Baltic J. Modern Computing, Vol. 14 (2026), No. 2, pp. 376–401
https://doi.org/10.22364/bjmc.2026.14.2.06

Modelling and Assessment of Asynchronous Evidence-based Network Flows for Cyber-attack Detection

Virgilijus KRINICKIJ, Linas BUKAUSKAS,

Institute of Computer Science, Vilnius University, Vilnius, Lithuania

virgilijus.krinickij@mif.vu.lt, linas.bukauskas@mif.vu.lt

ORCID 0009-0005-8248-5690, ORCID 0000-0002-9781-9690

Abstract. In cybersecurity, analysing network traffic is critical for identifying potential threats and mitigating incidents. Traditional approaches to network traffic analysis often rely on synchronous methods that may not fully capture the dynamic nature of network behaviour. This paper presents an approach for asynchronous evidence-based network traffic assessment, experimenting on synthetic cyber-attack templates and large network flow datasets available online. Our approach leverages asynchronous data collection to capture network traffic at varying time intervals, enabling the identification of incidents across different time frames and locations by processing and aligning the data. By combining asynchronous data collection with evidence-based assessment, our approach enables cybersecurity analysts to gain deeper insight into network traffic dynamics, enhance threat detection capabilities, and improve incident response effectiveness. We demonstrate the effectiveness of our approach through experimental evaluations using synthetic cyber-attack templates in a virtual environment, while capturing network flows. In summary, our research advances the field of network traffic analysis by presenting an approach that addresses the shortcomings of conventional synchronous techniques and lays the groundwork for more resilient, adaptive cybersecurity solutions.

Keywords: Network traffic analysis, dynamic time warping, incident detection, asynchronous evidence assessment, synthetic attack modelling, cybersecurity

1 Introduction

In the ever-evolving cybersecurity landscape, timely and accurate network-traffic analysis is the cornerstone of defence against escalating cyber threats. Conventional synchronous network analysis models frequently suffer from significant latency and resource bottlenecks, especially when processing the high-volume traffic characteristic of modern network flows (JIANG et al., 2014). These challenges are amplified by the

ever-growing number of internet-connected devices, which increases network load and the scale of telemetry that must be monitored. Therefore, network stream analysis for anomaly and cyber-attack detection in network flows has become increasingly important (Argyraki and Cheriton, 2005; Li and Chen, 2004; Heenan and Moradpoor, 2016).

Threat actors have also evolved alongside the increasing scale and complexity of modern networks. As a result, sophisticated cyber-attacks are difficult to detect in large-volume network flows (Kalinin and Krundyshev, 2021). Moreover, adversarial activity often originates from multiple geographic locations and time zones, resulting in dispersed, asynchronous traffic. The asynchronous result complicates the correlation of possible attack vectors, and the identification of cyber incidents becomes significantly harder (Diab et al., 2019).

Today, packet capture (PCAP) remains widely used as one of the most prominent mechanisms for network flow analysis. This allows experts to analyse network flows in offline mode for sophisticated cyber-attack detection. Many tools have been created to facilitate passive analysis of these files (Ulmer et al., 2019). However, due to the surge in sophisticated cyber-attacks, a shift to more dynamic and adaptive security measures is needed (Pandey, 2023). Having traditional defence systems assume a global view of all security logs being centrally gathered and fed to decision-support systems (Heenan and Moradpoor, 2016). These systems use machine learning (ML) algorithms, along with other algorithms and tools, to detect network traffic threats. In this context, predictive modelling will improve security measures by enhancing detective capabilities, especially in network flow analysis (Mehta et al., 2022; Komisarek et al., 2021; Ranjan et al., 2023; Clotet et al., 2018). However, in most cases, individualised solutions are applied to develop and support researchers or network investigators in responding to sophisticated cyber-attacks. Therefore, innovative approaches that enable dynamic control of remotely gathered data and algorithms applicable to a multitude of sources remain a critical challenge.

The key aim of this paper is to present an algorithmic method for assessing attack templates. The main goal is to implement and test dynamic network processing across asynchronous computer networks for incident and anomaly detection, while providing incident forensics and reporting. We implement distributed network monitors to gather network flows from various networks and network points to achieve the goal. For the experiments, we implemented data filters at the edge of each machine and modelled the desired flow. Such early data filtering enables prioritising relevant data.

The rest of this paper is organised as follows. In Section 2, relevant work done regarding asynchronous alignment of network flows, used algorithms for attack template forensics and the overall situation in the field are described. In Section 3, we describe the proposed model for the modelling and assessment of asynchronous network flows using architecture and flow processing. We continue with the implementation Section 4 and experiments Section 5. In the end, in Section 7, the conclusion and in Section 8, future work are presented.

2 Related Work

Recent research in cybersecurity focuses on anomaly detection and analysis of network flows. Aligning asynchronous records using dynamic time warping (DTW) can identify relevant actors and their communications across heterogeneous networks, aiding incident analysis (Krinickij and Bukauskas, 2024; Diab et al., 2019).

The current approach to asynchronous record alignment uses predefined algorithms on PCAP files. In the context of global network flow alignment, algorithms such as Needleman-Wunsch from bioinformatics (Likic, 2008) can be used as one variant. For local network flow alignment, DTW or Smith-Waterman algorithms are preferable (Müller, 2007; Beddoe, 2004). However, the classical DTW algorithm has several limitations. It is computationally expensive and requires comparing every element of two sequences, resulting in quadratic complexity and poor scalability for long inputs (Macrae and Dixon, 2010). Thus, DTW struggles with heterogeneous PCAP files from heterogeneous network points. Packets arrive in mismatched sequences. Therefore, DTW requires a huge warping window and becomes practically unusable for incident reconstruction (Krinickij and Bukauskas, 2024; Liang et al., 2021). Therefore, these constraints render DTW unsuitable for continuous network flow analysis as it requires both sequences to be fully known in advance (Krinickij and Bukauskas, 2024). For these reasons, researchers are exploring various approaches to improve the efficiency and applicability of DTW.

Efforts to reduce the quadratic computational burden include algorithms such as fast DTW (FastDTW), although this has not always been shown to outperform the original DTW (Wu and Keogh, 2020). Other variants propose different optimisation strategies: some approaches reduce the number of computations, whereas improving efficiency modestly (Lou et al., 2015). Segmental DTW divides the global cost matrix into smaller sub-matrices to be processed independently before merging results, although the runtime of weakly-ordered segmental DTW (WSDTW) is the same as DTW, and the runtime of strictly-ordered segmental DTW (SSDTW) is twice as much (Tsai, 2021). Event DTW (EventDTW), whereas more effective at handling frequency differences, retains quadratic complexity (Jiang et al., 2020). Parallelisation techniques such as parallelised diagonal DTW (ParDTW) and accelerated DTW (DTWax) exploit GPUs to accelerate execution (Yang et al., 2022; Sadasivan et al., 2023; Tralie and Dempsey, 2020), although their complexity remains the same as that of standard DTW. Flexible DTW (FlexDTW) introduces sequence start and end points flexibility, improving runtime but still incurring high memory usage (Bükey et al., 2023). Among these, the University of California, Riverside DTW (UCR-DTW) has been highlighted as one of the fastest implementations, capable of handling disk I/O efficiently and answering queries in fractions of a second (Rakthanmanon et al., 2012). Conversely, some attempts, such as 2-dimensional DTW (2DDW), have introduced even higher computational costs, with complexity increasing to $O(N^6)$ (Lei and Govindaraju, 2004). Although DTW traditionally compares only pairs of sequences (Krinickij and Bukauskas, 2024), researchers have proposed methods to extend it to multiple concurrent streams. Subsequence DTW (SUCR-DTW) employs multi-threading to match multiple time series in parallel, whereas extension subsequence DTW (ESUCR-DTW) generalises the approach to allow candidate detection across different subsequence lengths

(Giao and Anh, 2016). Another technique, multiple DTW (multiDTW), uses nested hierarchical query subgroups and tree-based data structures to prune less relevant queries, thus improving efficiency (Kremer et al., 2011).

Sliding window properties present another area for significant optimisation opportunities. The Sakoe–Chiba band was one of the earliest methods restricting alignments to a limited diagonal region, although this may exclude valid matches (Sakoe and Chiba, 2003). Later methods, such as online DTW (ODTW), reduce computational costs by comparing only small regions of the matrix, though this can lead to lower accuracy if older patterns are important (Oregi et al., 2017). Dynamic search step DTW (SegrDTW) instead adapts dynamic search windows around local features, avoiding rigid global constraints (Guo et al., 2022). Other techniques exploit structural properties: blocked DTW (BDTW) leverages repeated values to shrink matrix size (Sharabiani et al., 2018), whereas space-efficient DTW (SparseDTW) dynamically opens only promising cells, ensuring accurate results without fixed bands (Al-Naymat et al., 2012). Complementary to these approaches, some researchers have employed circular buffer structures to efficiently store subsequences, thereby reducing memory overhead (Rakthanmanon et al., 2012; Giao and Anh, 2016).

