitter encoding	value
Type	Scalar
Normalization	none
Max.	0.0015
Min.	0.0
Size	300
Sparsity	0.1
Periodic	false

Application parameters	value
Window	**rolling**
Sampling resolution	**100ms (5ms)**
Initial learning time	50 s
Learn during anomaly	true
Padding	false
Anomaly threshold	0.1
Anom. window duration	**2s**

Table 6.1: Optimized configuration for videostream using a *rolling* window. Only jitter is used for input. Values that directly differ from the other method are displayed in **bold**.

6.3.1 Detection results

Now following are the detection results and configurations for the test scenario. Following a number of reconfigurations, the parameters of table 6.1 and 6.2 have provided the best detection results. In section 6.7, important decision-making steps for choosing the parameters will be further evaluated.

Two possible solutions are provided, with the key difference being that one is using a *rolling* window (s. table 6.1), while the other one is using a *static* window (s. table 6.2) for metric observation and collection. In order to produce satisfying detection results, the chosen input of the *static* window consists of all 4 presented network metrics. Meanwhile the *rolling* window method relies only on jitter. Due to the choice of a multivariate input for the *static* window, the performance boosting *g.inhib* setting had to be turned off. The actual detection results of the test, including the anomaly score and anomaly window (1=active, 0=inactive), are displayed in figure 6.4 (*rolling* window, for related input see jitter in figure 6.1) and 6.5 (*static* window, for related input see figure 6.2). So

SP parameters	value
B_{str}	**1.5**
n_{col}	128
D_{loc}	0.4
r_{pot}	64
p_{pot}	0.2
syn_{inc}	0.12
syn_{con}	0.2
syn_{dec}	0.008
$g.inhib$	**false**

TM parameters	value
$n_{celpercol}$	6
seg_{max}	256
$tm.syn_{active}$	10
$tm.syn_{initperm}$	0.21
$tm.syn_{max}$	256
$tm.syn_{min}$	8
$tm.syn_{new}$	128
$tm.syn_{dec}$	0.1
$tm.syn_{inc}$	0.1

Jitter encoding	value
Type	Scalar
Normalization	none
Max.	0.0015
Min.	0.0
Size	300
Sparsity	0.1
Periodic	false

Avg. gap encoding	value
Type	Scalar
Normalization	none
Max.	0.0015
Min.	0.0
Size	300
Sparsity	0.1
Periodic	false

Bandwidth encoding	value
Type	Scalar
Normalization	none
Max.	10000
Min.	0
Size	300
Sparsity	0.1
Periodic	false

Frequency encoding	value
Type	Scalar
Normalization	none
Max.	50
Min.	0
Size	300
Sparsity	0.1
Periodic	false

Application parameters	value
Window	**static**
Sampling resolution	**1.5ms**
Initial learning time	50 s
Learn during anomaly	false
Padding	false
Anomaly threshold	0.1
Anom. window duration	**4.5s**

Table 6.2: Optimized configuration for videostream using a *static* window. All 4 available metrics are used as input. Values that directly differ from the other method are displayed in **bold**.

far, all attacks have been properly detected, while the static window method is showing some difficulties at establishing a continuous detection of the 2nd attack.

