Visualising of cyber crime data by Communication Structured Acyclic Nets

Mohammed Alahmadi

Supervisor: Prof. Maciej Koutny

School of Computing

Newcastle University

This dissertation is submitted for the degree of

Doctor of Philosophy

Monday 2nd September, 2024

I would like to dedicate this thesis to My World:

loving parent

Beloved wife and Daughter

Brothers and Sisters
and my Super Supervisor Maciej.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 50,000 words including bibliography, footnotes, tables and equations and has 54 figures.

Mohammed Alahmadi

Monday 2nd September, 2024

Abstract

Communication Structured Acyclic Nets (CSA-nets) are a Petri net-based formalism used to represent the behaviour of Complex Evolving Systems (CES). CSA-nets, comprising sets of acyclic nets, are suitable tools for modelling and visualising the behaviour of event-based systems. Each subsystem is represented using a separate acyclic net, linked to others through a set of buffer places depicting their interactions.

However, CSA-nets suffer from challenges especially in analysing and visualising CESs that have a large number of subsystems resulting from alternative and concurrent execution scenarios. Moreover, CSA-nets currently lack the capability to represent multiple or coloured tokens, thereby limiting their ability to represent several similar processes simultaneously. This thesis introduces extensions for CSA-nets to capture compactly the relationships between interacting systems' components represented by sets of acyclic nets. Specifically, it introduces a way of folding buffer places to address the issue of a large number of buffer places. Then it introduces a new class of CSA-nets, called Parameterised Communication Structured Acyclic Nets (PCSA-nets), using multi-coloured tokens and allowing places to accept multiple tokens distinguished by parameters. The thesis also aims at improving the visualisation of csa-nets by rearranging their component acyclic nets to minimise the number of crossing arcs by taking inspiration from the main ideas behind three well-known sorting algorithms (bubble sort, insertion sort, and selection sort). Furthermore, this thesis presents a novel approach that combines TCP protocol anomaly detection with visual analysis through CSA-nets. The strategy provides a clear visualisation of cyber attack behaviours, leading a deeper understanding of Distributed Denial of Service (DDoS) patterns and their underlying causes. A new concept of Timed-Coloured Communication Structured Acyclic Nets (TCCSA-nets)

is introduced, which allows elaboration of the system's performance and emphasising the system's operations in real-time. This approach allows for the classification of messages as abnormal if their duration exceeds a predetermined time limit.

Acknowledgements

I want to express my deepest gratitude to a number of extraordinary people who were essential in the creation of this thesis. Foremost, heartfelt thanks to my parents. Their consistent love and guidance have been the pillars of my strength. They always had faith in me and provided immeasurable support throughout my journey. I am deeply appreciative of their constant encouragement and the core values they instilled in me. To my love, my wife, Abeer, thank you for your patience, understanding, and unwavering presence throughout the highs and lows of this journey. Your love and encouragement have been a source of great comfort and inspiration, pushing me forward during challenging times. Big thanks to my lovely daughter, Rifa, who fills my life with love and makes it more valuable.

I owe a great deal to my supervisor for his invaluable guidance, expertise, and constructive feedback during my research. His knowledge and attention to detail have significantly enriched my work and my skills. I am grateful for your continuous support and for motivating me to reach my potential. Also, big thanks to my second supervisor, Dr. Marta Pietkiewicz-Koutny, for her support, especially during the conferences in Bergen and Lisbon.

I am thankful to my colleagues for their support, engaging discussions, and the memorable moments we shared during this academic endeavor. Special thanks to my SON group, Professor Brian Randell and Dr. Anirban Bhattacharyya, for their feedback, discussions, and suggestions during my research. Their shared experiences have made my journey more enjoyable and rewarding. Also, special and heartfelt thanks to my close colleagues, Tuwaillaa and Nadiyah, for their extensive support during my research and conferences. Moreover, I would like to express my sincere gratitude to Professor Alex Yakovlevand and Professor Franck Pommereau for their valuable insights, constructive feedback, and the engaging

discussion during my viva examination. Lastly, I want to acknowledge everyone who contributed, in any measure, to my academic journey and to this research. Your involvement has been crucial in helping me evolve and complete this thesis.

Table of contents

List of figures

Chapter 1

Introduction

1.1 Background

The rapid advancements in digital technologies have revolutionised the landscape of data generation and collection, leading to an unprecedented surge in the volume, velocity, and variety of data, often referred to as big data [66, 21]. This exponential growth of data has far outpaced the capabilities of traditional data processing and analysis techniques, necessitating the development of novel approaches to effectively handle and derive meaningful insights from this data deluge. *Modelling and visualisation* have emerged as crucial techniques in this regard, enabling the transformation of raw data into comprehensible and actionable knowledge. These techniques provide a structured framework to represent the complex relationships and patterns hidden within the data, facilitating the exploration, interpretation, and communication of data-driven insights. By leveraging various modelling paradigms, such as mathematical models, statistical models, and machine learning algorithms, researchers and practitioners can explore the intricate dynamics of complex systems and develop predictive and prescriptive models to support decision-making processes. Moreover, the integration of interactive visualisation techniques improves how we interpret and use these models. This allows users to explore and interact with the data directly. Consequently, they can uncover hidden patterns and trends that may not be apparent through traditional means. The combination of modelling and visualisation techniques has become indispensable. These

methods empower organizations to fully harness the potential of their data assets. As a result, they drive innovation across diverse domains, from business intelligence to scientific discovery. As the complexity and scale of data are growing continuously. Consequently, the development and refinement of advanced modelling and visualisation approaches will remain critical research areas. These efforts lay the foundation for more efficient, effective, and insightful data-driven decision-making in response to the evolving challenges posed by big data.

Recently, the advancement of complex evolving systems (CES) is posing more and more challenges to researchers and investigators. The hurdle is that CESs is that such systems have a large number of sub-components interacting with each other and are subject to an wide range of external factors [11]. Additionally, CES are usually characterised by complicated evolution features which means that the system is not fully designed at a single point in time, but it continuously evolves. In line with this evolving feature, it is hard to state that there are complete assumptions and correctly stated requirements for CESs. Therefore, there is a need to develop reliable usable software for this task. In practice, successful software keeps evolving without stopping, and it has been shown that such systems consume about 40-70% of the budget in the maintenance costs [37]. Accordingly, in the professional approach, evolution must be considered prior to the design. The importance of structure in aiding designers to cope with design complexity is well-accepted, especially in the software engineering domain with its procedures, classes, types, etc., and in the VLSI design domain with its higher order logics, graph-based models. The effective use of such structuring notations greatly reduces the complexity of designs, involved in their representation and manipulation. Thus, the main purpose of a system design is to define how a systems will work [69].

Cybercrime modelling and investigation is a prominent example of CES system [98]. In particular, with the rapid development of the Internet and the Internet of Things [99], cybercrimes are rapidly increasing, making them a significant challenge. These system consist of several subsystems that communicate with each other. This communication can be synchronous or asynchronous. One of the main hurdles in such systems is how to find out

how the system will behave. Cybercrime investigation systems use different techniques and tools such as formal model and attack graph to support with their investigations. Thus, to mitigate this complexity while trying to improve the system's performance, it is advisable to represent complex behaviours in a visualised form. This is expected to ease the analyses of the systems and depict the failure points. Visualising big complex systems using methods that ensure precision, consistency, and dependability makes it possible to provide a clearer view of the events and conditions occurring in the system. For example, in the context of network security, visualising data flow can help identify patterns of suspicious activity, such as unusual login attempts or data transfers. This allows linking the events to specific details, such as the location where the cybercrime occurred. In other words, the investigator has a clearer representation of the situation, which helps simplify the understanding of the crime.

There are not many tools that can support visualisation of crime investigations. However, some assistance can be provided by Petri nets [50], process algebras [26], and Unified Modeling Language (UML) [83]. The main limitation of these approaches is that they make it difficult to model the evolution of a system in a practical and detailed manner. They often struggle to capture all the relevant details and dynamic changes over time, missing critical interactions and behaviors. As a result, these models may not accurately reflect the true dynamics of the system, leading to incomplete or incorrect analyses. This issue is especially pronounced with large systems such as cybercrime investigations, where there are more interactions between components, or representing the system as one model makes it difficult to understand and trace. In contrast, offering an approach that allows representing the large system as subsystems while showing the interactions between these subsystems makes the modeling of such systems easier to analyse and investigate.

A promising notation in this area, in terms of formal semantics and user-friendly tools, are acyclic nets and Communication Structured Acyclic Nets (CSA-nets), which are an extension of acyclic nets introduced to characterise the behaviour of complex evolving systems. Figure 1.1 shows an acyclic net with three conditions (represented by circles) and two events (represented by squares). The idea of CSA-net is to combine multiple acyclic nets by means of relations (implemented using so-called buffer places) with the aim of modelling

the dependencies between the components. Moreover, CSA-nets can represent the system's evolving behaviour through the use of 'behavioural abstraction'. This provides CSA-net with the ability to portray how the system evolves. It should be noted that the idea behind CSA-nets is to improve the cognitive understanding of the modelled CESs by using acyclic nets for individual subsystems.

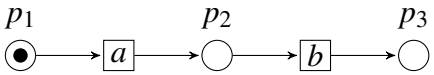

Fig. 1.1 Acyclic net.

In this thesis, we use CSA-nets as a means to handle various types of data within CESs. We develop algorithms and tool extensions for visualising and analysing data represented in extended CSA-nets. Also, we carry out evaluations and tests of these algorithms and tools to establish their effectiveness. Our approach was twofold, namely theoretical research and practical implementation. In the theoretical aspect, we propose a new class of models extending CSA-nets, focusing particularly on abstraction, and developing suitable formalisation for this purpose. For the practical implementation, we improve the visual graph representation of CSA-nets by reducing the number of crossings generated from communications between different subsystems. Moreover, in terms of application, we employ CSA-nets to analyse and detect SYN-flood attacks and create and implement supporting algorithms.

1.2 Aims and objectives

The aim of this PhD project was to develop and extend the CSA-nets framework to enhance its capability in modelling large systems, and to visualise such systems effectively. In essence, this would allow CSA-nets to effectively and meaningfully visualise behaviours of CESs which are common in practice yet still lack robust scientific and engineering support. This proposed research sought to address this imbalance by developing a methodology and tools centered on abstraction-based representations of system behaviours. Thus, the primary research question was:

Can CSA*-nets effectively model large and complex evolving systems?*

This has led to a subsequent research question:

Can CSA*-nets handle a substantial number of data entries in specific datasets, such as cybercrime data?*

The first question concerns the structure and components of CSA-nets and whether they can adequately represent large complex systems. The next question focuses on the capacity of CSA-nets to handle data within such representations. To address both questions, new methodologies for CSA-nets were designed and implemented, and the following key objectives have been identified.