In addition, data normalisation has been identified as a crucial step for precise similarity search. The possible problem lies in additional computational and memory costs. Normalisation is often applied per window in streaming contexts or optimised to minimise performance penalties (Rakthanmanon et al., 2012; Giao and Anh, 2016; Liang et al., 2021). Gathering large amounts of network flow from heterogeneous network points is a problem. Various open-source stream processing solutions can be used to collect network flow data in a centralised environment. For example, the widely used Apache Kafka platform (Komisarek et al., 2021; Kumar et al., 2025), or another less popular but applicable solution for amassing network flow, is the open-source message broker RabbitMQ (Kotov et al., 2024; Maatkamp et al., 2016). The latter might be preferred owing to its high throughput, scalability, and the ability to filter and order incoming network traffic data (Dobbelaere and Esmaili, 2017; Jones et al., 2011).

Adding such solutions using different ML models enables keeping up with network traffic streams. However, the application of ML to constantly evolving network flow streams is relatively new compared with passive analysis methods that have been around for some time (Mulinka and Casas, 2018; Ulmer et al., 2019; Rivera et al., 2021). In addition, in terms of sophisticated cyber-attacks, ML models perform worse than expected. Currently, attackers can poison models with data poisoning. This leads to heavily unreliable ML models (Khan and Ghafoor, 2024). Even small amounts of poisoned data can reduce accuracy and cause misclassification (Kure et al., 2025), making covert attacks difficult to detect while threatening critical systems and privacy (Maramreddy and Muppavaram, 2024). Such attacks, like label manipulation, data injection, or backdoors, undermine reliability in safety-critical domains and may even expose sensitive training data (Srivastava et al., 2024).

These studies collectively highlight significant gaps in current algorithms and methods for detecting cyber-attacks in asynchronous network flows. To the best of our knowledge, none of the prior studies mentioned explicitly addressed asynchronous evidence alignment in large network flows for a sophisticated cyber-attack assessment.

3 Proposed Model

In this section, we present a formal model for asynchronous network flow analysis, enabling approximate time-similar feature correlation to overcome the limitations of traditional synchronous methods. The assumption underlying the model's establishment is that time is not globally correlated and that the recorded time series are not homogeneous, implying that the time intervals of each data source are asynchronous.

3.1 Problem Formulation

We designed a model of the dynamic behaviour of network data flow to ease the detection of sophisticated cyber-attacks in recorded network data, focusing on asynchronous data flows between attacker and target machines in a cybersecurity context.

Definition 1 (Data Sources). Asynchronous data sources from heterogeneous points are presented as follows

$$\mathcal{D}_t = \{d^1, d^2, \ldots, d^n\}_t \tag{1}$$

where n is an arbitrary number of data sources that can be observed at a synchronised time t.

All asynchronous data sources observing the network are defined as $\mathcal{D}^*$ over an unspecified time interval. Two data sources yield uncommon data density in time as well as time floatation. d_t^1 and $d_{t'}^2$ where $d_t^1, d_{t'}^2 \in \mathcal{D}^*$ satisfying the condition that $t \neq t'$. Time t is a timestamp used to correlate local data occurrence sequentially, but is not considered as globally correlated time.

Definition 2 (Data Stream). We define $\mathcal{S}_t^d$ as a stream of network-observable packets sent from the data source, which is an actor's machine. The machine may be a threat actor, the target actor's machines, or any intermediate proxy. S stream has k packets of network data source $d = d_t^i \in \mathcal{D}_t$. Then,

$$\mathcal{S}_t^d = \langle P_1, P_2, \ldots, P_k \rangle_t^d \tag{2}$$

is a sequence of packets, where k is the number of observations in the data stream.

If the data stream is continuous in time, we denote the data stream as $\mathcal{S}_{[t;\infty]}^d$, and the source as $d = d_{[t;\infty]}^i \in \mathcal{D}^*$ The current time of the continuous data stream will be called now referred to the current state of time.

Given the data stream $\mathcal{S}_t^d$, a unique property is defined for each packet P_t^d. We denote $\mathcal{F} = \langle f_1, f_2, \ldots, f_m \rangle$ as packet properties, where f is the structural element of P_t^d. To simplify the notation, we use $P_t^d.f$ to show the relationship between the packet and a specific property.

Definition 3 (Packet Property Vectors). Any packet $P_t^d \in S_t^d$ is associated with a finite sequence of property value vectors according to properties f. We denote these property value vectors by:

$$\mathcal{V}(P_t^i.f) = \langle p_t^i | p_t^i \leftarrow P_t^i.f \wedge P_t^i \in S_t^i \rangle, \tag{3}$$

where j is the number of values of packet properties in stream S_t^d and p_t^d is the packet value of property f of $\mathcal{F}$.

To shorten the notation, we use $\mathcal{V}_t^i(f)$ to denote the packet value vector according to property f.

We define a packet vector filter for data packets based on their properties. The packet filter is a function $\mathbb{F} : \mathcal{V} \times \mathcal{F} \rightarrow \mathcal{V}$ that distinguishes packets according to the criteria from the data stream S_t^d.

Definition 4 (Packet Filter). Let ϕ be a defined property of a packet. We define the packet filter $\mathbb{F}$ as a function

$$\mathbb{F}\big(\ \mathcal{V}_t^d(f),\ \phi\ \big) = \langle p_t^d | p_t^d \in \mathcal{V}_t^d(f) \wedge f = \phi \rangle \tag{4}$$

Here, $\mathcal{V}_t^d \in \mathcal{S}_t^d$ is the filtered data stream that contains only packets whose property vectors satisfy ϕ.

Such filters distinguish packets in the stream according to the selected properties with a time complexity $\Theta(n)$, where n is the number of packets in the stream.

Due to the nature of data streams to be continuous, the ring buffer RBW^n is defined as an abstract data structure that is a fixed-size ring buffer for $\mathcal{S}_t^d$. The RBW is a data structure queue that operates enqueue and dequeue with time complexity $\Theta(1)$. Therefore, the alignment process is to compare buffered data with the specific parameters located, ignoring other filtered-out records.

Definition 5 (Threshold for Aligned Time Vector Window). We define a threshold ω and bins $\mathcal{K}$. The threshold ω is a limiter for the $\mathcal{S}_t^d$ if $\mathcal{S}_t^d$ exceeds ω amount of vectors. Then

$$\mathcal{V}(P_t^i) \in \mathcal{M}_{[t_s,t_e]} > \omega \tag{5}$$

We define a transformation $\mathcal{T}()$ for $\mathcal{V}(P_t^i) \in \mathcal{M}_{[t_s,t_e]}$ where,

$$\mathcal{T}(\mathcal{V}(P_t^i)) \in \mathcal{K} \tag{6}$$

then,

$$\mathcal{M}_{[t_s,t_e]} = \left\langle\ \mathcal{T}(\mathcal{V}(P_t^i))\ \middle|\ \mathcal{M}_{[t_s,t_e]} > \omega, t \in [t_s,t_e] \right\rangle \tag{7}$$

is a vector of filtered packets according to the threshold.

To simplify notation we use $\mathcal{M}_{[t_s,t_e]}$ as $\mathcal{M}$ in $[t_s,t_e]$.

To align within the threshold ω any given $\mathcal{M}'$ and $\mathcal{M}''$ we define alignment operator $\bowtie$: $\mathcal{V} \times \mathcal{V} \times \omega \Rightarrow \mathcal{V}$ aligns two independent data vectors,

$$\begin{aligned}\mathcal{M}' \bowtie_\omega \mathcal{M}'' ::= \big\langle P_{t'}^{i'}, P_{t''}^{i''} |\ \ & \forall t', t'' \exists P_{t'}^{i'} \in \mathcal{V}' \exists P_{t''}^{i''} \in \mathcal{V}'' \wedge \omega \in \mathbb{N} \wedge \\ & (\mathbb{F}\big(\ {\mathcal{V}'}_t^d(f),\ \phi\ \big) \sqcap \mathbb{F}\big(\ {\mathcal{V}''}_t^d(f),\ \phi\ \big)) \neq [] \big\rangle\end{aligned}$$

respecting the filter ϕ.

The proposed concepts play a major role in the processing and analysis of asynchronous network flows in PCAP files. Table 1 summarises the main notations used in the provided definitions.

Table 1: **Summary of the Main Notations.**

Notation	Explanation
$\mathcal{S}_t^d$	Data Source
RBW	Ring Buffer Window
$d_t^1, d_{t'}^2 \in \mathcal{D}^*$	Two data sources from different times
$t \neq t'$	Asynchronous time between two data sources
$d = d_t^i \in \mathcal{D}_t$	Data source is a part of data stream
$d = d_{[t;\infty]}^i \in \mathcal{D}^*$	Synchronized data streams in time
$\mathcal{F}$	Packet properties
$\mathcal{V}(P_t^i.f)$	Packet Property Vectors
$\mathbb{F}\Big(\ \mathcal{V}_t^d(f),\ \phi\ \Big)$	Packet filter
$\mathcal{M}_{[t_s,t_e]}$	Aligned Time Vector Window with start and end times
$\mathcal{V}_t^d \in \mathcal{S}_t^d$	Filtered Data

3.2 Proposed Model Architecture

DTW functionality constraints limit the flexibility of stream processing. Given a pair of sequences, DTW stores them in a matrix in computer RAM without any constraints or optimisation for later computation (Krinickij and Bukauskas, 2024). Thus, to further this process, we present a similarity algorithm that allows asynchronous data alignment. We present pseudo-code algorithm 1 of the data stream similarity that integrates all the mentioned concepts to process asynchronous network flows for incident detection in more detail. The algorithm maps events by analysing similarities within data streams using RBW and $\mathcal{M}_{[t_s,t_e]}$ windows, thereby aligning historical events across asynchronous PCAP files.