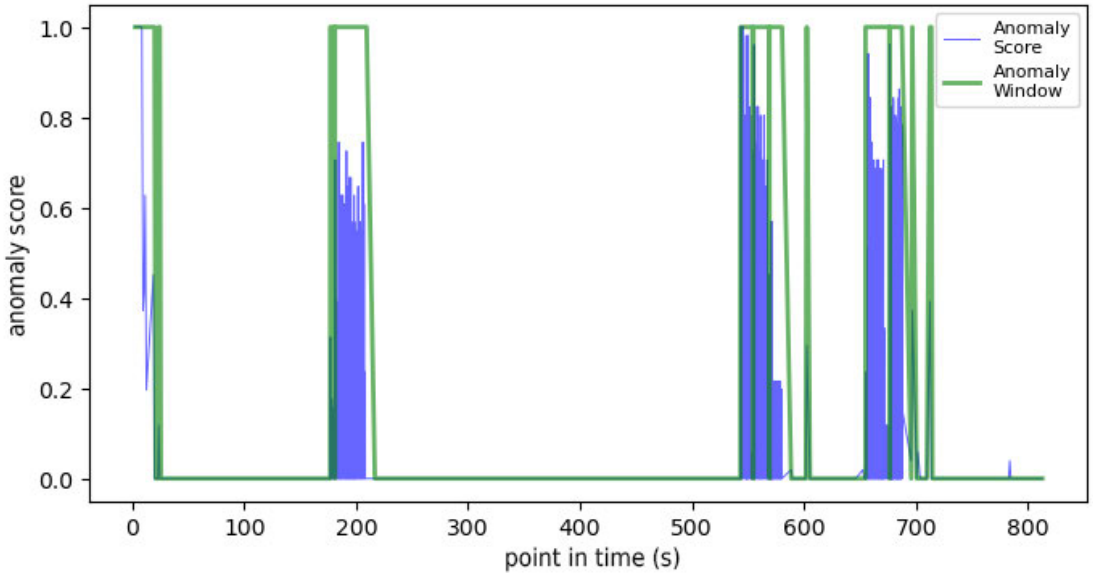

Figure 6.4: Detection results using a *rolling* window.

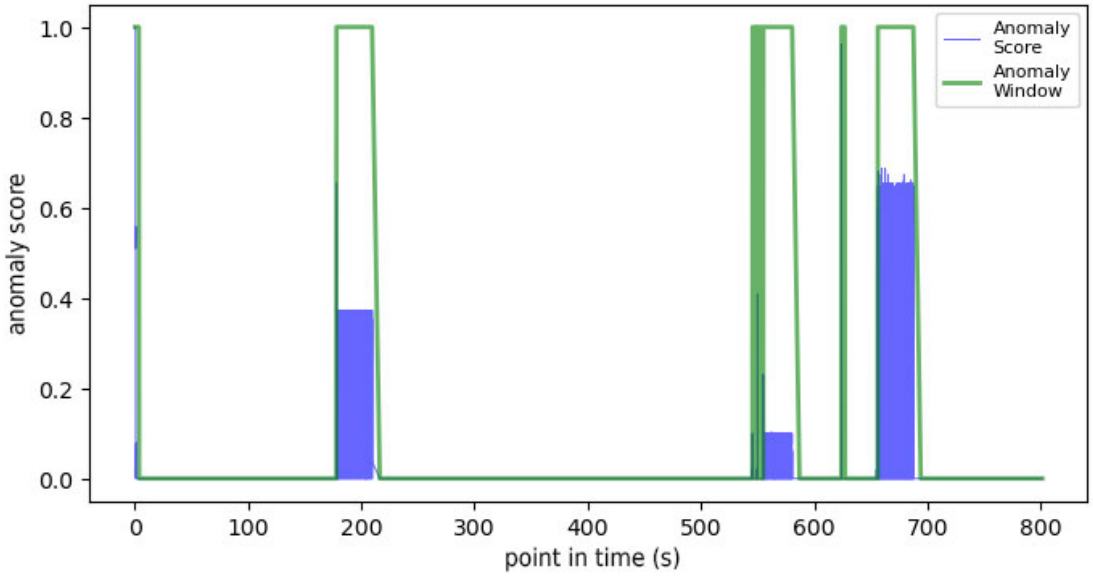

Figure 6.5: Detection results using a *static* window.

Method	rolling window	static window
TPR	0.97607	0.89206
FPR	0.01009	0.00474

Table 6.3: Detection rates for both input methods in comparison.

Based on the results, TPR and FPR can now be calculated for either method. The outcome of this calculation can be seen in table 6.3. Since a forced learning period is used, during which anomalies are ignored, that part is excluded from the calculation of the rates.
The detection rates show that both methods are reliable at overall detection of the attacks, while also causing false positives less than 1 % of the time.
A great coverage of the attacks can be achieved by using only jitter and a *rolling* window, measuring a TPR of 0.976 or 97.6 %. While better rates have been achieved for this method by counterintuitively enabling learning during anomalies, this came at the cost of processing time which has been prioritized.
The *static* window method with multiple metrics instead performs worse with a TPR of 0.892 or 89.2 %, mainly due to the bad coverage of the 2nd attack.
Finally, regarding the FPR, it's important to consider that it depends heavily on the proportion of non-anomalous against anomalous flow included in the test. The share of anomalous flow in this example is roughly 14 %. This is fairly high, considering that in a real-world scenario a response to the attacks should stop these from continuing or repeating. Furthermore the behaviour of the flow would be normal nearly 100 % of the time. To investigate this issue, an extended test including another 20 min of normal data in prior has been conducted. Since no false positives occured during the extended period of that test, an equally lower FPR was the result. In spite of these observations the shorter version is presented because it further highlights the speed in which the algorithm can adapt to the input.
Another observation concerning the false positives is that they mostly occur right after an anomaly. This could be happening due to contextual information in the TM still being messed up by the anomaly. A second potential reason is that the reestablishment of normal communication causes some irregular behaviour as well.