Objectives

1. **Determining the current limitations in CSA-nets**

   CSA-nets depend on the concept of conditions and events. Moreover, buffer places are used to depict the relations between different system components (acyclic nets). However, not all the involved components are of major importance to a specific task or scenario in the modelling. Intuitively, the complexity of the system gets worse as the number of components increases. Thus, the presence of a high number of components may degrade the performance and disadvantage investigators.

2. **Surveying the existing approaches for mitigating model complexity in domains such as Petri nets**

   Extensive literature exists on the topic of model complexity and approaches for mitigating it in various areas of computer science. The goal here was to understand the techniques used to manage model complexity and their effects on the model.

3. **Developing the concept of model parametrisation for CSA-nets**

   Extending the concept of CSA-nets to effectively manage various data types includes, specifically, addressing the challenge of numerous components in CSA-nets. The

goal here was to develop a folding mechanism for buffer places resulting in master buffer place. After that, the goal was to introduce parametrisation for CSA-nets using multi-coloured tokens.

4. **Improving the visualisation of CSA-nets**

   CSA-nets consist of large numbers of acyclic nets, especially when modelling CESs. In particular, the arcs representing relationsips induced by buffer places often intersect with each other, making the model particularly hard to understand. This could make CSA-nets an unattractive choice. The goal here was to improve the representation of CSA-nets by minimising arc crossings generated from communication between different subsystems.

5. **Exploring how CSA-net can be applied in the field of cybersecurity**

   Focusing on Distributed Denial of Service (DDoS) attacks as a case study, which are widely occurring on the Internet due to their ability to exploit multiple protocols. In particular, the SYN-flood attack, targeting the TCP three-way handshake process by overwhelming a system with a large number of SYN messages, thereby consuming its computational and communication resources. To offer deeper insights into the intricacies of the TCP SYN-flood attack, the goal was to propose a visual simulation approach with the aim of benefiting from features provided by CSA-nets.

1.3 Contributions

Our main contributions developed to meet the aims and objectives of the undertaken research are as follows:

1. **A survey of the existing approaches of mitigating model complexity in Petri nets**
   Numerous attempts have been made in the area of Petri nets with a specific focus on complexity management through methods like folding and parametrisation. To fulfill *Objective 2*, we conducted a review of the literature on these attempts, concentrating on techniques proposed for managing complexity. Our aim was to identify effective

approaches and methods that tackle the complexity of modelling, both in terms of structure and behaviour.

2. **Parametrisation of CSA-nets aimed at dealing with model complexity**

   CSA-nets consist of sets of acyclic nets that communicate with each other through buffer places. However, this can generate an excessive number of buffer places, making the model hard to visualise and analyse. Additionally, having several components that perform similar task can result in the development of a complicated model. We introduce two extensions for CSA-nets, which can be used to efficiently depict the relationships between interacting systems' components represented by sets of acyclic nets. Specifically, we introduced a way of folding buffer places, and introduced parametrisations for CSA-nets. The combination of these techniques should lead to improved visualisation and analysis of large and complex CSA-nets. These outcomes align with *Objective 3*.

3. **A practical improvement for the visualisation of CSA-nets**

   To meet *Objective 4*, we improved the portrayal of the CSA-net by reducing the number of crossing arcs. The effectiveness of visual representations is paramount in dealing with complex systems, where nodes and edges are crucial in conveying relationships and structures. Accordingly, we took the inspiration from the core concepts of three well-known sorting algorithms—bubble sort, insertion sort, and selection sort—and integrated them with formulas to calculate crossings. We conducted experimental comparisons to evaluate the improvements resulting from each approach. The results indicate the high effectiveness of the selection sort-inspired approach.

4. **Detecting SYN-Flood Attack Using CSA-nets**

   With respect to *Objective 5*, we introduced the concept of TCCSA-nets, which facilitates a detailed analysis of the system's performance and its real-time operations. Specifically, we modelled clients and servers as acyclic nets, capturing their communication through the three-way handshake. A new algorithm was introduced to monitor

communication between these nets. Essentially, this algorithm tracks packets that successfully complete the communication sequence, identifying any abnormal packets that fail in the process. This innovative approach enables the classification of messages that exceed a predefined processing time threshold as abnormal, while treating other messages as normal communication.

1.4 Thesis outline

The thesis is organised as follows:

Chapter 1 provides an introduction to the thesis, outlining its objectives and contributions. Additionally, it includes a list of publications that are part of this thesis.

Chapter 2 provides an overview of the background details and related work relevant to Structured Occurrence Nets, visualisation, and cybersecurity.

Chapter 3 presents acyclic nets and communication structured acyclic nets (CSA-nets), including related notions and properties.

Chapter 4 presents a new class of extended CSA-nets, called *parametrised* CSA-*nets*. Also, it providides notions and properties related to their structure and semantics.

Chapter 5 provides new approaches to improve representation of CSA-nets, using the ideas from well-known sorting algorithms to minimize their crossings.

Chapter 6 introduces the concept of TCSA-nets to detect SYN-flood attacks.

Chapter 7 summarises and concludes the work, and also proposes directions for further research.

1.5 List of Publications

Parts of this thesis have been documented in the following publications:

1. Alahmadi, M., 2021. *Master Channel Places for Communication Structured Acyclic Nets*. In PNSE@ Petri Nets (pp. 233-240).

2. Alahmadi, M.: *Parametrisation of* CSA-*nets*. PNSE@Petri Nets (2022)

3. Alahmadi, M.: *Parameterised* CSA-*nets*. PNSE@Petri Nets (2023)

4. Alahmadi, M. and Koutny, M.: *Improving placement of* CSA-*nets*. International Conference on Smart Computing and Application (ICSCA) (2023)

5. Alahmadi, M.: *Detecting SYN Flood Attack using* CSA-*nets*. CS & IT Conference Proceedings (2023)

6. Alahmadi, M., Alharbi, S., Alharbi, T., Almutairi, N., Alshammari, T., Bhattacharyya, A., Koutny, M., Li, B. and Randell, B.: *Structured Acyclic Nets*. arXiv preprint arXiv:2401.07308 (2024)

1.6 Basic formal notations used throughout the thesis

All sets used in the structures considered in this thesis are finite. The disjoint union of sets X and Y is denoted by $X \uplus Y$, and nonempty sets $X_1, \ldots, X_k$ form a partition of a set X if $X = X_1 \uplus \cdots \uplus X_k$. The set of all subset of a set X is denoted by $\mathbb{P}(X)$.

For a binary relation R, xRy means that $(x,y) \in R$. The *composition* of two binary relations, R and Q, is a binary relation given by $R \circ Q = \{(x,y) \mid \exists z : xRz \wedge zQy\}$. Moreover, for every $k \geq 1$, w define:

$$R^k = \begin{cases} R & \text{if } k = 1 \\ R \circ R^{k-1} & \text{otherwise .} \end{cases}$$

Let X be a set and $R \subseteq X \times X$. Then

1. $id_X = \{(x,x) \mid x \in X\}$ is the *identity* relation on X.

2. R is *reflexive* if $id_X \subseteq R$.

3. R is *irreflexive* if $R \cap id_X = \varnothing$.

4. R is *transitive* if $R \circ R \subseteq R$.

5. (X,R) is a *partial order* if R is irreflexive and transitive.

6. $R^+ = R^1 \cup R^2 \cup \ldots$ is the *transitive closure* of R.

7. R is *acyclic* if $R^+ \cap id_X = \varnothing$. ◇

1.7 List of Acronyms

This table provides a list of all acronyms and their full meanings used throughout the thesis.

Acronym	**Full Meaning**
CES	Complex Evolving Systems
AN	Acyclic Net
ON	Occurrence Net
SON	Structured Occurrence Net
CSO-nets	Communication Structured Occurrence Nets
BSO-nets	Behavioural Structured Occurrence Nets
PA-nets	Parameterised Acyclic Nets
PCSA-nets	Parameterised Communication Structured Acyclic Nets
TCCSA-nets	Timed-Coloured Communication Structured Acyclic Nets
TCP	Transmission Control Protocol
DoS	Denial of Service
DDoS	Distributed Denial of Service
SYN	Synchronous
ACK	Acknowledgment

Table 1.1 List of Acronyms and their full meanings

Chapter 2

Structured Occurrence Nets, Visualisation, and Cybersecurity

2.1 Introduction

This chapter provides the background for this thesis, offering an overview of Structured Occurrence Nets (SONs), visualisation, and cybersecurity. Specifically, it presents extensions of SONs, which will be explored further in Chapter 3 and Chapter 4. Moreover, the chapter presents a discussion and review of current literature on visualisation and crossing arcs, laying the groundwork for Chapter 5. It also provides an overview of cybersecurity and cyber attacks setting the stage for a detailed analysis of the TCP protocol in Chapter 6.

2.2 Structured occurrence nets (SONs)

Structured Occurrence Nets (SONs) are a framework of formal models that can be used to analyse and visualise complex evolving systems. This section will provide an overview of the literature surrounding SONs. A broad insight into SONs forms the foundation of this thesis, where we will address their structure, extensions, advantages and limitations.

2.2.1 Occurrence nets

Occurrence nets are directed acyclic graphs that represent causality and concurrency, encapsulating all information pertaining to a single system execution [106]. The fundamental relationships between events in occurrence nets involve causality and concurrency. Causally dependent events are ordered, while concurrent events remain unordered. Furthermore, it is possible to generate an occurrence net as a direct representation of a system's execution history [104]. During each system execution, the process is able to unambiguously delineate the causal and concurrent connections between the occurring events. This extends not only to computer components, but also to human and physical process components, such as those involved in criminal or accident investigations.

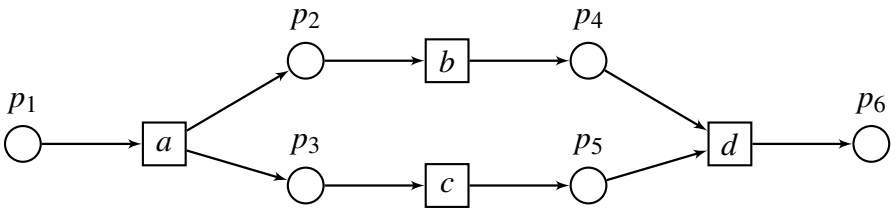

Fig. 2.1 Occurrence net.

Graphically, Figure 2.1 depicts an occurrence net composed of three main components: (i) places, represented by circles; (ii) events (or transitions), represented by rectangles; and (iii) flow relation, represented by directed arcs. The initial places in an occurrence net do not have an input event, whereas the final places have no output event. Moreover, each place can only have a single input event and only one output event [104]. Furthermore, every event within an occurrence net should have a minimum of one input and one output place. A global state, also referred to as a marking, is a set of places, and it is represented by black tokens placed inside the corresponding circles. The presence of a black token within a circle signifies its 'activation' within a potential execution history of the system modelled by the net [69].