The required input consolidates the arguments required for an initial program launch. We ask for the input data stream, the packet property, and its vector, which serves as a beacon between the two data streams for similarity alignment. The threshold and bins parameters are the verification parameters required to verify that the beacon vector amount gathered between the two data streams does not exceed the specified threshold in line 1.

From lines 2 to 12, we load the data streams in parallel, and for all packets in data streams, we are looking for packet property vectors that were defined in the input. The filtering process of the vectors is performed inside the RBW, whose size is also specified during the input. After vectors are identified, we extract the corresponding time intervals for all matched vectors in line 13.

Lines 7 to 10 show a ring buffer window, which is a constant window that does not recreate itself. Both streams have RBW as the primary sliding window of fixed size, which slices the given stream into RBW and moves forward vertically. The RBW have many intrinsic properties that are used over our recorded synthetic PCAP files. In addition, PCAP files from different internet sources are added to enhance and correlate the experimental results. Gathered cyber-attack evidence, per the simulation, realistically,

Algorithm 1 Data Stream Similarity Algorithm

Input: $\mathcal{S}_t^d$, f, $\mathcal{V}(P_t^i)$, RBW, threshold Ω, bins $\mathcal{K}$
for all $\mathcal{S}_t^d$ **in parallel do**
  for all packet p in $\mathcal{S}_t^d$ **do**
    $match(\text{RBW}(\mathbb{F}(p_t^d)))$
    extract p_t^d
    if $bytes \leq \text{RBW}$ **then**
      enqueue to RBW $\leftarrow bytes$
    else
      dequeue RBW $bytes \leftarrow 0$
    end if
  end for
end for
$all\mathcal{V} \leftarrow \cup match(\text{RBW}(\mathbb{F}(p_t^d)))$ **if empty then return**
$alignedTS \leftarrow ts \leftarrow \min(all\mathcal{V}), te \leftarrow \max(all\mathcal{V})$
$M_{[t_s,t_e]} \leftarrow \{p_t^d \in alignedTS \mid ts \leq t \leq te\}$
if $useBinning$ **then**
  $seq \leftarrow \text{Histogram}(M_{[t_s,t_e]}, \Omega, \mathcal{K})$
end if
$dist \leftarrow \text{DTW}(s_1, s_2)$
return $dist, t_{E2E}$

synthetic attack templates are already present in the computational procedures for the experiment.

Using the firs in first out (FIFO) principle, the ring buffer stores only the most recent packets from data streams. Fig. 1 shows the process flow between two data streams of different lengths. The window is closed if the capacity is reached, and all other packets are discarded.

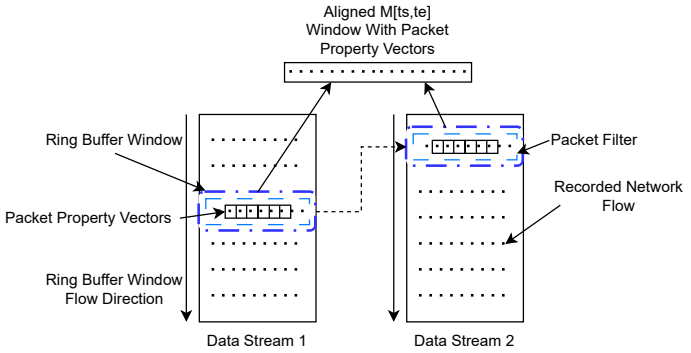


Fig. 1: Data Streams From Data Sources With Ring Buffer Windows for Asynchronous Alignment of Network Streams in Recorded PCAP Files

If the critical threshold is reached, we use the binning technique to reduce the number of beacon vectors (Knuth, 2006) in $\mathcal{M}_{[t_s,t_e]}$ to bins in the sequence. The critical threshold corresponds to the number of vectors found in the $\mathcal{M}_{[t_s,t_e]}$ window. The $\mathcal{M}_{[t_s,t_e]}$ window time intervals were divided into a fixed number of bins of equal width. The number of vector timestamps is counted and falls into each bin based on the amount presented. This way, we obtain a short digital sequence of fixed length from a very long sequence of vector timestamps. The technique involves grouping many separate values into a fixed amount of intervals and summing the number of values that fall into each bin. This is shown lines 16 to line 17.

Line 14 presents all the vector timestamps gathered. We then align the gathered vectors in line 15. Line 19 depicts the sequences that are a part of $\mathcal{M}_{[t_s,t_e]}$ and given to DTW distance calculation. Line 20 returns the distance calculation of DTW and the end-to-end (E2E) time required for computations to be performed. The E2E time calculation is a performance indicator that provides a quantitative comparison between our approach and a classical DTW computation on the same sequences.

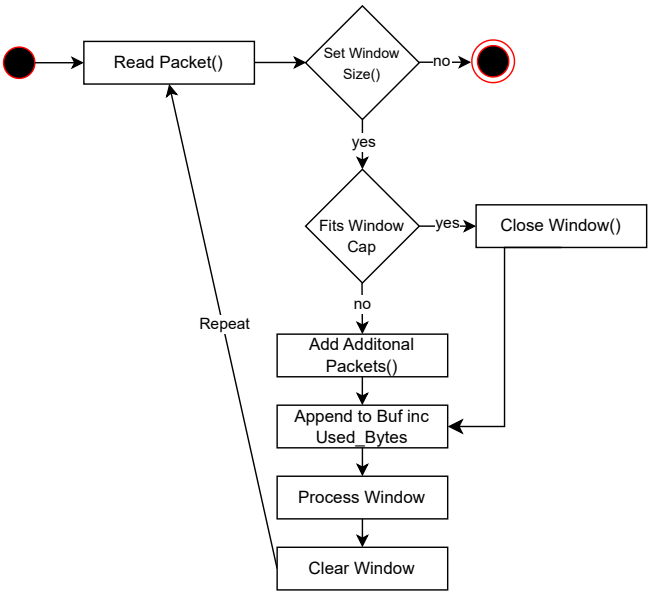


Fig. 2: Representation of Data Stream Flow in the Ring Buffer Window

Fig. 2 shows the detailed flow of the RBW. Packets are added to the window until the end of the window in the data stream. We checked whether the designated buffer window size had reached its designated capacity. If yes, we process the window, clear it afterwards, and add new packets for the second iteration.

In Fig. 3, we show the data flow in the proposed similarity algorithm. If the number

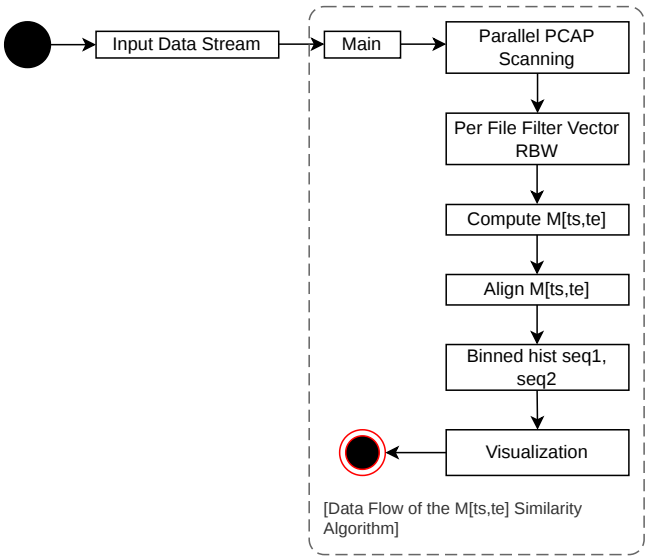


Fig. 3: Stream Data Flow of the Similarity Algorithm From the Parameter Input Until Result Visualisation

of vectors in the data streams is less than a predefined threshold, we continue computing and present the results. The step before the last measures time and sequences in quantity, and in the last step, we present a visual representation of the proposed model results. We also calculate the E2E time to determine the algorithm's runtime from the moment the parameter entry is complete until DTW completes the computations. The experimental visualisation part did not include E2E time monitoring.

The intrinsic properties of the windows in our algorithm give utilization to accelerate DTW functions. These properties are:

1. *Fixed window length.* Because of the limited sequence length, the DTW matrix did not exceed the specified limits.
2. *Fixed byte limit per window.* Windows were not overloaded with large amounts of data.
3. *Window transitions.* We obtained dense but manageable windows, rather than irregular, large chunks of data.
4. *Vector Filtering.* Instead of a $n \times m$ "all in one" matrix, DTW is performed in places where it makes sense to do so.

5. *Window updating.* When the RBW window moves through the data stream, the already filtered, outdated packets are removed from the window without reloading it. This creates a lower cost for the candidate vectors to be added to $\mathcal{M}_{[t_s,t_e]}$.

4 Implementation

In this section, we present several synthetically generated network flow cases with cyber-attack templates. The gathered network flow cases have different parameters and network behaviours, each with its own characteristics, which were recorded in PCAP files for further presentation to our model.

We simulate synthetic network flows in a laboratory environment. For this purpose, specific, sophisticated and known attack templates were chosen to simulate network traffic. The synthetic attack traffic is embedded within background benign traffic.