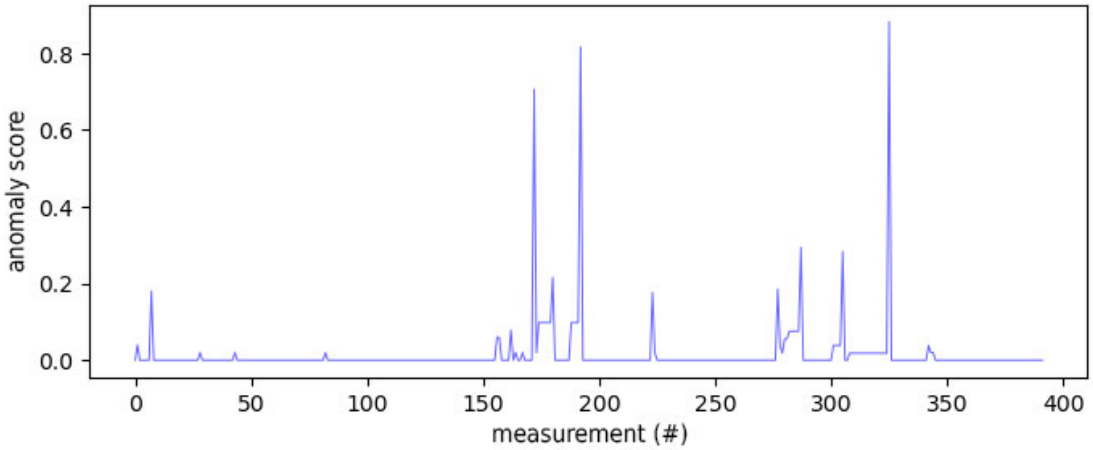

Figure 6.6: Typical behaviour of anomaly score during a collective anomaly, such as a DoS attack, which spans over many measurement periods. The timespan of this example is 2 s.

6.4 Anomaly window

Previously described in sec. 5.1.3, anomaly windows are used as a final interpretation of the anomaly score. It has been suggested that an MA could be used as a countermeasure to noise in the resulting anomaly score. Though based on the results of numerous testing using high resolutions with many iterations per second, there are multiple reasons not to use an MA for such purpose.
One of them is the high amount of anomaly scores valued 0 during an anomaly. When using high resolutions, scores above 0 are generally produced much less often than scores equal 0. A demonstration of this can be seen in figure 6.6, where 2 s worth of anomaly score results from the *rolling* window setup during the first attack are displayed. The result of this effect, in combination with MA, is that a sufficient smoothing length would have to be chosen to bridge these gaps, which causes an equally large delay until detection of actual anomalies.

Another reason is the lack of necessity to reduce noise in the anomaly score results, when the configuration of the system and input collection is well done. As a matter of fact, data sequences of normal behaviour express very few noise in the score after just a few seconds of learning (s. figure 6.4):
The number of scores above 0 is small and they mostly consist of low values (e.g. 0.01), which can be ignored using a simple threshold.

This doesn't mean that there are no scenarios where such a post-processing method would be beneficial. It rather depends entirely on the given data and the HTM model configuration. But even then, a more sophisticated approach of filtering the score could be of greater benefit to the system than a mere MA.
As a result, the raw anomaly score and a threshold have been chosen for the configurations as the anomaly window method.

In section 5.1.4 it has been explained that the high number of iterations could lead to fast learning of the anomalies themselves. The proposal to mitigate this effect was to disable learning in the TM during an active anomaly window. By applying this technique it was possible to keep the specialization towards normal behaviour of most configuration variants clean. This solution additionally improved the execution speed during the anomalies and their congested input. Tests have been conducted with this setting enabled and disabled and the *static* window configuration shows the necessity of this mechanism for it's success (s. results of disabled setting in figure 6.7). While the *rolling* window method only runs slightly faster with this setting enabled, the *static* window method shows significantly worse detection and execution speed (roughly 40 times slower) when it is disabled. The difference between these two likely stems from the increased resolution used with the *static* window which creates at least 3 times as much measurements for this test.