In their original form, SONs are an extension of occurrence nets used to represent the execution behaviour of complex systems [105, 104, 106]. They consist of multiple occurrence nets that are connected through different types of formal relationships designed to capture information pertaining to either the actual/expected interaction behaviours or the collected evidence to be analysed. The strength of SONs lies in their structure, where complex repre-

sentations can be simplified when compared to analogous representations while providing a direct means of modelling emerging structures [69].

2.2.2 Communication structured occurrence nets (CSO-nets)

Communication Structured Occurrence Nets (CSO-nets) comprise multiple occurrence nets that are linked through unique elements referred to as 'buffer places'. Buffer places are capable of modelling both asynchronous and synchronous communication [69]. Events serve as the means to link these occurrence nets with each other through these buffer places. Figure 2.2 depicts a simple CSO-net, which comprises two occurrence nets: $ocnet_1$ and $ocnet_2$. Asynchronous communication between the two occurrence nets is represented by arrows, as seen between the events c and a in different occurrence nets linked through a single buffer place q_1. Thus, event a will never be executed before c in any execution sequence. A synchronous communication is represented by arrows pointing in both directions, using buffer places q_2 and q_3, as observed between events b and d. Thus, these two events have to be executed simultaneously.

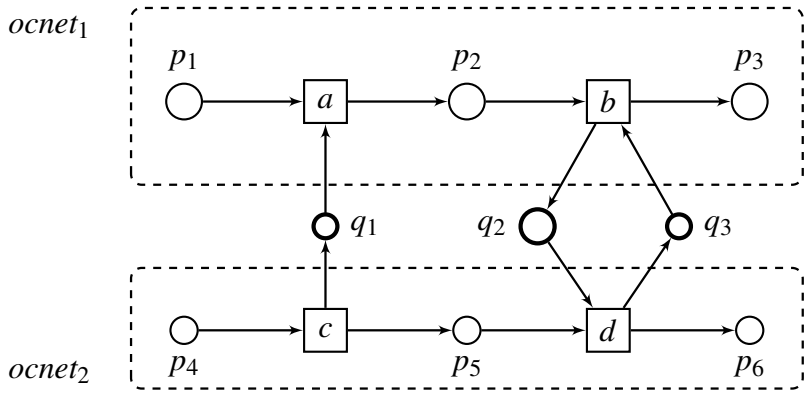

Fig. 2.2 Communication structured occurrence net (CSO-net).

2.2.3 Behavioural structured occurrence nets (BSO-nets)

Behavioural Structured Occurrence Nets (BSO-nets) are used for modelling the activities of evolving systems. Each BSO-net contains a two-level hierarchy of CSO-nets which is useful

for tracking the execution history [106]. The lower level represents behavioural details for various stages of evolution contained within the upper level. BSO-nets offer insights into the stages of a system's evolution on one level, while each phase of activities represents the stages of the system's evolution on the other level [69]. The connections across both levels help to highlight the relationships between the two kinds of behaviors.

The intended interpretation of Figure 2.3 is that the upper level offers a high-level perspective of a system that has undergone two successive versions, depicted by two states ($p1$ and $p2$) of the upper occurrence net, with the event in the middle representing a version update (transition a). The lower occurrence net illustrates the system's behavior during the same period. Moreover, the 'behavior' relation functions across the two levels of description, linking states in the lower part with those in the upper part that abstract them.

That is, any state can be viewed either as a state of a system or as representing a system with its own states and events. For example, p1 can be seen either as a state of the system or as a system with its own states and events (including $r1$, b, and $r3$). Generally, it is possible to have sets of related occurrence nets, some showing the evolution of systems and others depicting their behaviors. Thus, the former can be seen as the behavioral abstraction of the latter. A formal definition of this two-level occurrence net is omitted in this thesis; for more details, see ([106, 69]).

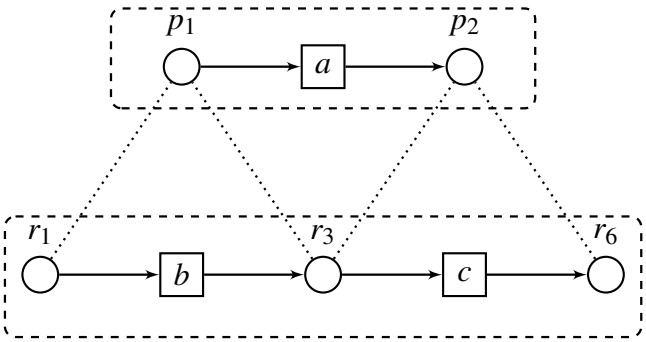

Fig. 2.3 Behavioural structured occurrence net (BSO-net).

2.2.4 Acyclic nets and communication structured acyclic nets (CSA-nets)

The original models of the SON framework were based on occurrence nets. More recently, and in this thesis, this base model has been generalised to acyclic nets [8]. In general, an acyclic net represents multiple rather than a unique execution history of a system, and can be seen as a collection of overlapping occurrence nets. The main part of this thesis will be concerned with the model of CSA-net which is a direct generalisation of CSO-nets where the role of the component occurrence nets is played by acyclic nets.

2.3 Other graphical modelling techniques

In addition to the models described above, there exist other formal, systematic, and mathematical frameworks that are able to represent, analyse, and verify intricate system behaviours. Commonly used models include (general) Petri nets, coloured Petri nets, attack graphs, and finite state machines. These models contribute to improved reliability, validity and predictability throughout the design process of computational modelling, primarily due to their ability to offer visual representations and rigorous methodologies for modelling, validating, and optimising processes. Such models are invaluable in various domains, including software design and cybersecurity. This section will explore several modelling techniques cited in the literature, identifying their importance, uses, and advantages.

2.3.1 Petri nets

Petri nets are a mathematical and graphical tool suitable for a wide range of systems, first proposed by Carl Adam Petri in 1962 to simulate concurrent behaviour within systems. Petri nets are particularly useful when analysing information processing systems characterised by concurrency, asynchrony, distribution, parallelism, and nondeterminism. Visually, Petri nets are often likened to flow charts, block diagrams, and networks, and the tokens in Petri nets facilitate the visualisation of the system's dynamic and simultaneous actions [89]. Petri nets are

also capable of facilitating communication between practitioners and theoreticians, empowering practitioners to learn more systematic modelling techniques from theoreticians while enabling theoreticians to construct more realistic models based on practitioners' ideas [89].

2.3.2 Coloured Petri nets (CPNs)

Coloured Petri nets (CPNs) are a graph-oriented language for concurrent system design, simulation, and verification with their base provided by Petri nets [59]. CPNs offer a discrete-event modelling language that combines the capabilities of Petri nets with a high-level programming language. The CPN programming language, based on the standard ML programming language, provides a natural means of expressing information types for managing information and creating parameterisable models. The primary difference between Petri nets and CPNs is that in CPNs a place can contain multiple (coloured) tokens with associated data attributes at any given time. Consequently, CPNs are suitable for applications where multiple data tokens are possible, such as distributed systems and communication protocols. As in Petri nets, there are four major components that make up a CPN: (i) places; (ii) transitions and (iii) arcs, represented by circles, rectangles or boxes, and arrows, respectively. These three components are additionally annotated and describe how to manipulate the coloured tokens (with colours which can be attributed to specific data types).

2.3.3 Attack graphs

The primary function of attack graphs is to identify potential areas of vulnerability in a system that can provide routes of attack for cybercriminals [112]. Attack graphs are commonly used in areas such as computer networks and systems, offering insights into vulnerabilities and corresponding countermeasures. In essence, attack graphs provide visual approach to identifying the routes that an attacker may use to compromise a network. Specifically, it represents every node that has vulnerabilities linked to it. The node with the greatest likelihood of being attacked can be termed a vulnerable node. Simple network scans are inadequate for assessing the security of modern complex, multiple platforms

networks with software applications. Therefore, it is crucial to ensure regular assessment of cybersecurity defences [55]. Using attack graphs, cyber-attack graphs offer a visual representation and deeper understanding of the connections among a network's vulnerabilities, potential strategies of an attacker, and the comprehensive risk to an organization's network. Thus, attack graphs allow organisations to identify vulnerabilities and provide important insight into how their networks can be exploited, making attack graphs essential for protecting critical digital assets [114]. Figure 2.4 shows an example of an attack graph representation of the modeled access token manipulation [130].

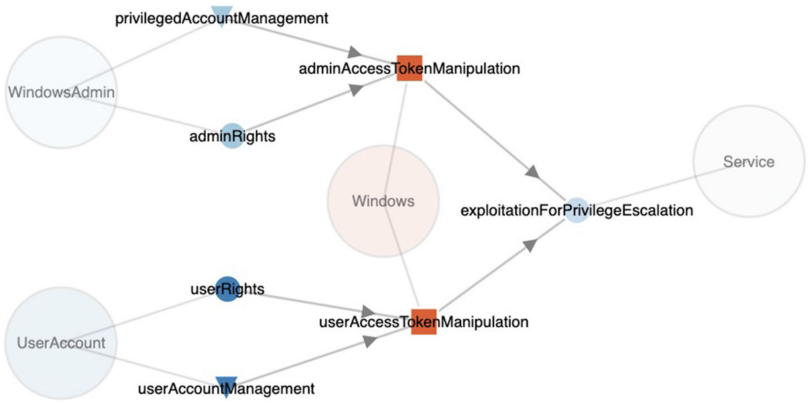

Fig. 2.4 Attack graph [130]

2.3.4 Finite state machines

Finite State Machines (FSMs) are a conceptual model commonly used in computational theory [14]. In essence, FSMs depict how a system changes between different states, capturing the system's behaviour and outcomes in a sequential flow of events. In any particular instance, an FSM is in one of the states coming from a predefined finite set of states. The progression from one state to another is termed as a transition and is in response to the specified inputs. The behaviour of an FSM is predetermined by its initial state [113]. There are two basic kinds of finite-state machines, deterministic FSMs and nondeterministic FSMs, and for every non-deterministic FSM, an equivalent deterministic FSM can be designed [2]. Various contemporary applications use FSMs, executing predefined sequences of operations controlled by series of external events. Although FSMs have been used in computational modelling for a

long period, the FSM based approaches are limited by their memory constraints, determined by their state size [77]. Moreover, FSM can not represent concurrency. Thus, FSM have been surpassed by other computation models, primarily due to computational power of modern modelling techniques.

2.4 Visualisation

This section focuses on visualisation and its significance for data representation. Additionally, it examines the issue of crossing arcs, a common challenge in visualisation, and explores sorting algorithms as a potential solution in the domain of SONs.