The sophisticated attack template consists of command-and-control (C2) threat actor virtual machines and target virtual machines, with malware and its beacons deployed on the target machines (Gardiner et al., 2014). The second attack consists of a standard network-mapping (Nmap) port scan, and the third attack is a specific TCP SYN ACK/RST flag-related attack. All the virtual machines for the experimental cases were part of the `Proxmox` virtual environment created in our laboratory (Oleksiuk and Oleksiuk, 2021). `Proxmox` supports different types of hypervisors. These hypervisors allowed us to create virtual machines for our experimental bedrock. For network flow collection across different virtual machines, we use a centralised system with `RabbitMQ` as the broker, which is also a virtual machine in the `Proxmox` environment. `RabbitMQ` has distributed network monitors deployed on every virtual machine in the network. These network monitors send network flow data to our broker, which stores it in PCAP files.

For the first attack case, the network setup in the `Proxmox` environment consisted of 9 virtual machines: 4 C2 servers and 5 infected clients. In addition to beacons, five infected machines have different types of malware installed. Overall, four simulated teams, each with two to three machines, were distributed across our laboratory. The result was 18 PCAP files, ranging from 5 to 36 megabytes, over two iteration periods while gathering network flow. C2 activity and other activities in the network were completed over a 2.5-hour period, during 2 recording sessions. All network flows were gathered from both actor perspectives. All network traffic was routed via an unencrypted channel.

Different attributes and vectors were chosen from the first attack case, which identified beacon vectors in the PCAP files. These attributes are dispersed between the PCAP files and represent different C2 beacon implementations. Table 2 lists different attributes with their vectors and protocols that have been recorded in the PCAP files. The attribute names follow the standard naming convention used by `Wireshark`, a network protocol analyser software.

All recorded PCAP files, based on PCAP attribute vectors, have distinct numbers of packets identified as heartbeat packets. These heartbeats corresponded to the beacon vectors recorded in the PCAP file.

Table 2: Beacon Attribute and Their Values in PCAP Files With Volumetric Statistics

Protocol	Attribute	Beacon Vector	Total Amount	Mean t [s]	Median t [s]
HTTP	http.request.uri	/beacon	122	55.50	0
HTTP	text	//4A	59	52.10	3.06
TLS 1.3	tls.handshake.ja3	78f0dc5...	1484	7.36	0.05
TCP	tcp.dstport	8888	4282	3.25	1.02
TCP	tcp.dstport	5000	4109	3.90	0
TCP	tcp.flags	SYN	9501	1.72	0.02

Every heartbeat-type communication observed in this study was condensed into six rows. Each row corresponding to a distinct beacon heartbeat amount differs across C2 implementations on different machines. For each distinct type, we report the total number of heartbeats. In addition, we provide two additional volumetric indicators. Average and the median inter-arrival times between consecutive heartbeats.

Every beacon heartbeat shown in the table can be characterised according to different protocols and specific measurement counts. These metrics are sufficient to determine several different behaviours of C2 channels and their respective environments in the network stream.

In some cases, the median is zero. This could occur because a beacon sends multiple packets at once, and both are recorded at almost the same timestamp. This implies that the network flow is gathered too densely. In our experiments, we did not change the vector density.

The second and third attack cases require only 2 virtual machines for the experiments. That is a threat actor and target machines. Both cases have network flow recording times of 5-10 minutes from a heterogeneous perspective, and the PCAP file sizes are 5-15 megabytes.

In the first case, the gathered network traffic spans a larger time frame than in the second or third cases. Therefore, in our study, we also use large network traffic PCAP files recorded at different times and from various sources on the internet as the fourth case. Specifically for the fourth case experiments, we use Mid-Atlantic Collegiate Cyber Defense Competition (MACCDC) datasets, which are public PCAP files captured and published at network research vendor (NETRESEC) (NETRESEC, 2015).

Overall, network capacity in our simulated network environment for the recorded cases two and three is lower than the standard I/O capacity due to firewall rules and other policies of our institution. Otherwise, the network flow data gathered from our laboratory's setup exhibits distinct behaviour during simulated cyber-attacks. In the first case, internet traffic was generated to simulate a scenario in which malware is downloaded to the virtual machines, and beacon activity was generated between the threat actor and the target machines. The beacons listed in Table 2 have different spark times in PCAP files, suggesting possible heartbeat activity.

Also, in Table 3, beacon values define a specific place in the packet structure where they appear. These attributes, with their values, represent network flow characteristics extracted at different protocol layers. For transport level characteristics

`tcp.dstport`, `tcp.flags` that represent connection behaviour, service targeting and session state. These fields are explicit in the packet structure. Application-level characteristics, such as `http.request.uri`, are only visible in decrypted traffic and can be interpreted as a requested resource. Cryptographic characteristic `tls.handshake.ja3` is a client fingerprint characteristic that is not explicitly sent, which can be considered as a sophisticated attack pattern or an anomaly. The `text` is a heuristic characteristic that indicates a human-readable payload. These packet characteristics are observable attributes of beacons extracted from different protocol layers and describe the communication behaviour in the PCAP files.

Table 3: Beacon Attribute Values in Packet Structure Placement

Place in Packet Structure	**Attribute**	**Beacon Vector**
TCP Payload (HTTP Request)	`http.request.uri`	`/beacon`
TCP Payload (HTTP Request)	`text`	`//4A`
TCP Payload (TLS ClientHello)	`tls.handshake.ja3`	`78f0dc5...`
TCP Header	`tcp.dstport`	`8888`
TCP Header	`tcp.dstport`	`5000`
TCP Header	`tcp.flags`	`SYN`

The second and third simulated attack templates generate relatively static flows that mimic usual communication between network machines based on internet protocol logic. In the second case, the port scan activity is characterised by a high number of short-lived flows originating from a single source and targeting multiple destination ports. The third attack template generates a specific number of flags that could be considered beacons for malicious activity between virtual machines. The second and third attack templates also use TCP header flags and destination ports as primary sources of identification within the packet structure. Other synthetic attack characteristics include asymmetric packet distribution (case two), elevated connection rates in cases one and two, and short-lived flows in cases one and three.

The external PCAP files lack documentation of specific traffic patterns. Consequently, our model is designed to be incident-agnostic. The model does not rely on knowledge of events or attack templates used in the network flow. Instead, our proposed model searches for similar patterns in heterogeneous flows and aligns them by identifying and aligning vectors derived from the surrounding noise in the network flow. For this reason, our prototype presents a list of possible attributes and vectors as input to the proposed model implementation. This enrichment enables us to adopt a more flexible approach to network flows and better assess the proposed model's validity.

5 Experiments and Results

In this section, we present the experimental results obtained when implementing the proposed model. Overall, experiments were performed using 20 data sets across four

cases. Experiments were performed using different input parameters. In addition, different RBW sizes were used to process the datasets, which were also a part of the input parameters. The RBW sizes per example were $16, 32, 64$, and 128 KB. In the presented results, we show a correlation between classical DTW and our model in terms of the computational time required for both sequences. The sequence lengths are shown on the x-axis, and the program runtime is shown on the y-axis.

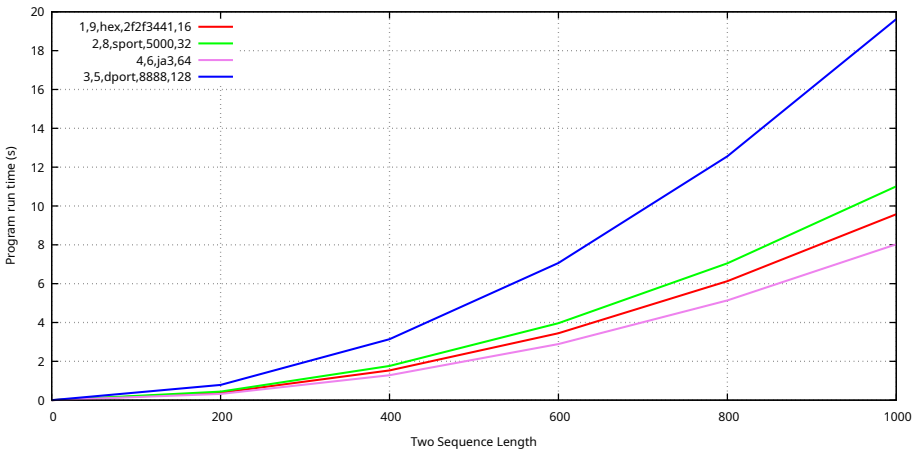
All experiments were conducted on a physical computer that had an AMD Ryzen 9 processor, 64GB of RAM, and Linux Mint 22.1 LTS at the time of the experiments. All of the experiments had a conditional number of input parameters chosen:

- PCAP file selection.
- Packet property.
- Packet property vector.
- Different size of RBW. Which are $16, 32, 64$, and 128 KB in size, respectively.
- Bins count.
- Packet property vector threshold.