6.5 Timings

As mentioned in section 5, a fast reaction is a feature that would be well received within a safety-critical TSN environment. Timings are measured by comparing the beginning of an anomaly to the time when it is detected by the system.
Furthermore, the processing time of each component should be as low as possible because of the sequential nature of the system. The slowest component in the processing chain will dictate the maximum frequency, in which data can be processed.

6.5.1 Detection delay

The detection delay is defined by the amount of time between the first packet out of order and the time of the iteration, where the anomaly is detected by the system. It

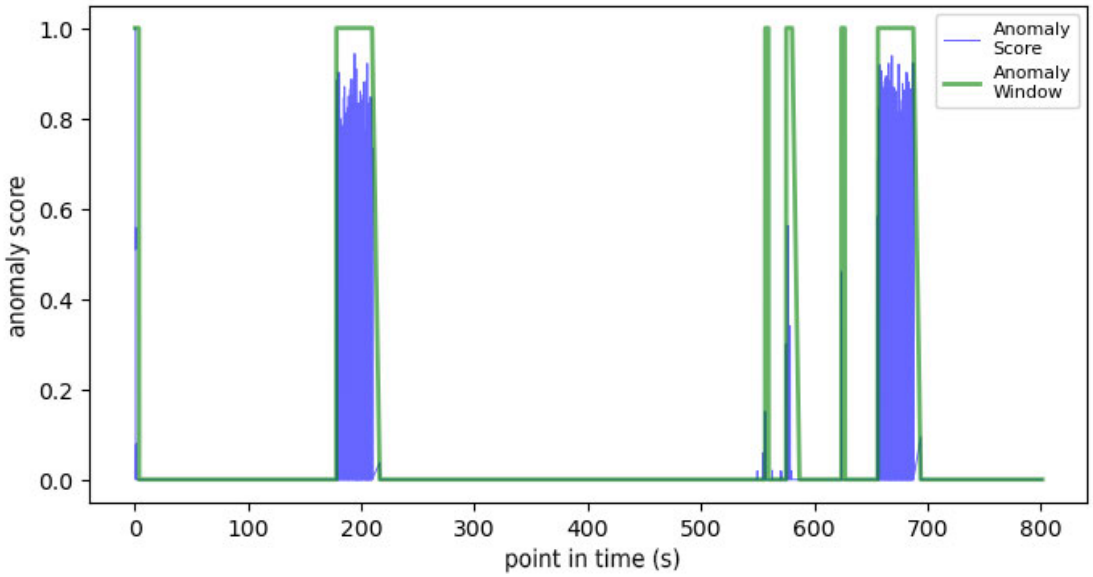

Figure 6.7: Detection results of *rolling* window configuration but with setting to suppress learning during anomalies disabled. The detection of the 2nd attack is now much worse.

is further influenced by the window size for metric collection and the sampling rate for rolling windows, since these values depict the smallest amount of time until the data is exposed to the HTM system.
Another factor for increased delay could be the interpretation method of the anomaly score (s. section 5.1.3), since some methods require knowledge of previous iterations and therefore introduce delay. In the final configurations of this work, only the newest score is interpreted (s. section 6.4), which doesn't add additional delay.
The actual delay measured for the described scenario is shown in table 6.4. It shows that the *rolling* window method has managed faster timings, staying well below 1 s, than the *static* window method, which needs at least 1 s for detection.
Determining whether or not this speed is sufficient for realtime detection depends on the type of flow it is applied to. Some connections (e.g. actuation, like breaking and steering) might need to be secured within just a few dozen ms at most after the start of an anomaly. The achieved delays of the *rolling* window method are on par with average human reaction times [1] and may suffice for less critical connections in the network, such as a camera used for parking. There the displayed video stream could be switched to a different camera without drastically impacting the driver's visual decision basis.
Nevertheless, the best result of 117.2 ms demonstrates adequate detection speed for an

Method	rolling window	static window
Attack 1	0.1172 s	1.1425 s
Attack 2	0.6808 s	2.6606 s
Attack 3	0.2029 s	1.1697 s

Table 6.4: Detection delay for both input methods in comparison.

input with much noise. The difference in delay between the methods mostly results from differently shaped input, which hints further optimization potential.
Additional improvements to the system or input could therefore provide even better results. Monitoring real TSN communication with time-critical properties for example would produce more stable metrics, eventually increasing the sensitivity of the system towards the violations of typical behaviour. As can be seen in figure 6.1 jitter, there is currently no obvious pattern to normal behaviour except for oscillation within a value range. Yet the detection of lack of that behaviour can take place within the fraction of a second.
A conclusion drawn from the detection delay is that sampling the input more frequently doesn't necessarily speed up detection, since the less frequently sampling method (5 ms vs. 1.5 ms) achieved faster detection. Instead it is important to choose a sampling method that amplifies the difference between normal and abnormal patterns (see difference of jitter between figure 6.1 and figure 6.2).