Visualisation has been widely utilised to provide researchers and decision-makers with detailed insights into data. It involves representing data graphically in a way that is both easy to understand and articulate. Consequently, visualisation is particularly important for analysing complex system and large data sets in a clear, precise, and efficient manner. Moreover, visualisation facilitates the analysis of data from multiple sources and diverse perspectives, enabling insights at various levels [79]. Graphical representation often communicates ideas more clearly than the text alone, making the primary benefit of visualisation its capacity to reveal hidden patterns more readily than other formats, such as raw data tables [87]. Another advantage of visualisation is its rapid adaptability to model changes. It is also characterised by an high optimisation capability, which has led to a wide range of applications in the field of computer science. As problems increase in complexity, visualisation tends to have a greater impact. In other words, it is beneficial for large data sets, where it can assist in identifying correlations and detecting anomalies [3].

2.4.1 Crossing arcs

Visualisations have a unique ability to present complex data from multiple sources while identifying intricate relationships, making data easily accessible. It is important to ensure that predefined criteria are in place to ensure data visualisation exhibits both efficiency and clarity, facilitating straightforward interpretation. Although graph visualisation is playing

an essential role in showing its ability to present complex data, one of key challenges that must be addressed is the presence of 'crossing arcs'. The term 'crossing arcs' refers to the intersections that occur when edges or arcs in a graph cross over one another [93].

The crossing of the arcs in diagrams can have a significant impact on the clarity and interpretability of visualisation. Specifically, it makes difficult for users to identify the links between nodes and recognise connections, especially of large and complex models. However, despite the importance of edges/arrows in the structure and representation of graphics in linking components, crossing arcs is consider one of the most common barriers in visualisations [75]. Thus, identifying the presence of crossing arcs, followed by attempts to minimise the occurrence of crossing arcs in visual representations, is crucial. This is important for facilitating and enhancing the understanding of data while preventing erroneous interpretations [129, 75]. However, visualisation of data and minimising the occurrence of crossing arcs is not a new concept and has been an area of intense research for several years. One well-established attempt to minimise crossing arcs is to colour the crossing edges with a different colour. This approach makes the edges stand out from each other and aims to reduce the negative effect of crossing edges. [61]. Despite the intentions of this approach, the use of different colours may not be beneficial, especially with large and complex systems that contain a significant number of interactions between their components.

In addition to using contrasting colours to mitigate the impact of crossing arcs, another approach is to convert visualised graphs into a better format, such as planar graphs [41]. This can be achieved by applying several vertexes splitting criteria and removes the crossing edges. However, this approach can make it difficult to track the original non-split nodes and may limit the graphs readability. Therefore, it is important to use the minimum number of vertex splittings, but identifying the minimum number required is difficult. In a recent study [92], the authors addressed the difficulty of tracking original non-split nodes when using vertex splitting. Specifically, they proposed new approach by re-embedding the split nodes while keeping the non-split nodes in their original positions, thus maintaining stability. An earlier attempt to reduce crossing edges was presented in [82], which proposed a heuristic

strategy using the strategic oscillation approach. The approach proposed in centred around an iterative greed strategy for both the constructive and destructive processes.

Using CSA-nets as an example, for a CSA-net representing system behaviour, the component acyclic nets yield rather simple graphs that are typically very long and contain very few crossings. However, the primary problem with this approach is that arcs representing communication between subsystems (acyclic nets) can cross several acyclic nets, and the resulting crossing edges may severely diminish the comprehension of the overall graphical display. In an attempt to address this problem, the component acyclic nets can be reordered in a way that minimises the number of crossings. To work towards achieving this goal, one can use ideas coming from the domain of sorting algorithm. In this particular case, sorting algorithms can reorder the component acyclic nets aiming at reducing the crossing edges. This approach offers several benefits, including not changing the display of the individual component acyclic nets.

2.4.2 Reducing crossing using sorting algorithms

In the domain of graph visualisation, the clarity of the graph is critical for understanding the data presented. One of the major challenges in this area is mitigating the crossing of arcs which could create visual clutter and impede comprehension. This issue is especially prevalent in layered graphs where nodes are arranged across distinct levels. The sequence in which these nodes are positioned can significantly influence the number of arc crossings. Therefore, the quality of graph representation often hinges on the proper ordering of its elements. To address this, sorting algorithms have proven to be a valuable method for reducing arc crossings. They aid in establishing an optimal or near-optimal sequence of nodes within each layer, which diminishes the likelihood of arc crossings. However, the selection of an appropriate sorting algorithm is critical in optimising the clarity of a visualisation [120]. In this section, we will recall three well-known algorithms: Insertion Sort, Bubble Sort, and Selection Sort.

Insertion sort

Insertion sort [88] is a popular sorting algorithm. Assuming that there is an array of n values, it begins by creating another empty array, and then it picks up one value at a time from the unsorted array and inserts it in its correct position in the sorted one. The time complexity is $O(n^2)$ in the worst case which limits its applicability with big data [62].

Selection sort

Selection sort [54] is another popular sorting algorithm that works by dividing the array into two parts, the left part which is the sorted one and the right part which is the unsorted one. The algorithm begins by searching for the smallest item in the right part, then swaps it with the leftmost item in the left part. This exercise is repeated until all the items in the right part are moved to the left part. As in the insertion sort, the algorithm's time complexity is $O(n^2)$ in its worst case [46].

Bubble sort

Bubble sort works by iterating over the array of items and comparing each item with the following one and swapping them if the following item is smaller. This is repeated until there are no more swaps to perform. As for the other two sorting algorithms, in the worst-case it has time complexity of $O(n^2)$ [47].

2.5 Cybersecurity

Cybersecurity is the practice of protecting internet connections from various threats [109]. The field of cybersecurity encompasses the protection of all connected hardware, software, and data [116]. Fundamentally, it aims to thwart unauthorised access to these components by protecting them from any potential attack [84]. This protection can be implemented at both the global and individual levels. At its highest level, cybersecurity seeks to prevent attacks that could disable or disrupt a system's services such as malware, phishing, and Distributed Denial of Service (DDoS) attacks [128]. A significant challenge is that many cyber threats

manifest at the protocol level [40]. Robust cybersecurity should enable individuals and enterprises to maintain a strong defense against malicious attacks. This would ensure that customers can trust that their data will remain secure, unaltered, and accessible only through authorised means.

This section provides an overview of cyber security, including protocols, network attacks, and mitigation strategies. These topics, particularly the network protocols and defensive measures against network attacks, will be delved into more comprehensively in Chapter 6.

2.5.1 Networking and Protocols

Protocols are defined as sets of rules facilitating the exchange of data between devices [108]. They are essential in maintaining active and secure Internet-based activities and communications. These protocols play a crucial role in the digital realm through enabling communication among various Internet components. Cyberattacks often happen when these protocol rules are broken [36]. Based on their main functionality, protocols are generally categorised into three broad categories. The first is network communication protocols, primarily dedicated to transferring data across the Internet. This category also handles tasks like authentication and error detection with the Transmission Control Protocol (TCP) [117] and can be example in this category. The second category is network security protocols which focus on ensuring safe data transmission. The Secure Socket Layer (SSL) [35] is a protocol in this category which helps secure internet connections and enables safe communication between both ends. The third category is network management protocols, which establish various policies for monitoring network performance. This facilitates easy troubleshooting for a reliable, fault-tolerant network. The Internet Control Message Protocol (ICMP) [124] is well-known in this category for aiding in diagnosing network connectivity issues.

2.5.2 Network attacks and threats

Networking is integral to everyday communications but remains vulnerable to cyber-attacks. These attacks often aim for unauthorised access to networks, with the intention to either steal

or alter data. Generally, network attacks fall into two main categories. The first is *passive* attacks [101], where attackers may eavesdrop on or intercept sensitive information, lacking the capability to modify the data. The second category is *active* attacks [90], which involve direct interference with the data, potentially altering or damaging it. Various forms of such attacks include:

1. Malware [18]: This kind of attack has a wide range of forms. It is abbreviated after malicious software and can be any kind of virus, spyware, or trojan. The intention of these programs varies, but the majority aim to steal personal data or prevent access to specific resources.

2. Man-in-the-Middle Attack [122]: This kind of attack occurs when an attacker eavesdrops on a communication channel between two parties. By this, the attacker can hijack the channel to either alter the message sent or steal the data.

3. Denial-of-Service (DoS) Attack [94]: This kind of attack represents a major risk in which the attacker floods the network with a huge amount of messages with the aim of depleting their resources and bandwidth. This consumes the server's resources and slows it down or is completely overwhelmed and unable to respond. This attack has a more hazardous form when it is distributed DoS (DDoS). Such DDoS involves several attackers performing DoS on the same server at the same time.

2.5.3 TCP protocol

The Transmission Control Protocol (TCP) facilitates the exchange of data between devices and ensures reliable packet delivery. Often described as the backbone of the Internet due to its extensive range of applications, TCP is particularly prevalent in network communications where errors in data delivery are unacceptable. This protocol begins by establishing a connection between the sender and the receiver, maintaining this connection throughout the communication process, which classifies TCP as a connection-oriented protocol. TCP employs a method known as the three-way handshake to establish connections. Additionally, it divides larger data into smaller packets, ensuring data integrity throughout the process [27].

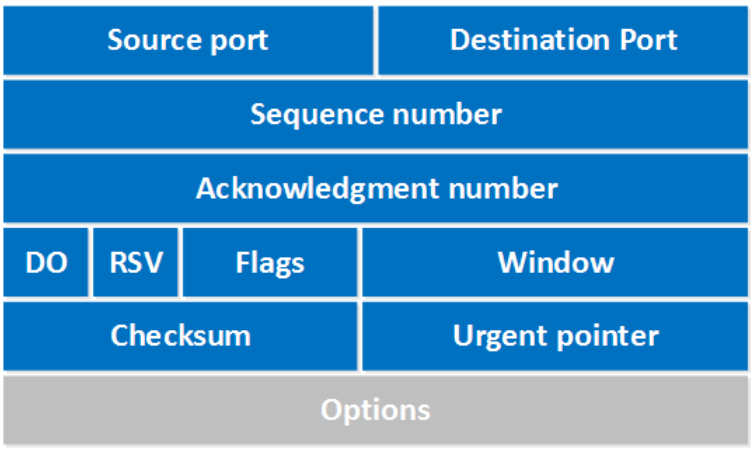

Fig. 2.5 TCP header.

TCP also employs error detection strategies to ensure that the data has been received correctly. This includes setting the connection timeout duration, activating the checksum field, and receiving and sending acknowledgments. Accordingly, any missing packets can be easily detected and recovered by resending them again.

TCP connection is a full duplex that is established using a well-known mechanism referred to as a three-way handshake. Generally, this mechanism allows both the sender and receiver to synchronize (SYN) and acknowledge (ACK) each other. As its name implies, it consists of three consecutive steps, as follows. First, the client sends a SYN message to the receiver (server), requesting to connect. Second, the receiver responds with both SYN and ACK. The SYN in this case means that the server is ready to connect and the ACK confirms receiving the sender's previous SYN message. Finally, the client sends ACK to the server, confirming receiving its previous message and the connection is now established. These steps are summarised in Figure 2.6.