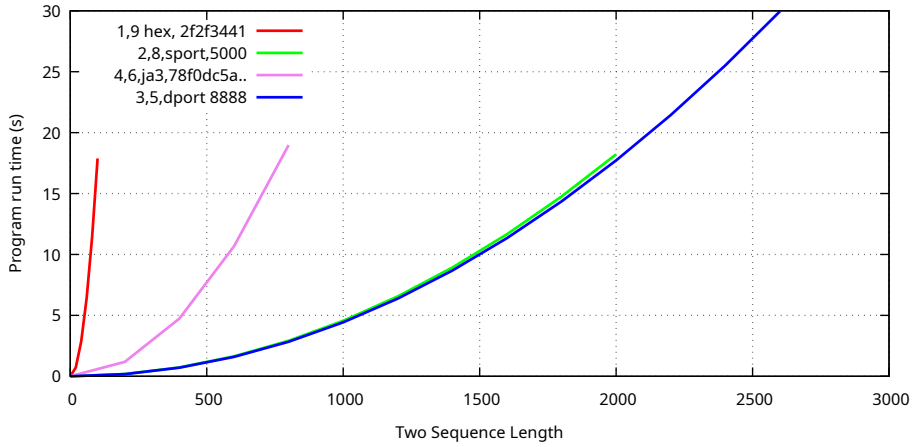
Fig. 4a shows the results from case one, experiment A, where we used the PCAP files that were recorded in our laboratory's network environment. The first two numbers in the figures legend are the PCAP files from the recorded sets we chose based on the C2 environment in the network for the experiments, where one PCAP file is the C2 attacker actor server machine, and the other is the target machine. The third parameter corresponds to the packet property, and the fourth is the packet property vector. The last number in the legend represents the RBW size. The sequence length does not exceed 1000, which is the sum of the predefined bin count of 500 per sequence. Depending on the number of found sequences, PCAP binning was applied to every experiment iteration. Different RBW sizes had an impact. The smaller the RBW size, the faster the computational time. The other parameters did not have a significant effect on the vector amount. The fixed amount of RBW in the last iteration, which measured 128 in size, had the highest computational time of 20 seconds.

For each PCAP file, the vector timestamps were prepared as constructed sequences of fixed-length histograms from the aligned time window $\mathcal{M}_{[t_s,t_e]}$. This reduces the sequences and allows DTW to be applied afterwards. Using binning allows DTW to not count millions of possible vectors. The vector values were compressed into 500 bins per sequence. Then the DTW distance is processed for both sequences every 500 bins.

This allows us to work with the binned values rather than with every vector, which could be a sequence of counting in millions. The DTW distance calculates the minimum alignment distance between sequences, but does not return the full alignment or the entire matrix. Distance measurement is a good benchmark method that returns a single value, the optimal DTW distance between two sequences, for a vector (Meert et al., 2020). The distance measurement showed indifference to the accumulation of sequences. In addition, this method is the most cost-efficient compared to a full warping path or sequence alignment due to lower computational resource requirements. The distance measurement allows the vectors to be pulled slightly towards each other and aligned in time. This measurement allows us to evaluate the dynamics in form, that is whether the vectors rhythm over $\mathcal{M}_{[t_s,t_e]}$ changes similarly over time.



(a)



(b)

Fig. 4: Case One, Experiment A: Results from PCAP Files Recorded in a Controlled Network Environment for the Proposed Model With Different RBW Sizes and for Classical DTW Distance Measurement.

The binning technique aggregates all the vectors in ts, te into 500 slots. After this step, both sequences had a length of 500 bins. Then, the DTW distance computes a 500×500 matrix instead of the $all \times all$ matrix from both sequences, even if the sequences are filtered.

Then DTW measures not the individual packet distances but the similarity of two time profiles of aligned $\mathcal{M}_{[t_s,t_e]}$. We evaluated the macro-dynamics of the alignment, how much, and when the activity was spotted. It is possible that millions of vectors are influenced by the number of bins, and DTW works with this aggregated data, making it

fast and meaningful for measuring macro-similarity. For DTW distance measurement, we used a Python library (Meert et al., 2020).

Fig. 4b in case one, experiment A had the same input parameters as for our model for every iteration. Here, we use classical DTW with our generated PCAP files in a controlled network environment. We can see the number of vectors found per experimental iteration and the time required to process them. The vector amount between the two sequences is the same as in the experiment with our proposed model in Fig. 4a, but the computational time required is higher.

Table 4 shows the input parameters for PCAP files that were recorded with synthetic cyber-attacks in our laboratory's network environment.

Table 4: Case One, Experiment A Input Parameters Chosen for the PCAP Files That Were Generated in Our Laboratories Controlled Network Environment

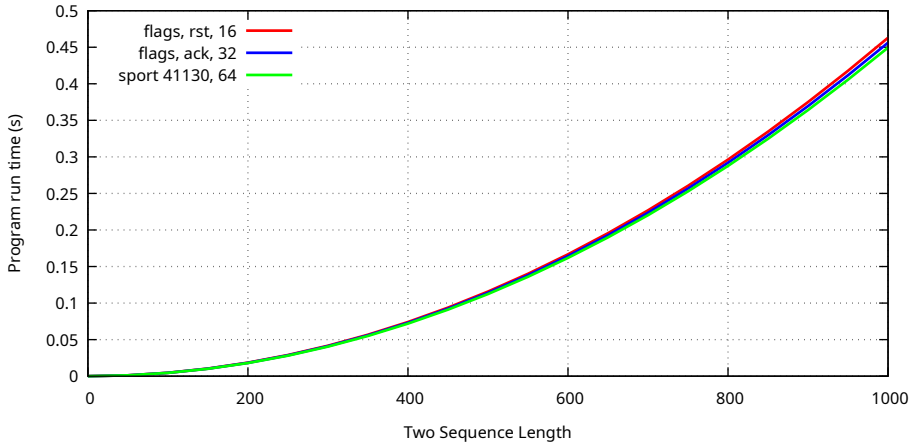
PCAP Files	**Attribute**	**Beacon Vector**	RBW **Size**
2 PCAP	`hex`	`2f2f3441`	16KB
2 PCAP	`Source Port`	`5000`	32KB
2 PCAP	`Ja3`	`78f0dc5...`	64KB
2 PCAP	`Destination Port`	`8888`	128KB

In case two, experiment B, there is a noticeable decrease in computational time required by our approach and the classical DTW implementation. In this experiment, the amount of packets is slightly higher than in case one, but the effect of time reduction for computational time is still evident. Also, the runtime in Fig. 5a and Fig. 5c increases linearly. Furthermore, comparing the classical DTW to our model's result curves, which nearly coincide over the same time needed for computations, suggests that the RBW dependency has no overhead and that vector density per-bin count has minimal effect under the provided parameters.

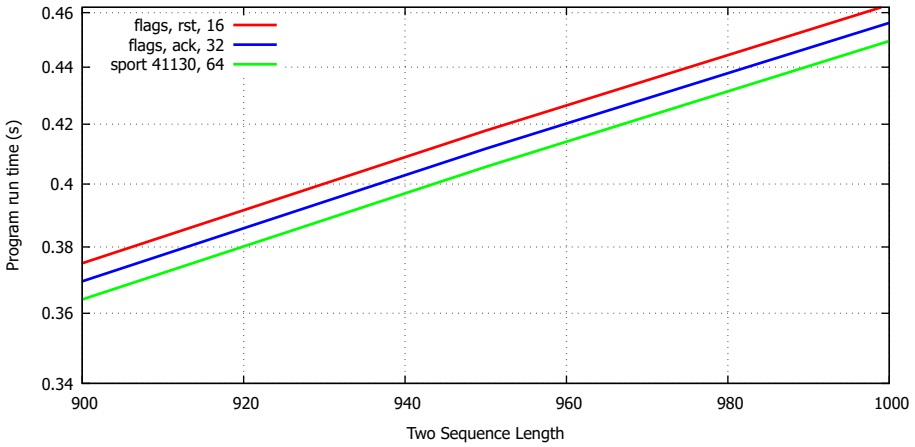
In Fig. 5b, we show the scaled representation of case two, experiment B to visualise the proximity of computational time. The results based on vector count suggest that the time required to perform computations was nearly the same, with a very small offset per iteration.

Across the entire range of the classical DTW in Fig. 5c, the flags RST vector alignment incurs the highest time, the source port is intermediate, and the flag ACK alignment is the lowest for the classical DTW. The bigger the amount of vectors found between the PCAP files, the higher the time for computations needed.

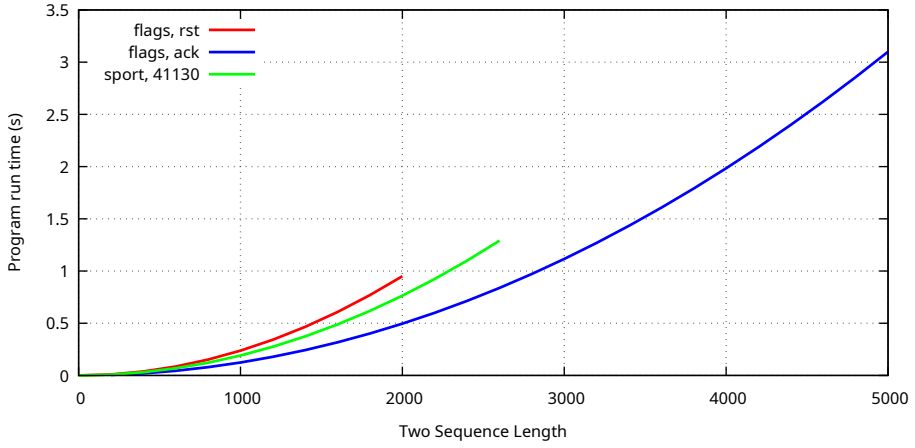
For case three, experiment C in Fig. 6a and classical DTW in Fig. 6c based on Nmap port scan in the PCAP files, it is evident that mostly two specific attributes and their vectors are suitable for checking. That is the SYN and RST flags from both threat actor and target actor perspectives. The overall selection of the parameters in the provided Fig. 6a of our model has the same RBW rule dependency, and vector density does not affect the overall performance. At the same time, the amount needed to process the vectors is dramatically different in comparison.