6.5.2 Processing delay

Processing delay caused in each step of the workflow will add to the total time until detection. More severe impacts of high processing times, which exceed the used time period for a single measurement, could be: increasing buffer or stack use, leading to inability of performing near real-time operation or even cause application crashes. A runtime analysis provides further insight into the time needed for the various operations: Built with efficiency in mind, the metric preprocessor and normalization step have proven to not slow down overall system performance significantly. These pre-processing steps have execution times around 16 µs/iter. on average with a standard deviation of 23 µs/iter. (rolling window, worse performance due to queue management).
HTM however produces higher timings, as would be expected due to it's higher complexity. Being the slowest component of the system, HTM will set the limits for the resolutions that can be achieved. A detailed dissection of it's processing time is shown

Method	rolling window	static window
Min	0.1013 ms	0.1225 ms
Mean	0.2153 ms	0.5801 ms
99th Percentile	1.6339 ms	2.6932 ms
99.9th Percentile	6.2228 ms	12.0408 ms
Max	18.4723 ms	26.1325 ms

Table 6.5: Processing delay for both input methods in comparison.

in table 6.5.
Next to min, mean and max, the 99th and 99.9th percentile have been included to get a better view of worst case scenarios. Particularly the difference between 99th and 99.9th percentile shows that there are few performance outliers. As for these, there is no guarantee that external factors like scheduling weren't the actual cause. The 99th percentile could be considered as a guideline for a realistical choice of the maximum resolution for these configurations. This would mean that the rolling window method could also run slightly faster subsampling while the static window method could use a longer resolution for consistent performance.
While the measured processing delays should qualify the system for many realtime purposes, porting the system over to the original C++implementation of HTM would further reduce execution time and overhead.

6.6 Memory consumption

Using a large traffic capture file, an extended test has been performed to evaluate the RAM usage of the proposed system and HTM configuration. As can be seen in figure 6.8, the system uses about 240 MB of memory when reaching a steady state. This would mean that multiple instances could run on a single device, as long as it has enough memory and processing power.

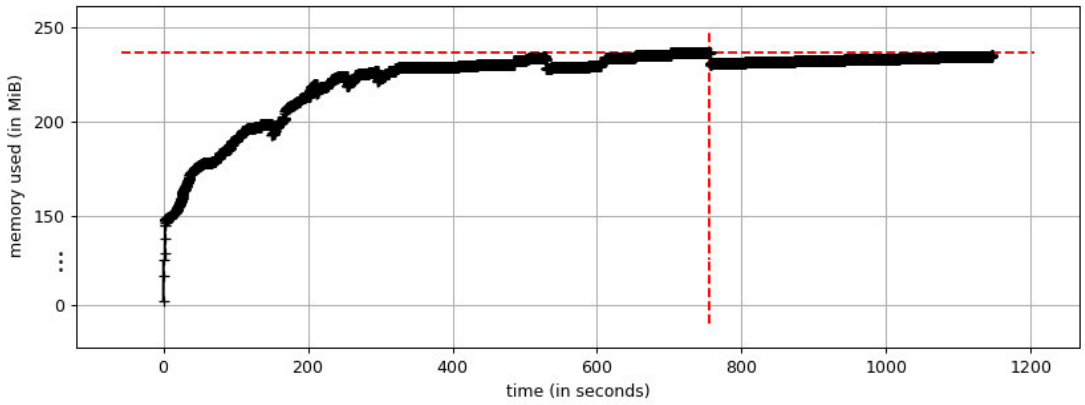

Figure 6.8: HTM memory consumption

6.7 Tuning of parameters

A great part of developing an HTM based system is to find a balance of parameters for the encoding, SP and TM, that work well together. For this it's important to keep the separate responsibilities of each component in mind, which is as following:
The encoding step translates data into an HTM-compatible format,
the SP interprets that data and learns to distinguish input patterns from each other (spatial features),
while the TM learns the order in which they appear and which patterns are related temporally (temporal features).
Great inter-dependence of these 3 means that a misconfiguration on any part can have a very negative effect on the end result. One big problem of the configuration process is that without a suitable encoded input, the SP & TM will always produce dissatisfying results. On the other hand, finding a good encoding setting can be challenging when a lot of different data is fed into the system and the SP & TM are not yet optimized.