The TCP has an informative header, as shown in 2.5, including 11 fields. The most important fields are the source and destination port numbers. Each of these is 16 bits and is used to represent the sender and receiver applications. Moreover, to achieve reliability, two fields; sequence and acknowledgment numbers are used. Both are 32 bits. The former represents the amount of data sent in one session, while the latter represents a request for the next TCP segment. For easy decapsulating the header a 4-bit field named DO is used to represent the header length. For the connection-oriented feature, 9-bits flag known as control bits are used.

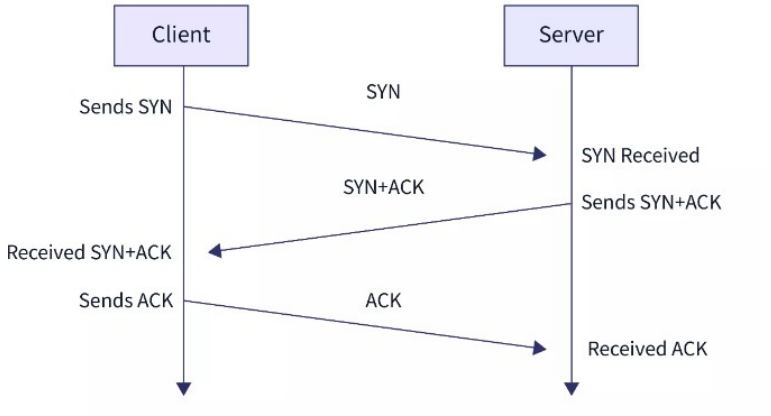

Fig. 2.6 Three-way handshake process

They are used to establish connections, send data and terminate connections. These include an urgent pointer, ACK flag, push (PSH), reset connection (RST), SYN, and FIN. Additionally, there are 16-bits referred to as window defining how many bytes the receiver is willing to receive. Another 16-bits is referred to as a checksum is used to check if the TCP header is OK or not. A final field is the option that can be valued between 0 and 320 bits.

In this thesis, our primary focus is on using the TCP protocol as a case study. The reason behind choosing such a protocol is twofold. First, it is widely used in a large number of applications. Second, it suffers from several types of cyberattacks, like the SYN-flood attack.

SYN-flood attack

Despite its high importance on the Internet, TCP is vulnerable to several cyberattacks, such as SYN-flood attack [133]. Specifically, the SYN-flood attack targets the three-way handshake mechanism during the establishment of a TCP connection. Technically, SYN-flood is a type of DDoS that exploits the three-way handshake to consume the server's resources. It involves sending repeated SYN request packets to the server ports using fake IPs. These requests appear to be legitimate, and the server is deceived and tries to respond to them (SYN-ACK), which, in turn, wastes its resources. This attack is summarised in Figure 2.7. The hurdle is that it is hard for the TCP protocol to detect such a situation. This is because TCP is called a half-open connection [91]. This occurs when the third step of the three-way handshake

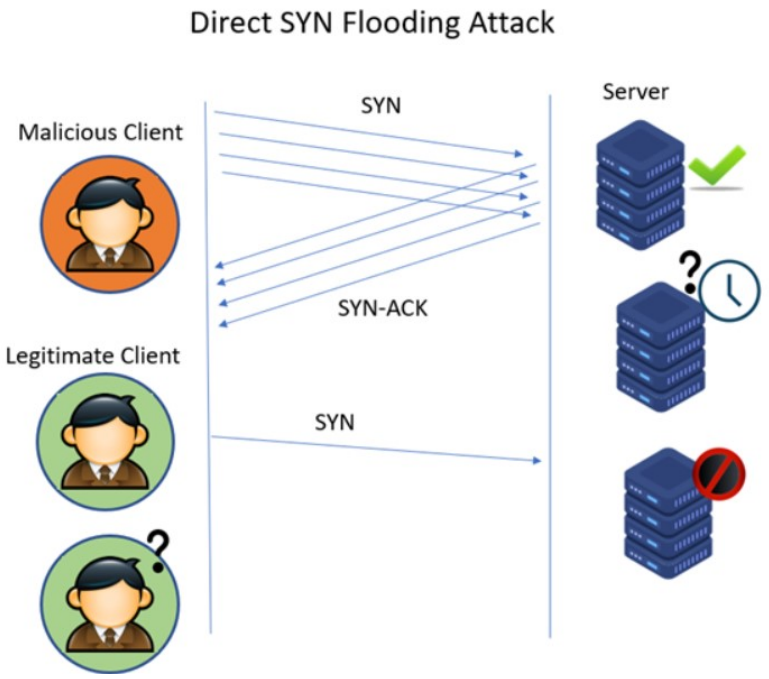

Fig. 2.7 A graphical representation for direct SYN-flood attack.

including sending final ACK to the server, fails or if the host closes the connection without acknowledging the other.

Attackers use various source IP addresses while flooding the target server with SYN packets; otherwise, their attacks will be readily prevented by firewalls. This is commonly known as IP address spoofing which includes injecting raw IP packets with authentic IP and TCP headers and also manipulating local firewall rules to bypass the firewall entirely. Additionally, IP address spoofing techniques can be classified into various categories depending on the type of spoofed source addresses used in the attack packets [4].

The SYN-flood attack is carried out in three ways:

1. Direct SYN Flood Attack; shown in Figure 2.7: This situation arises when an attacker floods the target with an overwhelming number of SYN messages originating from the same source IP address. To counteract this, the targeted device needs to be sufficiently sophisticated to refrain from replying to SYN-ACKs messages.

2. SYN Spoofed Attack; shown in Figure 2.8: Involves sending a massive amount of SYN messages from different IP addresses. This is named forged/spoof IP addresses and are

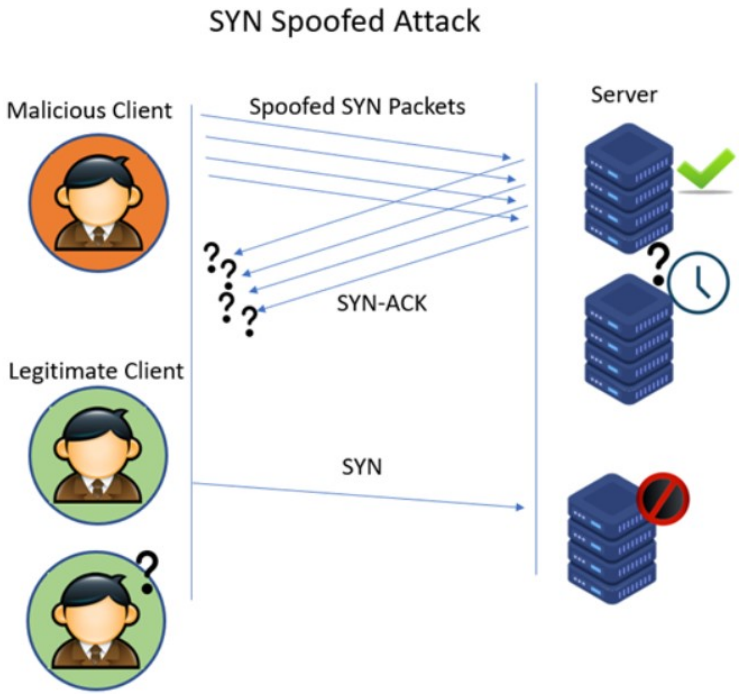

Fig. 2.8 A graphical representation for SYN Spoofed attack.

hard to discover. In this way, the server responds to a fake device (spoofed IP), so the packet is discarded over the Internet. The major task on the attacker side is to correctly choose the set of IP addresses that are not in use.

3. DDoS SYN attack; shown in Figure 2.9: In this variant, the victim server receives SYN packets at the same time from several infected computers under the control by the attacker. These infected computers are called botnets.

In the literature, researchers provided several techniques to detect SYN-floods:

1. SYN Cookies: This method involves encoding the connection state information into the initial sequence number (ISN) sent by the server in the SYN-ACK packet. This allows the server to avoid allocating resources for half-open connections until the client sends the final ACK packet.

2. Increasing SYN Queue Size: By increasing the size of the SYN queue on the server, the system can handle more simultaneous connection attempts, making it more resilient to

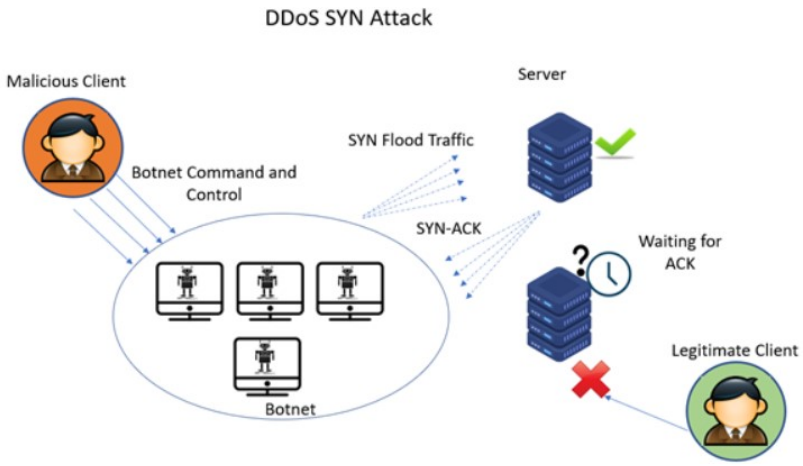

Fig. 2.9 DDoS SYN attack

SYN-flood attacks. However, this approach may not be sufficient against large-scale attacks.

3. Rate Limiting: This approach involves limiting the number of SYN packets accepted by the server within a specified period. While this can help mitigate SYN-flood attacks, it may also affect legitimate clients' ability to connect.

4. Eliminating the oldest half-open connection: By eliminating the predefined number of the oldest half-open connection from the SYN log. This frees up the queue and allows new connections to be established. However, this does not work well with DDoS.

5. Firewall Filtering: This involves configuring the firewall rule with the aim of filtering out, but not limited to, the DDoS SYN messages and unauthorised access.