(a)



(b)



(c)

Fig. 5: Case Two, Experiment B: Results from PCAP Files Recorded in a Controlled Network Environment for a Specific Flag-Related Attack Scenario, including the proposed model with different RBW sizes, scaled results, and classical DTW distance measurement.

In Fig. 6b, a scaled version of the presented case three, experiment C representation is shown. The computation time here is also similar, but based on the RBW size, we can determine that the time required to efficiently process the vectors is determined by the minimised RBW value.

For both Fig. 6a and Fig. 6c the curves nearly coincide over the evaluated ranges. This is evident because the number of flags is nearly the same in both PCAP files, which implies that the attacker sends a specific number of packets to test the ports and receives a specific number of denied RST packets for the scanned ports.

Case four, experiment D used compressed PCAP files gathered from NETRESEC (NETRESEC, 2015). The compressed file size ranged from 300 to 400 MB, and the total packet count was millions. Specifically, the average number of packets in the compressed PCAP files was 10 million. As shown in Fig. 7a, we used different RBW window sizes to determine the computational efficiency of the proposed model. The two sequence lengths on the x-axis are the predefined bin counts, each set to 500. The bin count per sequence was not changed during the experiments. Because the packet count exceeded the threshold amount, Fig. 7a shows the length of both sequences and the time it takes for the experiment to run. Time monitoring began immediately after parameter input and ended before the figure of the experimental results was created. This means that our model's time is monitored throughout all phases of computation.

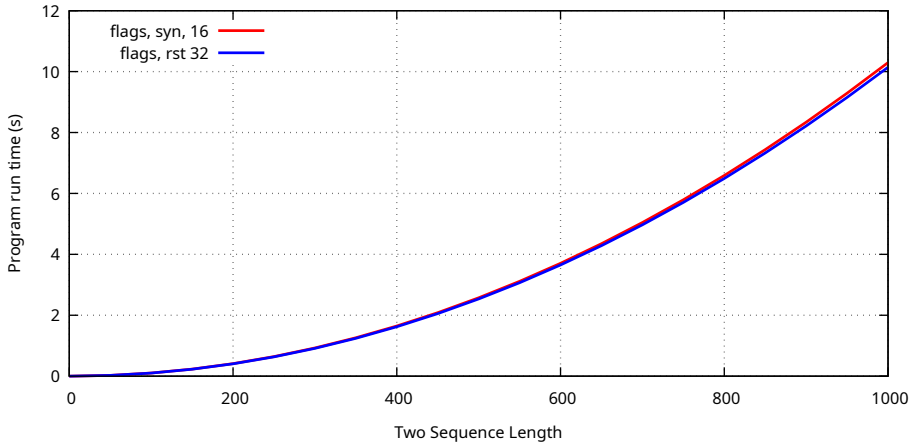
Based on the number of vectors, which varied across experimental iterations, the computation time ranged from 1,000 to 1,400 seconds for 10 million records per file. The computational time increased with the vector count. In Fig. 7a, experiments with effectiveness time are shown. The vector amount ranged from 1000 to 4.5 million per sequence. A slight difference in computational time is observed for RBW, but the major difference lies in the vector count. Table 5 lists the input parameters chosen for case four, experiment D.

Table 5: Case Four, Experiment D Input Parameters Chosen for Extensive PCAP Files From the Internet

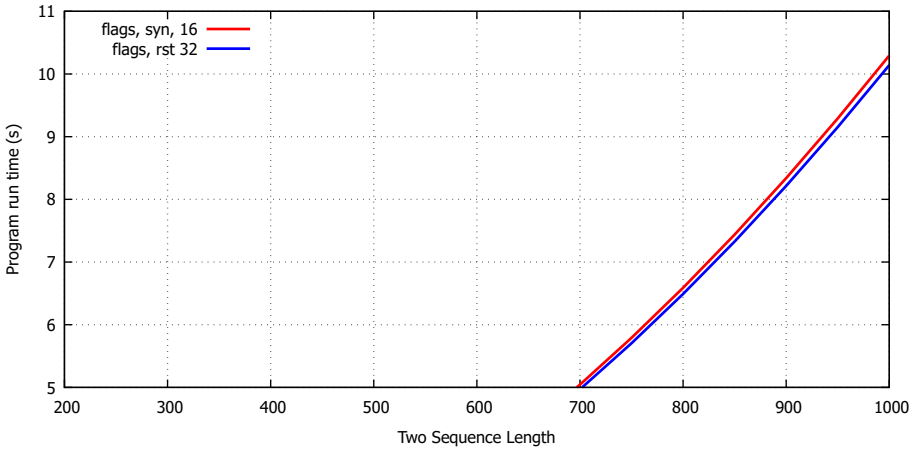
PCAP Files	**Attribute**	**Beacon Vector**	**RBW Size**
2 PCAP.gz	`Source Port`	`443`	16KB
2 PCAP.gz	`TCP Flag`	`SYN`	32KB
2 PCAP.gz	`TCP Flag`	`PSH`	64KB
2 PCAP.gz	`Destination Port`	`22`	128KB

Case four, experiment D, as shown in Fig. 7b, had the same input parameters for every iteration. Only two of the four iterations were feasible due to the massive vector count per experiment. The time and amount required to perform the computations exceed those of the previous experiments by a significant margin. The number of vectors was also extensive.

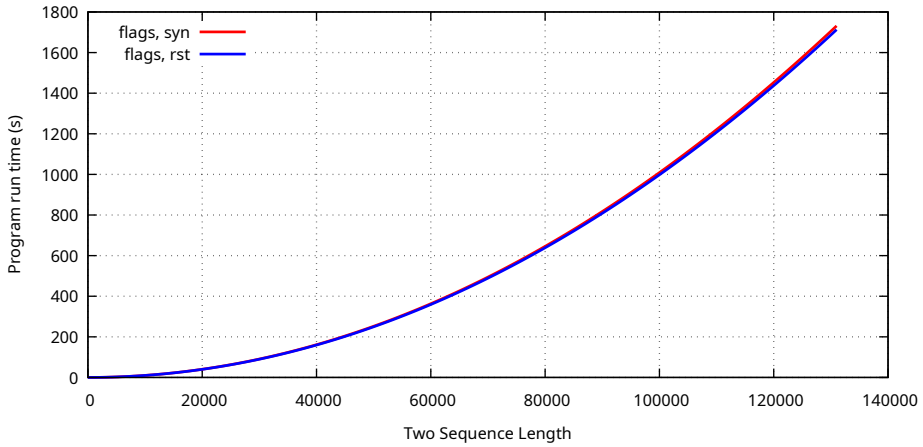
In every experiment, the time increased almost linearly along the x-axis. The indicated linear scaling in the graphs under the provided parametrised conditions showed no quadratic behaviour.



(a)

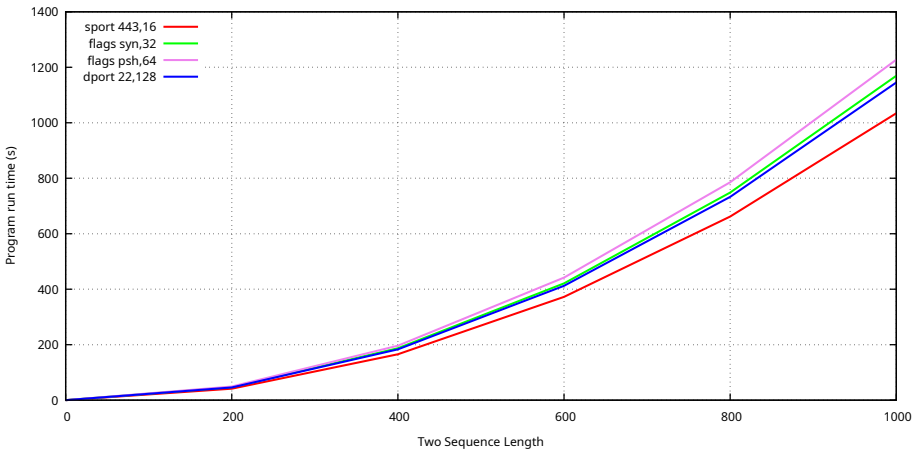


(b)

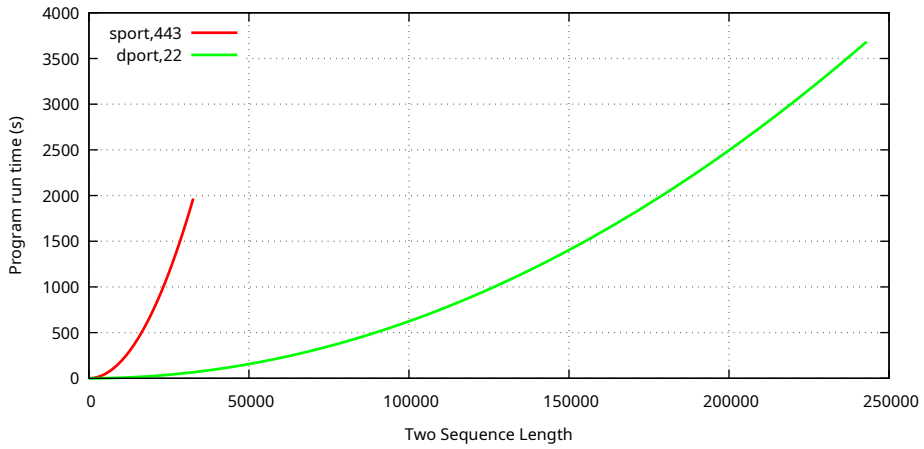


(c)

Fig. 6: Case Three, Experiment C: Results from PCAP Files Recorded in a Controlled Network Environment for the Nmap Port Scan Attack Scenario, including the proposed model With different RBW sizes, scaled results, and classical DTW distance measurement.