Hence, the following procedure is suggested for general tuning of the system for anomaly detection, since it has delivered the best results of all approaches:
Based on the data that is to be encoded, roughly choose settings that can express differences in the data that shall be noticed.
Configure SP & TM to be just sensitive enough, so that normal flow behaviour produces low and steady anomaly scores.
Revisit the encoding settings and try various methods (resolution, normalization etc.) by

monitoring the anomaly score output and trying to raise it on anomaly occasions while still keeping it low during normal behaviour.
Do fine-tuning of the SP in regards to sensitivity for different input patterns, as it acts like a second encoder.

Following these guidelines can provide a good basis for the system but in each step there are many considerations, which will be explained in the following paragraphs. To wrap up the configuration in the end, it might still be necessary to reconfigure the different parts a number of times to better extract spatial or temporal features. An alternative to the manual finalization can also be provided by particle swarm optimization, which is included in the HTM framework. Though this would require the implementation of an automated system performance metric (e.g. combination of TPR and FPR) and a certain configuration baseline to produce desired results.

6.7.1 Input & Encoding

Encoding data in a way that it fits into a bit-array can be done in many different ways. While very specialized methods can provide great results (e.g. mapping of single CAN fields), this work is using scalar values in a time series and will therefore use the scalar encoder.
An important note about the effectiveness of the RDSE versus the scalar encoder is that the RDSE significantly reduces run-to-run consistency in comparison. Using it would cause such a great inconsistency that the similarity between runs became unreliably low. The scalar encoder instead produces results almost so consistent that deterministic behaviour could be assumed, even though random factors are still used during the initialization of HTM.
The following settings are available in regards to input and will have an impact on the system performance:

- Sparsity, describing the share of bits used to represent the data.
- Minimum and maximum value. Values outside of this range can either be clipped into the range or the range can be repeated periodically.
- One setting out of size, radius or resolution. In combination with the sparsity and value range, each of these result in a rule of three and are therefore mutually exclusive.

- Normalization or raw input. Other forms of pre-processing such as filters might be used instead. This is not an encoder setting.
- Insertion of zero padding when no measurements are available

The value range (minimum and maximum) should be chosen so that the typically produced values can be accomodated. In this system the choice is to clip values into that range rather than using a periodic range. This was based on the assumption that it wouldn't normally happen and that it would be interpreted as abnormal behaviour if encoded arrays would have active bits at the edge of their boundaries.
The sparsity should be sparse in general, allowing for many different data representations but also some overlap. For this system a general value of 0.1 or 10 % has been chosen since other variations have shown no real benefit and other settings can compensate for higher or lower choices of sparsity.
One of such settings that compensate the sparsity is the size, radius or resolution. In this work, a constant size of 300 has been chosen, since it allowed for an even balance of multiple inputs. This way, only the value range remains a changeable factor for the actual resolution or overlap of the encoding, eliminating two other factors from the configuration process. The ranges have been picked by observing the highest and smallest values occuring in the metric measurements. Different structuring of the range by lower maximum limits and periodicity, higher maximum limits or other variants have not benefited the results.
Additionally, neither the normalization methods nor the use of padding have improved the system in this test scenario.

6.7.2 Spatial Pooler & Temporal Memory

Using the default HTM configuration (s. table 6.6) as a baseline, a manual heuristic search for parameters has been performed. Single parameters have been incremented and decremented in steps while monitoring the resulting anomaly scores in regards to FPR and TPR. By locking one or more parameters to a fixed value, such as n_{col}= 256, other parameters can be searched thoroughly to establish a number of well working variants. Then the best performing models are used as a new baseline and the process is repeated until the system performs well enough on a parameter set. Additionally, many steps can be tested at the same time by running multiple instances of the program at the same time.