2.6 Conclusion

This chapter introduced the background related for the thesis by introducing three areas. Initially, we delved into SON and explored their properties and extensions including acyclic nets and communication acyclic nets. However, while CSA-nets offer a robust framework for modeling various processes, it has some limitations. A notable drawback is its inability to efficiently model scenarios involving multiple tokens, which affects its capacity to abstract components in complex systems comprehensively. This restriction underscores a critical

need for further development in this area to enhance its modeling capabilities. This will be the topic of Chapter 4. The second focus area encompassed visualisation techniques aimed at minimising crossing arcs in the domain of CSA-nets which is the topic of Chapter 5. In particular, CSA-net faces significant challenges in representing large-scale systems. Specifically, the issue of crossing arcs remains particularly problematic, detracting from the clarity and effectiveness of the visual representation. This limitation points to an urgent need for innovative solutions that can address these challenges, thereby improving the interpretability and utility of CSA-net in representing complex systems. The third area of exploration provided a basic background about the topic of Chapter 6. Specifically, it provided how to model and visual TCP attacks. While CSA-net emerges as a promising framework for modeling and analysing complex systems such as cyber crime investigation, its application in the domain of cybersecurity, particularly in combating cybercrime, reveals gaps. The framework's current capabilities fall short of effectively addressing the nuances and complexities of cyber threats, indicating a pivotal area for future research and development. However, addressing these challenges is essential for advancing the field and enhancing the practical application of SON in diverse domains.

Chapter 3

Acyclic Nets and Communication Structured Acyclic Nets

3.1 Introduction

Acyclic Petri nets are a type of Petri nets that are directed graphs with no cycles. This implies that there are no circular path in which one can begin at a certain point (either a place or transition) and return to the same point by following a sequence of directed connections. The absence of such cycles simplifies analysis of the representations of behaviours and their attributes. In particular, they are useful where the sequence of events is strictly linear, such as in manufacturing processes and workflow systems. That is, they are beneficial for modelling situations in which a particular process or operation does not occur more than once, or in which processes do not return to their already visited point.

In recent years, there has been significant interest in acyclic nets, as they can be used to model system behaviours in various application areas; for example, cybercrime investigation [57, 132], AI, and clinical investigations. The model presented in this chapter is that of Communication Structured Acyclic Nets (CSA-nets), as used in [12, 13]. A CSA-net consists of multiple acyclic nets interacting through buffer places. CSA-nets allow both asynchronous and synchronous communication between acyclic nets, and so can represent the concurrent

behaviour of the system as well as synchronisation behaviour. They are suitable tools for modelling and visualising the behaviour of event-based systems.

This chapter adapts the work presented in [8], which includes the foundation for all notions and semantics in this thesis. Specifically, it introduces the notions and notations required to develop the current work and is organised as follows. In Section 3.2, we introduce the basic notions and examples related to acyclic nets. Section 3.3 presents the semantics of such a model, including step sequences, executed steps, and defines well-formed acyclic nets. In Section 3.5, we introduce the notion of Communication Structured Acyclic Nets (CSA-nets) and provide examples demonstrating how different executions of a concurrent scenario are generated. Furthermore, in Section 3.6, we offer examples to illustrate how various behavioural notions related to step sequences and reachable markings of CSA-net are captured.

3.2 Acyclic nets

When used, for example, as a representation of an incident, an acyclic (Petri) net is a component "database" of empirical facts (expressed using places, transitions, and arcs linking them) accumulated during an investigation. Transitions (events) and places (conditions/local states) are related through arrows representing causal and/or temporal dependencies (hence, the database is required to be acyclic). Moreover, acyclic nets can represent alternative ways of interpreting what has happened and so may exhibit (backwards and forwards) non-determinism.

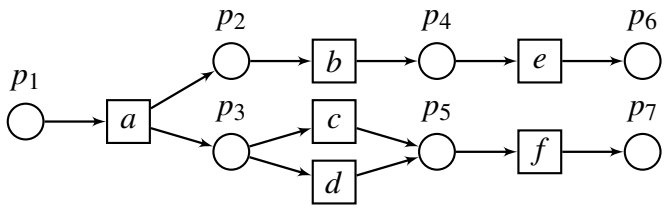

Fig. 3.1 Acyclic net $acnet_1$.

Definition 3.2.1 (acyclic net) *An* acyclic net *is a triple* $acnet = (P,T,F)$*, where* $P(= P_{acnet})$ *and* $T(= T_{acnet})$ *are disjoint finite sets of* places *and* transitions *respectively, and* $F(= F_{acnet})$ *is the* flow relation *included in* $(P \times T) \cup (T \times P)$ *such that:*

1. *P is nonempty.*
2. *F is acyclic.*
3. *For every $t \in T$, there are $p,q \in P$ such that pFt and tFq.*

The set of all acyclic nets is denoted by AN. ◇

Graphically, places are represented by circles, transitions by boxes, arcs between the nodes represent the flow relation, and markings are indicated by black tokens placed inside the circles.

In addition to the acyclicity of F, it is required that each event has at least one pre-condition (pre-place) and at least one post-condition (post-place). Acyclic net can exhibit backward non-determinism (when more than one arrow is incoming to a place) as well as forward non-determinism (when more than one arrow is outgoing from a place).

Notation 1 (direct precedence in acyclic net) *Let acnet be an acyclic net. To indicate relationships between different nodes, for all $x \in P_{acnet} \cup T_{acnet}$ and $X \subseteq P_{acnet} \cup T_{acnet}$, we denote the* directly preceding *and* directly following *nodes as follows:*

$$\begin{array}{rclcrcl} {}^\bullet x & = & \mathrm{pre}_{acnet}(x) & = & \{z \mid zF_{acnet}x\}, & {}^\bullet X & = & \mathrm{pre}_{acnet}(X) & = & \bigcup\{{}^\bullet z \mid z \in X\} \\ x^\bullet & = & \mathrm{post}_{acnet}(x) & = & \{z \mid xF_{acnet}z\}, & X^\bullet & = & \mathrm{post}_{acnet}(X) & = & \bigcup\{z^\bullet \mid z \in X\}\,. \end{array}$$

Moreover, the initial *and* final *places are respectively given by:*

$$P^{init}_{acnet} = \{p \in P \mid {}^\bullet p = \varnothing\} \;\; \textit{and} \;\; P^{fin}_{acnet} = \{p \in P \mid p^\bullet = \varnothing\}\,.$$

Note that having the notations like ${}^\bullet x$ in addition to (more explicit) $\mathrm{pre}_{acnet}(x)$ *helps to keep some of the subsequent formulas short.*

Proposition 3.2.1 ([8]) $P_{acnet} = P^{init}_{acnet} \uplus \text{post}_{acnet}(T_{acnet}) = P^{fin}_{acnet} \uplus \text{pre}_{acnet}(T_{acnet})$, *for every acyclic net acnet.*

Example 1. In Figure 3.1, $acnet_1$ is an acyclic net such that ${}^\bullet p_5 = \text{pre}_{acnet_1}(p_5) = \{c,d\}$ and $a^\bullet = \text{post}_{acnet_1}(a) = \{p_2, p_3\}$. Moreover, $P^{init}_{acnet_1} = \{p_1\}$ and $P^{fin}_{acnet_1} = \{p_6, p_7\}$. ◇

Definition 3.2.2 (occurrence net) *An* occurrence net *is an acyclic net such that* $|{}^\bullet p| \leq 1$ *and* $|p^\bullet| \leq 1$, *for every place p. The set of all occurrence nets is denoted by ON.* ◇

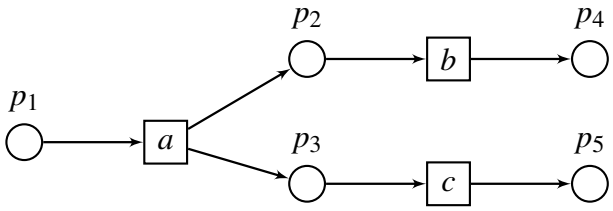

Fig. 3.2 Occurrence net.

An occurrence net is an acyclic net with a completely clear record of all causal dependencies between the events (transitions) it involves. In other words, it exhibits backward determinism (at most one arrow is coming to a place) and forward determinism (at most one arrow is going out from a place). According to the step sequence semantics defined later on, occurrence nets are deterministic in the sense that each transition has a fixed set of direct or indirect predecessors, which have to occur before the transition is executed. Once a transition is enabled, no other transition can disable it. We also consider acyclic nets where only backward non-determinism is disallowed.

Definition 3.2.3 (backward deterministic acyclic net) *A* backward deterministic acyclic net *is an acyclic net such that* $|{}^\bullet p| \leq 1$, *for every place p. The set of all backward deterministic acyclic nets is denoted by BDAN.* ◇

In the literature, backward deterministic acyclic nets are sometimes called *nondeterministic occurrence nets* or even *occurrence nets* (in which case occurrence nets as defined above are called *deterministic occurrence nets*).

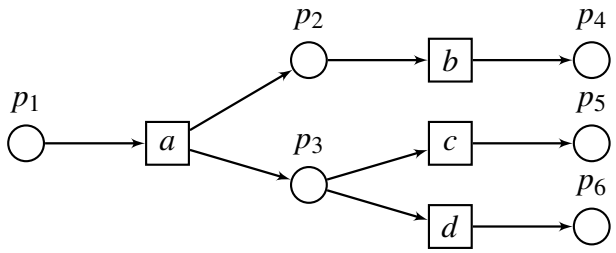

Fig. 3.3 Backward deterministic acyclic net $bdacnet_1$.

Example 2. Figure 3.3 shows a backward deterministic acyclic net. ◇

An acyclic net may exhibit both forward and backward nondeterminism and, as a result, represent several different possible execution histories. Next, we present scenarios and maximal scenarios as structurally defined execution histories of acyclic nets. The role of such notions is, in particular, to identify structurally all execution histories which can then be inspected, simulated, and analysed using graph based algorithms.

Scenarios of an acyclic net are acyclic subnets which start at the same initial marking and are both backward and forward deterministic. As a result, each scenario represents a distinct execution history with clearly determined causal relationships. Note that scenarios are more abstract than (mixed) step sequences defiined latyer on, as one scenario will in general correspond to many step sequences.

Below, for two acyclic nets, *acnet* and *acnet*$'$, we denote $acnet' \sqsubseteq acnet$ if $P_{acnet'} \subseteq P_{acnet}$, $T_{acnet'} \subseteq T_{acnet}$, $P^{init}_{acnet'} = P^{init}_{acnet}$, and

$$F_{acnet'} = F_{acnet}|_{(P_{acnet'} \times T_{acnet'}) \cup (T_{acnet'} \times P_{acnet'})}.$$

In other words, if *acnet*$'$ is a subnet of *acnet* with the same set of initial places.

Definition 3.2.4 (scenario and maximal scenario) *Let acnet be an acyclic net.*

1. *A* scenario *of acnet is an occurrence net ocnet such that ocnet* $\sqsubseteq$ *acnet.*

2. *A* maximal scenario *of acnet is a scenario ocnet such that there is no scenario ocnet*$'$ *satisfying ocnet* $\sqsubseteq$ *ocnet*$'$ *and ocnet*$' \neq$ *ocnet.*

The set of all scenarios of acnet is scenarios$(acnet)$*, and the set of all maximal scenarios is* maxscenarios$(acnet)$. ◇

Intuitively, scenarios represent possible deterministic executions (concurrent histories). Maximal scenarios are complete in the sense that they cannot be extended any further.