(a)



(b)

Fig. 7: Case Four, Experiment D: Results from Extensive Compressed PCAP Files from the Internet for the Proposed Model With Different RBW Sizes and for Classical DTW Distance Measurement.

In Fig. 8, we present a combined representation from the extensive PCAP files, which shows our model and the classical DTW time efficiency. The representation of our model using the binning technique will never exceed the threshold constant compared with the classical model. This was particularly evident in the combined efficiency of the two models.

From DTW perspective we get a 500×500 matrix not depending on the sequence lengths in aligned $\mathcal{M}_{[t_s,t_e]}$. For classical DTW without window properties, we observed a significant increase in time across the provided experimental iterations. Classical

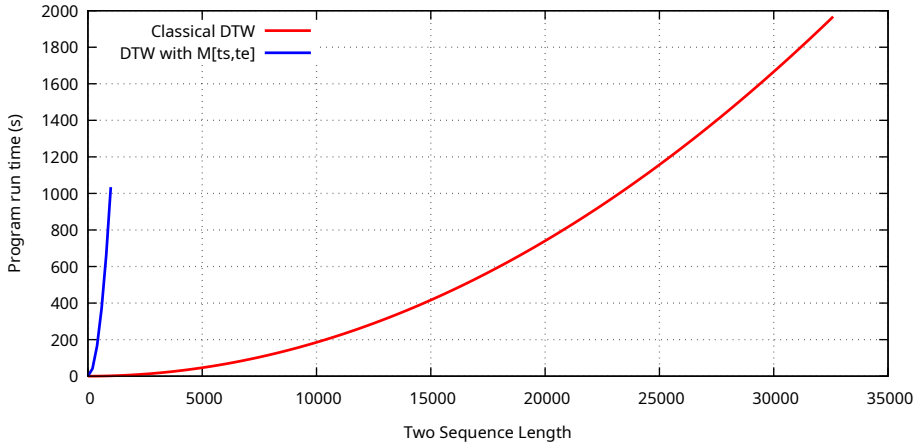


Fig. 8: Combined Results of Classical DTW and Our Proposed Model With Extensive PCAP Files From the Internet

DTW can match our model's computational time when the number of vectors is small. Our model's computational time did not exceed a significant margin, whereas classical DTW across iterations took much longer. The DTW window properties showed a significant reduction in computational time. Adding window properties before the DTW computation minimises the computation time.

In summary, the experimental results demonstrate that the proposed pipeline can systematically extract event-based network-flow vectors, align and transform them into temporal representations, and compare traffic patterns using sequence-alignment methods.

6 Discussion

Data filtering is an important process that must be addressed. For PCAP files that can accumulate large amounts of data quickly, we must eliminate as many unnecessary parameters as possible. Subsequently, data normalisation is also required to determine even greater computational capabilities. This results in the application of a specific technique that requires additional knowledge and experimentation. The gathered results indicate that the proposed model in its vector extraction pipeline can process network-flow data in a structured and repeatable manner. The observed runtime behaviour suggests that the computational cost is largely influenced by sequence length, binning configuration, and the selected matching strategy.

DTW can align similar behaviours even when they are shifted in time. Taking DTW as a baseline for comparing traffic patterns with our model is useful for C2-like and other communication, where beaconing, repeated, or other actions may not occur at identical timestamps. Compared with existing DTW variants, the proposed model should be positioned less as a new DTW algorithm and more as a network-flow

analysis pipeline that uses DTW as an interpretable baseline for pattern matching. The proposed approach is designed for continuous heterogeneous network traffic, where the main contribution is not only the alignment itself, but also the extraction of flow-context vectors such as flags, ports, JA3 indicators and others. Therefore, DTW variants such as FastDTW, UCR-DTW, SUCR-DTW, ODTW, EventDTW, SparseDTW, SSDTW, and FlexDTW can be treated as computational or methodological baselines, while the proposed model extends the problem toward context-aware streaming cyber-attack
detection.

DTW has a quadratic complexity. Extensive data flows that were minimised make DTW more stable and cheap, even when $\mathcal{M}_{[t_s,t_e]}$ window has hundreds of thousands of vectors aligned. The downside is that we can lose alignment owing to a small bin count per sequence. This will achieve only a macro-dynamical evaluation of the cyber-attack pattern over time. For micro-dynamical analysis in alignment, we could utilise the bin count, which should be dynamic, or use other patterns to evaluate the clarity of the patterns. For small vector sets, the classical DTW algorithm is sufficient for detecting cyber-attacks. For sophisticated cyber-attacks and extensive network flows, especially in today's network environments, DTW is slow to implement.

Limitations

For network flow gathering, we used a centralised approach with a network broker. This may become a single point of failure if the centralised system breaks without any backup. Furthermore, a significant amount of physical disk space is required to accommodate consistent network flow to the network broker provider's machine. Fast disk replacement or attachment is not possible due to additional configuration requirements and costs. Incoming network traffic can overwhelm the broker machine, particularly when the network's heterogeneous. This may cause the network broker to experience data leakage. Because of this, not all network flows would be gathered by the network broker due to the massive network flow generated previously. In addition, the PCAP format can be addressed. For our experiments, we used the standard PCAP format. A new generation packet capture (PCAP-NG) format can be used. The new generation structure allows us to gather more metadata and timestamps with greater precision than the old standard. Additionally, the new generation can store data in blocks and process multiple network cards in the same PCAP file. Although this is a take for the future, technical implementation capabilities have not yet been discussed. Currently, we do not rely solely on metadata. We strive to utilise the full packet length. This is because of the significant amount of data already collected, which requires analysis in its own right.

The method we use in this study for cybersecurity attack detection is not part of the standard system behaviour documented in XDR, EDR, and related specifications. These systems use metadata types such as packet behavioural recognition or utilise standard behaviours, such as log scanning with additional attention to service or system sign-ins, kernel-level inspection, database attack tags, and others. Deep packet inspection has not been utilised in standard systems. Therefore, these systems cannot perform large-scale network flow analysis across heterogeneous networks to detect sophisticated

cyber-attacks. Furthermore, encrypted traffic limits deep packet inspection, forcing the model to rely on unencrypted data that may be less discriminative.

Also, in evolving network streams, an ML approach would be a beneficial solution for detecting sophisticated cyber-attacks, however, these methods face their own challenges. Model theft, data poisoning, and model overfilling for extensive network flows are today's problems that will be addressed in the near future.

7 Conclusion

This research advances the computational efficiency of network flow alignment algorithms, particularly for detecting sophisticated cyber-attacks across both small and large network data flows. The research conducted in this study provides a fundamental understanding of asynchronous evidence-based assessment and analysis of network traffic for cybersecurity purposes.

Providing an attack template scenario between virtual machines and a centralised broker for network flow gathering enables us to model the data and conduct experiments for the proposed approach. Recorded network traffic and other, more extensive traffic in PCAP format from internet sources were used as datasets for our model experiments. The experimental results showed a promising efficiency boost over the classical DTW algorithm.

Overall, our research advances network traffic analysis for cyber-attack detection by presenting an approach that pushes the boundaries of pattern recognition across small and large-scale datasets.

8 Future Work

The quadratic problem of the DTW algorithm is reduced to a linear one using sliding-window properties, thereby reducing the problem's complexity. The spectrum of properties enables a dynamic approach to use the proposed model for many different cyber-attack scenarios. However, this method has limitations. It is difficult to identify potentially heavy, sophisticated attacks that are masked by extensive volumes of network traffic, even if they originate from non-heterogeneous flows. This comes down to preprocessing and processing relevant data in the PCAP files that are not always perfectly recorded. The definitive distinction comes from the computational power and time required to process the data. For future reference, possible ML integration should be considered.

One of the other issues that we will investigate is a deep asynchronous content-based data assessment as a potential scope for future research. This comes down to managing extensive network flows to the point where we can determine the specifics needed to identify potential, heavily loaded, sophisticated cyber-attacks.

Hierarchical indexing is another point of consideration. Using this method, we aim to identify the parameters that define relationships between groups of data, enabling us to conduct a more complete analysis across different packet layers.

Also, we plan to analyse data flows in more realistic and diverse scenarios rather than relying solely on synthetic datasets. This will allow us to further evaluate and

validate the proposed attack detection models in heterogeneous and dynamic traffic environments. Real-world data flows present more realistic conditions, while synthetic data enables straightforward identification of patterns. This should provide a stronger basis for the applicability of the model.

References

Al-Naymat, G., Chawla, S., Taheri, J. (2012). Sparsedtw: A novel approach to speed up dynamic time warping, *arXiv preprint arXiv:1201.2969* .

Argyraki, K., Cheriton, D. (2005). Network capabilities: The good, the bad and the ugly, *ACM HotNets-IV* **139**, 140.

Beddoe, M. A. (2004). Network protocol analysis using bioinformatics algorithms, *Toorcon* **26**(6), 1095–1098.