SP parameters	value
B_{str}	0
n_{col}	1024
D_{loc}	0.05
r_{pot}	16
p_{pot}	0.5
syn_{inc}	0.05
syn_{con}	0.1
syn_{dec}	0.008
$g.inhib$	false

TM parameters	value
$n_{celpercol}$	32
seg_{max}	255
$tm.syn_{active}$	13
$tm.syn_{initperm}$	0.21
$tm.syn_{max}$	255
$tm.syn_{min}$	10
$tm.syn_{new}$	20
$tm.syn_{dec}$	0.1
$tm.syn_{inc}$	0.1

Table 6.6: Default HTM parameters.

During this process some parameters quickly showed favorable value ranges:
syn_{dec}, seg_{max}, $tm.syn_{max}$, $tm.syn_{dec}$ and $tm.syn_{inc}$ have remained unchanged through most tests after good default performance and various failed attempts to adjust them.
n_{col} has worked best below a quarter of it's default value, which decreases the capacity and size of spatial patterns in the SP. This then acts like a compression layer which reduces the dimensionality of the data, further reducing noise. A similar technique is often used on recreational neural networks (e.g. autoencoder).
Concerning the TM, the $n_{celpercol}$ has been kept below 10 most of the time. One reason for this is that a higher $n_{celpercol}$ increases execution time. Another reason is that the sequential patterns of the testing data are rather short-lived. A $n_{celpercol}$ of 6 provides a well balance between high-order memory capability and execution time for the test data.

These findings have provided the minimal basis for final tuning of the remaining parameters using the described search process.

7 Conclusion

In this work an HTM-based NADS for IVN has been implemented and evaluated. Around the HTM framework a number of options have been added for reactive online learning suppression, input normalization, output interpretation and padding. To insert data into the HTM system, traffic capture and network metric calculation modules have been implemented with full control over resolution and the type of measurement windows. By using these options in a large variety of constellations and observing their effectiveness, the ability of HTM to work in a realtime environment and also what steps it takes have been evaluated. It has been found that many of these constellations work well while some excel in either detection performance, hardware performance or provide a good mix of both.

HTM could fulfill the provided requirements to a degree. It shows sufficient performance for realtime environments in terms of execution speed. With detection delays of a few 100 ms at most in a noisy dataset, as well as achieving good detection rates, the HTM framework has demonstrated it's potential benefits for realtime anomaly detection. HTM is capable of providing sufficient detection to serve as a decision basis for responses to attacks or other types of anomalies within the IVN. Another property that is often mentioned about HTM is it's robustness when it comes to input containing a lot of noise, which could be observed in the tests as well.

As with many other types of ML-based NAD, the effectiveness of the system depends on some optimization for the flow that is to be monitored. This is due to each type of communication (e.g. steer-by-wire and videostream) expressing different behaviour. Part of this optimization can be automated using the particle swarm optimization for parameters, which is included in HTM.

Thanks to the simple structuring of the encoding process, it is possible to easily implement custom encoders for a variety of purposes, such as the mapping of CAN fields for application layer detection [19]. Other than that, HTM also provides other mechanisms that are typically used in ML, such as categorization or image processing. This enables it for a similarly wide range of applications to that of Neural Networks (NN).

The framework also shows some niche advantages over NNs in a few areas, including: Online learning capabilities, allowing for changes in communication behaviour (e.g. through an update of a program) to get learned during runtime. Simple deployment, since no external libraries besides HTM are needed for the system to work. And lastly the ability to use multivariate inputs as a means of cross-validation.

7.1 Outlook

For future work regarding ML-based systems in an IVN it would be interesting to see if and in which categories HTM can provide an edge over NNs. Especially the performance in a TSN-based network on streams with more stable patterns remains to be seen, as it could provide an environment for even faster detection times and accuracy for all types of NADSs. Due to properties like online unsupervised learning it can also be applied in different ways from NNs. For example by creating subsequent HTM systems dynamically. These could then further process the information provided by the parent system (e.g. detecting the streams of a range of sensors and monitoring their sending behaviour with subsystems). While the community around HTM is small in size when compared to NNs, the project is enjoying rather frequent additions and updates. As each problem has a different best solution, the larger variety of sophisticated approaches will only benefit the outcome.