Note that an occurrence net has exactly one maximal scenario (itself).

Example 3. Figure 3.4 shows two maximal scenarios, $ocnet_1$ and $ocnet_2$, of the acyclic net in Figure 3.1. ◇

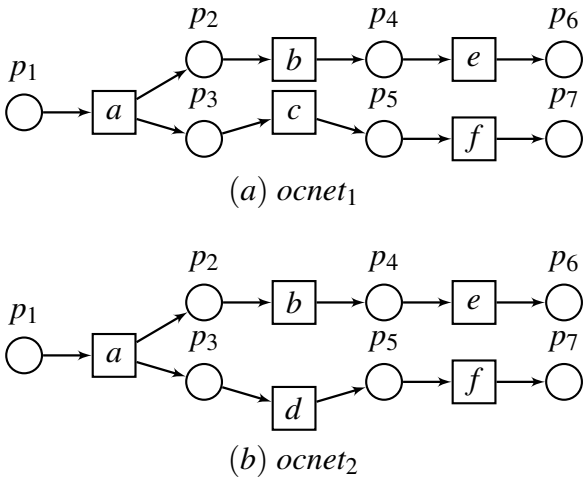

Fig. 3.4 Two maximal scenarios of the acyclic net in Figure 3.1.

3.2.1 Causality, concurrency, and conflict

This section introduces structural properties of acyclic nets.

Definition 3.2.5 (structural notions) *Let $acnet = (P,T,F)$ be an acyclic net.*

1. *Two transitions $t \neq u \in T$ are in* direct (forward) conflict*, denoted $t\#_0 u$, if they have a common place in their predecessors, i.e., ${}^\bullet t \cap {}^\bullet u \neq \varnothing$.*
2. *Two transitions $t \neq u \in T$ are in* direct backward conflict *if they have a common place in their successors, i.e., $t^\bullet \cap u^\bullet \neq \varnothing$.*

3. *Two nodes $x \neq y \in P \cup T$ are* concurrent*, denoted $x \,\mathrm{co}\, y$, if neither $x \# y$ nor $x F^+ y$ nor $y F^+ x$.*

4. $\mathrm{caused}_{acnet}(x) = \{y \in P \cup T \mid x F^+ y\}$ *are the elements* caused *by $x \in P \cup T$. Moreover,* $\mathrm{caused}_{acnet}(X) = \bigcup_{x \in X} \mathrm{caused}_{acnet}(x)$*, for $X \subseteq P \cup T$.*

$\diamond$

Intuitively, conflicts between transitions/events arise when they share a common pre-place (in forward non-determinism) or a common post-place (in backward non-determinism), while concurrency is a result of multiple post-places emerging from a single transition [71].

In an acyclic net, *forward non-determinism* can only occur when there are output transitions for some place p (i.e., $|p^\bullet| > 1$), and *backward non-determinism* can only occur when there are multiple input transitions for some place p (i.e., $|{}^\bullet p| > 1$).

3.3 Step sequence semantics of acyclic nets

In this section we introduce notions related to the behaviour of acyclic nets. A step is a set of transitions/events that could have happened at the same time and caused a move from one sysytem's state (marking) to another.

Definition 3.3.1 (step and marking of acyclic net) *Let acnet be an acyclic net.*

1. *The* markings *of acnet, denoted by* $\mathrm{markings}(acnet)$*, are the subsets of P:*

$$\mathrm{markings}(acnet) = \mathbb{P}(P_{acnet}).$$

2. *The* steps *of acnet, denoted by* $\mathrm{steps}(acnet)$*, are defined as:*

$$\mathrm{steps}(acnet) = \{U \in \mathbb{P}(T) \mid \forall t \neq u \in U : \mathrm{pre}_{acnet}(t) \cap \mathrm{pre}_{acnet}(u) = \varnothing\}.$$

Hence, in a step no two transitions share a pre-place.

3. *The default* initial *marking of acnet is* $M^{init}_{acnet} = \{p \in P \mid \mathrm{pre}_{acnet}(p) = \varnothing\}$. ◇

Graphically, markings are indicated by black tokens placed inside the corresponding circles.

Example 4. For the acyclic net $bdacnet_1$ depicted in Figure 3.3, we have $M^{init}_{bdacnet_1} = \{p_1\}$ and $\mathrm{steps}(bdacnet_1) = \{U \in \mathbb{P}(\{a,b,c,d\}) \mid c \notin U \vee d \notin U\}$. ◇

Definition 3.3.2 (enabled and executed step of acyclic net) *Let M be a marking of an acyclic net acnet. A step U of acnet is* enabled *at marking M if* $\mathrm{pre}_{acnet}(U) \subseteq M$. *It can then be* executed *and yield the marking*

$$M' = (M \cup \mathrm{post}_{acnet}(U)) \setminus \mathrm{pre}_{acnet}(U).$$

This is denoted by $M[U\rangle_{acnet} M'$. ◇

Enabling a step in a global state (marking) amounts to having all its pre-places marked. The execution of such a step adds tokens to all its post-places and then removes tokens from all its pre-places.

Note that markings of acyclic nets are safe by definition, i.e., a place can only 'hold' at most one token. That is, if $M[U\rangle_{acnet} M'$ and $a \neq b \in U$ are such that $p \in a^\bullet \cap b^\bullet$, then p will 'hold' only one token in M'.

To capture the behaviour of acyclic nets (and other nets later on), we use step sequences, involving reachable markings and executed steps.

Definition 3.3.3 ((mixed) step sequence of acyclic net) *Let $M_0, M_1 \ldots, M_k$ ($k \geq 0$) be markings and $U_1, \ldots, U_k$ be steps of an acyclic net acnet such that $M^{init}_{acnet} = M_0$ and we have $M_{i-1}[U_i\rangle_{acnet} M_i$, for every $1 \leq i \leq k$. Then*

1. $\mu = M_0 U_1 M_1 \ldots M_{k-1} U_k M_k$ *is a* mixed step sequence from M_0 to M_k.

2. $\sigma = U_1 \ldots U_k$ *is a* step sequence from M_0 to M_k.

The above two notions are denoted by $M_0[\mu\rangle\rangle_{acnet} M_k$ *and* $M_0[\sigma\rangle_{acnet} M_k$, *respectively. Moreover,* $M_0[\sigma\rangle_{acnet}$ *denotes that σ is a step sequence* enabled at M_0, *and* $M_0[\rangle_{acnet} M_k$ *denotes that M_k is* reachable from M_0. ◇

Note that if $k = 0$ then $\mu = M_0$ and the corresponding step sequence σ is the *empty* sequence denoted by λ.

The previous definition considers the start point of an execution as an arbitrary marking. The next definition will introduce various notions related to behaviour, and assumes that the starting point of system executions is the default initial marking.

Definition 3.3.4 (behaviour of acyclic net) *The following sets capture various behavioural notions related to step sequences and reachable markings of an acyclic net acnet.*

1. $\mathrm{sseq}(acnet) = \{\sigma \mid M^{init}_{acnet}[\sigma\rangle_{acnet} M\}$ step sequences.
2. $\mathrm{mixsseq}(acnet) = \{\mu \mid M^{init}_{acnet}[\mu\rangle\rangle_{acnet} M\}$ mixed step sequences.
3. $\mathrm{maxsseq}(acnet) = \{\sigma \in \mathrm{sseq}(acnet) \mid \neg\exists U : \sigma U \in \mathrm{sseq}(acnet)\}$
   maximal step sequences.
4. $\mathrm{maxmixsseq}(acnet) = \{\mu \in \mathrm{mixsseq}(acnet) \mid \neg\exists U, M : \mu U M \in \mathrm{mixsseq}(acnet)\}$
   maximal mixed step sequences.
5. $\mathrm{reachable}(acnet) = \{M \mid M^{init}_{acnet}[\rangle_{acnet} M\}$ reachable markings.
6. $\mathrm{finreachable}(acnet) = \{M \mid \exists \sigma \in \mathrm{maxsseq}(acnet) : M^{init}_{acnet}[\sigma\rangle_{acnet} M\}$
   final reachable markings.
7. $\mathrm{fseq}(acnet) = \{U_1 \ldots U_k \in \mathrm{sseq}(acnet) \mid k \geq 1 \implies |U_1| = \cdots = |U_k| = 1\}$
   firing sequences.

We can treat individual transitions as singleton steps; e.g., a step sequence $\{t\}\{u\}\{w,v\}\{z\}$ can be denoted by $tu\{w,v\}z$. ◇

Example 5. The following hold for the acyclic net $bdacnet_1$ in Figure 3.5.

1. $\mathrm{sseq}(bdacnet_1) = \{\lambda, a, ab, ac, ad, abc, acb, abd, adb, a\{b,c\}, a\{b,d\}\}$.
2. $\mathrm{mixsseq}(bdacnet_1) = \{\{p_1\}, \{p_1\}a\{p_2,p_3\}, \{p_1\}a\{p_2,p_3\}\{b,c\}\{p_4,p_5\}, \ldots\}$.

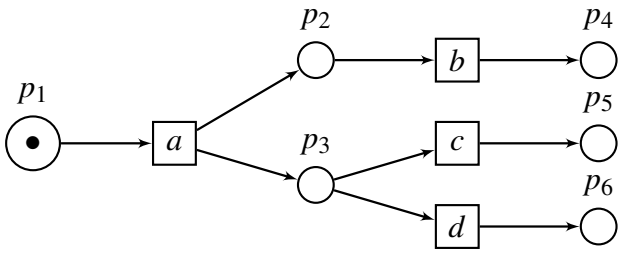

Fig. 3.5 Acyclic net $bdacnet_1$ of Figure 3.3 with the initial marking indicated.

3. $\text{maxsseq}(bdacnet_1) = \{abc, acb, a\{b,c\}, abd, adb, a\{b,d\}\}$.

4. $\text{maxmixsseq}(bdacnet_1) = \{\{p_1\}a\{p_2,p_3\}\{b,c\}\{p_4,p_5\},\dots\}$.

5. $\text{reachable}(bdacnet_1) = \{\{p_1\},\{p_2,p_3\},\{p_2,p_5\},\{p_2,p_6\},\{p_4,p_3\},\dots\}$.

6. $\text{finreachable}(bdacnet_1) = \{\{p_4,p_5\},\{p_4,p_6\}\}$.

7. $\text{fseq}(bdacnet_1) = \{\lambda, a, ab, ac, ad, abc, acb, abd, adb\}$. ◇

3.4 Well-formed acyclic nets

A basic requirement of criterion applied to acyclic nets is well-formedness. Its major purpose is to guarantee a clear depiction of causality in modelled behaviours. A well-formed definition of acyclic net is derived from the notion of a well-formed step sequence. Note that well-formedness is stronger than safeness. In particular, it prevents the same transition from executing more than once. So, a well-formed acyclic net represents executions in which each transition can only be executed once.