Bükey, I., Zhang, J., Tsai, T. (2023). Flexdtw: Dynamic time warping with flexible boundary conditions, ISMIR.

Clotet, X., Moyano, J., León, G. (2018). A real-time anomaly-based ids for cyber-attack detection at the industrial process level of critical infrastructures, *International Journal of Critical Infrastructure Protection* **23**, 11–20.

Diab, D. M., AsSadhan, B., Binsalleeh, H., Lambotharan, S., Kyriakopoulos, K. G., Ghafir, I. (2019). Anomaly detection using dynamic time warping, *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, IEEE, pp. 193–198.

Dobbelaere, P., Esmaili, K. S. (2017). Kafka versus rabbitmq: A comparative study of two industry reference publish/subscribe implementations: Industry paper, *Proceedings of the 11th ACM international conference on distributed and event-based systems*, pp. 227–238.

Gardiner, J., Cova, M., Nagaraja, S. (2014). Command & control: Understanding, denying and detecting-a review of malware c2 techniques, detection and defences, *arXiv preprint arXiv:1408.1136* .

Giao, B. C., Anh, D. T. (2016). Similarity search for numerous patterns over multiple time series streams under dynamic time warping which supports data normalization, *Vietnam Journal of Computer Science* **3**(3), 181–196.

Guo, F., Zou, F., Luo, S., Liao, L., Wu, J., Yu, X., Zhang, C. (2022). The fast detection of abnormal etc data based on an improved dtw algorithm, *Electronics* **11**(13), 1981.

Heenan, R., Moradpoor, N. (2016). Introduction to security onion, *The First Post Graduate Cyber Security Symposium*.

JIANG, W., TIAN, Z., CAI, C., GONG, B. (2014). Bottleneck analysis for data acquisition in high-speed network traffic monitoring, *NETWORK TECHNOLOGY AND APPLICATION* .

Jiang, Y., Qi, Y., Wang, W. K., Bent, B., Avram, R., Olgin, J., Dunn, J. (2020). Eventdtw: An improved dynamic time warping algorithm for aligning biomedical signals of nonuniform sampling frequencies, *Sensors* **20**(9), 2700.

Jones, B., Luxenberg, S., McGrath, D., Trampert, P., Weldon, J. (2011). Rabbitmq performance and scalability analysis, *project on CS* **4284**.

Kalinin, M. O., Krundyshev, V. M. (2021). Analysis of a huge amount of network traffic based on quantum machine learning, *Automatic Control and Computer Sciences* **55**(8), 1165–1174.

Khan, M., Ghafoor, L. (2024). Adversarial machine learning in the context of network security: Challenges and solutions, *Journal of Computational Intelligence and Robotics* **4**(1), 51–63.

Knuth, K. H. (2006). Optimal data-based binning for histograms, *arXiv preprint physics/0605197* .

Komisarek, M., Pawlicki, M., Kozik, R., Choras, M. (2021). Machine learning based approach to anomaly and cyberattack detection in streamed network traffic data., *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* **12**(1), 3–19.

Kotov, M. S., Tolyupa, S. V., Nakonechnyi, V. S. (2024). Method of building local area network simulation based on amqp and its support protocols suite, *Telekomunikatsiini ta Informatsiini Tekhnologii* (3), 102–119. in Ukrainian.

Kremer, H., Günnemann, S., Ivanescu, A.-M., Assent, I., Seidl, T. (2011). Efficient processing of multiple DTW queries in time series databases, *International Conference on Scientific and Statistical Database Management*, Springer, pp. 150–167.

Krinickij, V., Bukauskas, L. (2024). Asynchronous record alignment of network flows for incident detection and reconstruction, *European Conference on Cyber Warfare and Security*, Vol. 23, pp. 249–256.

Kumar, K. M. K., Reddy, M. V. S., Ullas, K. et al. (2025). Distributed intrusion detection system using kafka and spark streaming, *2025 International Conference on Visual Analytics and Data Visualization (ICVADV)*, IEEE, pp. 302–309.

Kure, H. I., Sarkar, P., Ndanusa, A. B., Nwajana, A. O. (2025). Detecting and preventing data poisoning attacks on ai models, *arXiv preprint arXiv:2503.09302* .

Lei, H., Govindaraju, V. (2004). Direct image matching by dynamic warping, *2004 Conference on Computer Vision and Pattern Recognition Workshop*, IEEE, pp. 76–76.

Li, C., Chen, G. (2004). Synchronization in general complex dynamical networks with coupling delays, *Physica A: Statistical Mechanics and its Applications* **343**, 263–278.

Liang, S., Peng, X., Qi, H. J., Zonouz, S., Beyah, R. (2021). A practical side-channel based intrusion detection system for additive manufacturing systems, *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, IEEE, pp. 1075–1087.

Likic, V. (2008). The needleman-wunsch algorithm for sequence alignment, *Lecture given at the 7th Melbourne Bioinformatics Course, Bi021 Molecular Science and Biotechnology Institute, University of Melbourne* pp. 1–46.

Lou, Y., Ao, H., Dong, Y. (2015). Improvement of dynamic time warping (dtw) algorithm, *2015 14th international symposium on distributed computing and applications for business engineering and science (DCABES)*, IEEE, pp. 384–387.

Maatkamp, M., van Delden, M., LeKhac, N. A. (2016). Unidirectional secure information transfer via rabbitmq, *arXiv preprint arXiv:1602.07467* .

Macrae, R., Dixon, S. (2010). Accurate real-time windowed time warping., *ISMIR*, pp. 423–428.

Maramreddy, Y. R., Muppavaram, K. (2024). Detecting and mitigating data poisoning attacks in machine learning: A weighted average approach, *Engineering, Technology & Applied Science Research* **14**(4), 15505–15509.

Meert, W., Hendrickx, K., Van Craenendonck, T. (2020). wannesm/dtaidistance v2. 0.0, *Zenodo* .

Mehta, K., Sood, V. M., Singh, J., Sharma, D., Chhabra, P. (2022). Securing cyber infrastructure of iot-based networks using ai and ml, *2022 Seventh International Conference on Parallel, Distributed and Grid Computing (PDGC)*, IEEE, pp. 378–383.

Mulinka, P., Casas, P. (2018). Adaptive network security through stream machine learning, *Proceedings of the ACM SIGCOMM 2018 Conference on Posters and Demos*, pp. 4–5.

Müller, M. (2007). Dynamic time warping, *Information retrieval for music and motion* pp. 69–84.

NETRESEC, A. (2015). Publicly available pcap files, *Retrieved January* **14**, 2017. `https://www.netresec.com/?page=MACCDC`

Oleksiuk, V. P., Oleksiuk, O. R. (2021). The practice of developing the academic cloud using the proxmox ve platform, *Educational Technology Quarterly* **2021**(4), 605–616.

Oregi, I., Pérez, A., Del Ser, J., Lozano, J. A. (2017). On-line dynamic time warping for streaming time series, *Joint european conference on machine learning and knowledge discovery in databases*, Springer, pp. 591–605.

Pandey, S. (2023). Cybersecurity trends and challenges, *INTERANTIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT* . `https://api.semanticscholar.org/CorpusID:261023858`

Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping, *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 262–270.

Ranjan, N., Kancherla, D. B., Ravuri, P. et al. (2023). The development of new cyber security and networking solutions using artificial intelligence and machine learning, *Journal of Informatics Education and Research* **3**(2).

Rivera, J. J. D., Khan, T. A., Akbar, W., Afaq, M., Song, W.-C. (2021). An ml based anomaly detection system in real-time data streams, *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, IEEE, pp. 1329–1334.

Sadasivan, H., Stiffler, D., Tirumala, A., Israeli, J., Narayanasamy, S. (2023). Accelerated dynamic time warping on gpu for selective nanopore sequencing, *BioRxiv* pp. 2023–03.

Sakoe, H., Chiba, S. (2003). Dynamic programming algorithm optimization for spoken word recognition, *IEEE transactions on acoustics, speech, and signal processing* **26**(1), 43–49.

Sharabiani, A., Darabi, H., Harford, S., Douzali, E., Karim, F., Johnson, H., Chen, S. (2018). Asymptotic dynamic time warping calculation with utilizing value repetition, *Knowledge and Information Systems* **57**(2), 359–388.

Srivastava, M., Kaushik, A., Loughran, R., McDaid, K. (2024). Data poisoning attacks in the training phase of machine learning models: A review.

Tralie, C., Dempsey, E. (2020). Exact, parallelizable dynamic time warping alignment with linear memory. arxiv, *arXiv preprint arXiv:2008.02734* .

Tsai, T. (2021). Segmental dtw: A parallelizable alternative to dynamic time warping, *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp. 106–110.

Ulmer, A., Sessler, D., Kohlhammer, J. (2019). Netcapvis: Web-based progressive visual analytics for network packet captures, *2019 IEEE Symposium on Visualization for Cyber Security (VizSec)*, IEEE, pp. 1–10.

Wu, R., Keogh, E. J. (2020). Fastdtw is approximate and generally slower than the algorithm it approximates, *IEEE Transactions on Knowledge and Data Engineering* **34**(8), 3779–3785.

Yang, D., Shaw, T., Tsai, T. (2022). A study of parallelizable alternatives to dynamic time warping for aligning long sequences, *IEEE/ACM Transactions on Audio, Speech, and Language Processing* **30**, 2117–2127.

Received March 26, 2026 , revised May 25, 2026, accepted June 9, 2026