Bibliography

[1] Abbasi Kesbi, Reza ; Memarzadeh-Tehran, Hamidreza ; Deen, M.J.: A Technique to Estimate the Human Reaction Time Based on Visual Perception. In: *Healthcare Technology Letters* 4 (2017), 01

[2] Ahmad, Subutai ; Lavin, Alexander ; Purdy, Scott ; Agha, Zuha: Unsupervised real-time anomaly detection for streaming data. In: *Neurocomputing* 262 (2017), S. 134–147. – URL https://www.sciencedirect.com/science/article/pii/S0925231217309864. – Online Real-Time Learning Strategies for Data Streams. – ISSN 0925-2312

[3] Ahmed, Mohiuddin ; Naser Mahmood, Abdun ; Hu, Jiankun: A survey of network anomaly detection techniques. In: *Journal of Network and Computer Applications* 60 (2016), S. 19–31. – URL https://www.sciencedirect.com/science/article/pii/S1084804515002891. – ISSN 1084-8045

[4] community htm: *htm-school-viz*. GithubRepository. Jun 2021. – URL https://github.com/htm-community/htm-school-viz

[5] community htm: *htm.core*. GithubRepository. Feb 2021. – URL https://github.com/htm-community/htm.core

[6] Cui, Yuwei ; Ahmad, Subutai ; Hawkins, Jeff: Continuous Online Sequence Learning with an Unsupervised Neural Network Model. In: *Neural Computation* 28 (2016), 11, Nr. 11, S. 2474–2504. – URL https://doi.org/10.1162/NECO_a_00893. – ISSN 0899-7667

[7] Cui, Yuwei ; Ahmad, Subutai ; Hawkins, Jeff: The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding. In: *Frontiers in Computational Neuroscience* 11 (2017), S. 111. – URL https://www.frontiersin.org/article/10.3389/fncom.2017.00111. – ISSN 1662-5188

[8] Halba, Khalid ; Mahmoudi, Charif ; Griffor, Edward: Robust Safety for Autonomous Vehicles through Reconfigurable Networking. In: *Electronic Proceedings in Theoretical Computer Science* 269 (2018), Apr, S. 48–58. – URL http://dx.doi.org/10.4204/EPTCS.269.5. – ISSN 2075-2180

[9] Hawkins, Jeff ; Ahmad, Subutai: Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex. In: *Frontiers in Neural Circuits* 10 (2016), S. 23. – URL https://www.frontiersin.org/article/10.3389/fncir.2016.00023. – ISSN 1662-5110

[10] Häckel, T. ; Meyer, P. ; Korf, F. ; Schmidt, T. C.: Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication. In: *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019, S. 1–5

[11] IEEE 802.1 TSN Task Group: *IEEE 802.1 Time-Sensitive Networking Task Group*. Online. – URL https://1.ieee802.org/tsn/

[12] Kim, S.: Detecting contextual network anomaly in the radio network controller from bayesian data analysis. (2015)

[13] Meyer, P. ; Häckel, T. ; Korf, F. ; Schmidt, T. C.: Network Anomaly Detection in Cars based on Time-Sensitive Ingress Control. In: *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, 2020, S. 1–5

[14] Padilla, D. E. ; Brinkworth, R. ; McDonnell, M. D.: Performance of a hierarchical temporal memory network in noisy sequence learning. In: *2013 IEEE International Conference on Computational Intelligence and Cybernetics (CYBERNETICSCOM)*, 2013, S. 45–51

[15] Purdy, Scott: Encoding Data for HTM Systems. (2016), 02

[16] Rajbahadur, Gopi K. ; Malton, Andrew J. ; Walenstein, Andrew ; Hassan, Ahmed E.: A Survey of Anomaly Detection for Connected Vehicle Cybersecurity and Safety. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, S. 421–426

[17] Steinbach, Till: *Ethernet-basierte Fahrzeugnetzwerkarchitekturen für zukünftige Echtzeitsysteme im Automobil*. Springer Vieweg, 2018. – ISBN 978-3-658-23499-7

[18] Szancer, S.: Traffic Analysis in V2X Application-Level Gateways. (2021)

[19] Wang, Chundong ; Zhao, Zhentang ; Gong, Liangyi ; Zhu, Likun ; Liu, Zheli ; Cheng, Xiaochun: A Distributed Anomaly Detection System for In-Vehicle Network Using HTM. In: *IEEE Access* 6 (2018), S. 9091–9098

[20] Waszecki, P. ; Mundhenk, P. ; Steinhorst, S. ; Lukasiewycz, M. ; Karri, R. ; Chakraborty, S.: Automotive Electrical and Electronic Architecture Security via Distributed In-Vehicle Traffic Monitoring. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36 (2017), Nr. 11, S. 1790–1803

A Appendix

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Hierarchical temporal memory for in-car network anomaly detection

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original