Definition 3.4.1 (well-formed step sequence of acyclic net) *A step sequence $U_1 \dots U_k$ of an acyclic net acnet is* well-formed *if the following hold for every $M_0U_1M_1 \dots M_{k-1}U_kM_k \in$ $\text{mixsseq}(acnet)$:*

- $\text{post}_{acnet}(t) \cap \text{post}_{acnet}(u) = \varnothing$, *for every* $1 \leq i \leq k$ *and all* $t \neq u \in U_i$.
- $\text{post}_{acnet}(U_i) \cap \text{post}_{acnet}(U_j) = \varnothing$, *for all* $1 \leq i < j \leq k$. ◇

Intuitively, in a step sequence of a well-formed acyclic net, no place 'receives' a token more than once. This ensures an unambiguous representation of causality in the behaviour of *acnet*. It then follows that in such a step sequence, no place is a pre-place of an executed step more than once, the order of execution of transitions does not influence the resulting marking, and each step sequence can be sequentialised to a firing sequence.

Definition 3.4.2 (well-formed acyclic net) *An acyclic net is* well-formed *if each transition occurs in at least one step sequence and all the step sequences are well-formed. The set of all well-formed acyclic nets is denoted by WFAN.* ◇

Checking that an acyclic net is well-formed can be done by looking at its firing sequences.

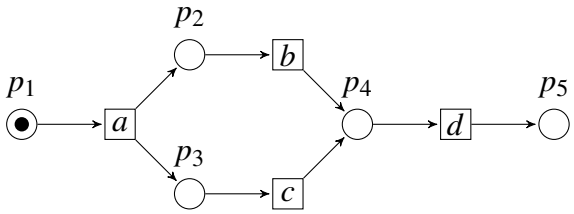

Fig. 3.6 Acyclic net which is not well-formed.

Example 6. The acyclic net in Figure 3.5 is well-formed, but the acyclic net in Figure 3.6 is not. The reason why the acyclic net in Figure 3.6 is not well-formed is that, intuitively, nothing prevents the execution of b and c first, followed by d. However, in such a case, it is impossible to determine whether d was caused by b or by c. In other words, the causal history for transition d is unclear, as it is not evident whether it is caused by transition b or transition c. Nonetheless, transition d will be executed whenever a token is generated by transition b or c. ◇

Proposition 3.4.1 ([8]) *An acyclic net acnet is well-formed iff each transition occurs in at least one firing sequence and all the firing sequences are well-formed.*

3.5 Communication structured acyclic nets (CSA-nets)

A communication structured acyclic net is a complete "database" consisting of several disjoint acyclic nets that can communicate through special buffer places. These buffer places allow instantaneous transfer of tokens and can involve a cycle when synchronous communication is modelled. Also, CSA-nets can exhibit backward and forward non-determinism.

Definition 3.5.1 (CSA-net) *A* communication structured acyclic net (or CSA-net) *is a tuple*

$$csan = (acnet_1, \ldots, acnet_n, Q, W) \qquad (n > 1)$$

such that:

1. $acnet_1, \ldots, acnet_n$ *are well-formed acyclic nets with disjoint sets of nodes (i.e., places and transitions). We also denote:*

$$\begin{aligned} P_{csan} &= P_{acnet_1} \cup \cdots \cup P_{acnet_n} \\ T_{csan} &= T_{acnet_1} \cup \cdots \cup T_{acnet_n} \\ F_{csan} &= F_{acnet_1} \cup \cdots \cup F_{acnet_n} \ . \end{aligned}$$

2. Q *is a finite set of* buffer places *disjoint from* $P_{csan} \cup T_{csan}$.

3. $W \subseteq (Q \times T_{csan}) \cup (T_{csan} \times Q)$ *is a set of arcs.*

4. *For every buffer place* q*:*

   (i) there is at least one transition t *such that* tWq*; and*

   (ii) if tWq *and* qWu *then transitions* t *and* u *belong to different component acyclic nets.*

   ◇

That is, in addition to requiring the disjointness of the component acyclic nets and the buffer places, it is also required that buffer places pass tokens between different acyclic nets.

As stated in [67], when sufficient resources are available, the standard Petri nets can be viewed as an asynchronous concurrency model with a firing (or step) sequence semantics.

When a step is executed, each of its transitions may also be fired. It is hard to express (structurally) that a transition that is enabled must (wait to) synchronise with another transition. However, it is easy to make an otherwise enabled transition to wait for another transition to be enabled by employing a message (in the form of a token that the second transition leaves in a designated input location of the first transition). The introduction of *buffer places* which are used by the CSA-nets, was motivated by such considerations.

Assume that transition t has an output buffer place q which is also an input buffer place for transition v. When t is fired, the token will be added to q. Such a token can either stay there to be used at a later time (following the standard asynchronous communication), or be immediately taken by v in the same step. Such a communication is like a telephone connection between a caller and callee with an answering machine.

The existing literature includes concepts similar to buffer places. According to [29], standard Petri net models lack a mechanism for synchronous communication, necessitating additional places and transitions that can lead to complex structures. The paper [29] suggested using communication channels to extend Coloured Petri Nets, by following the ideas described in Communicating Sequential Processes (CSP)[56], which is a programming notation used to describe patterns of interaction in concurrent systems, and Calculus of Communicating Systems (CCS)[85], which is a process algebra developed for describing computational processes in concurrent systems, where multiple computations happen simultaneously and potentially interact with each other. This type of extension is considered a robust description primitive (see, e.g., [74]). Another example is Petri nets with zero places by [25]. The primary characteristic of zero-safe nets is the fundamental concept of transition synchronisation, leading to the notion of concurrent 'transaction'. To achieve this, zero-safe nets include not only ordinary places, known as stable places, but also zero places, which cannot hold any tokens in observable markings. Moreover, they are based on firing sequences that progress from one 'stable' marking (a marking where all zero places are empty) to another stable marking, without impacting ordinary places along the way (e.g., [68] modelled protocols in which one partner can be ahead of the other during communication using such a feature).

The REO approach [17], which is a channel-based exogenous coordination model, builds complex coordinators, called connectors, from simpler ones and uses channels (similar to buffer places) to coordinate software components. Similarly, [110] employed REO channels to construct Petri net models for communication between systems.

Benefits in modelling using buffer places can be illustrated by the following scenario with two acyclic nets, $acnet_1$ and $acnet_2$. Assume that $acnet_1$ wants to communicate synchronously with $acnet_2$ through any of the transitions $t_1,\ldots,t_n$, and $acnet_2$ wants to synchronise with $acnet_1$ via any of the transitions $v_1,\ldots,v_n$. This could be achieved by gluing each t_i with each v_j, leading to the creation of n^2 new transitions. Alternatively, we could add just two new buffer places, q and q', along with $4n$ new arcs (i.e., $(t_1,q),\ldots,(t_n,q)$, $(v_1,q'),\ldots,(v_n,q')$, $(q',t_1),\ldots,(q',t_n)$, and $(q,v_1),\ldots,(q,v_n)$). Extending this scenario to include k acyclic nets with a requirement for synchronisation all of them using exactly one of the n transitions across these k nets, the direct modelling approach would necessitate creating n^k new transitions. However, achieving synchronisation could be equally effective by employing just k buffer places, significantly simplifying the model.

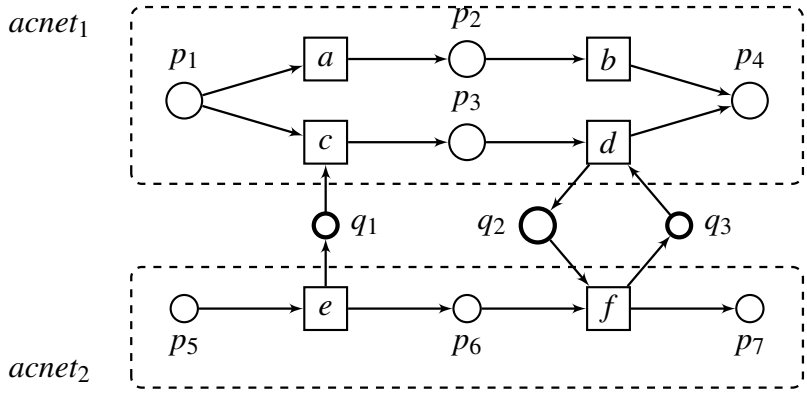

Fig. 3.7 Communication structured acyclic net *csan*.

Notation 2 (direct precedence in CSA-net) *Let* $csan = (acnet_1,\ldots,acnet_n,Q,W)$ *be a* CSA-*net,* $x \in P_{csan} \cup T_{csan} \cup Q_{csan}$*, and* $X \subseteq P_{csan} \cup T_{csan} \cup Q_{csan}$*. Then*

$$\begin{array}{lcllcl} \mathrm{pre}_{csan}(x) & = & \{y \mid y F_{csan} x \vee y W_{csan} x\}, & \mathrm{pre}_{csan}(X) & = & \bigcup\{\mathrm{pre}_{csan}(z) \mid z \in X\} \\ \mathrm{post}_{csan}(x) & = & \{y \mid x F_{csan} y \vee x W_{csan} y\}, & \mathrm{post}_{csan}(X) & = & \bigcup\{\mathrm{post}_{csan}(z) \mid z \in X\} \end{array}$$

denote direct predecessors and successors of x and X, respectively. Moreover,

$$P^{init}_{csan} = P^{init}_{acnet_1} \cup \cdots \cup P^{init}_{acnet_n} \;\; \textit{and} \;\; P^{fin}_{csan} = P^{fin}_{acnet_1} \cup \cdots \cup P^{fin}_{acnet_n}$$

are the initial *and* final *places of csan, respectively.* ◇

In CSA-nets, the buffer places can be considered as intermediary points that transfer tokens between different component acyclic nets. By holding tokens, buffer places facilitate the flow of control between individual acyclic nets. The causal nature of buffer places differs, for example, from that of places in the standard occurrence nets which introduce causal relationship between input transitions and output transitions. Instead, buffer places introduce what is usually called 'weak causality'. For example, suppose that t is an enabled input transition of a buffer place b, and v is an output transition of b (with pre-places disjoint from the pre-places of t) whose all pre-places other than b are marked. Then, two execution scenarios are possible: t is fired first and then v is fired, or t and v are fired simultaneously as the step $\{t,v\}$. The second scenario would not be possible if b was an ordinary place. Intuitively, weak causality in this case means that t is executed 'not later' than v. Note that weak causality of buffer places was first discussed in [70]

Proposition 3.5.1 ([8]) *For every* CSA*-net csan,*

$$P_{acnet} \uplus Q_{acnet} = P^{init}_{acnet} \uplus \mathrm{post}_{csan}(T_{csan}) = P^{fin}_{acnet} \uplus \mathrm{pre}_{acnet}(T_{acnet}).$$

Example 7. Figure 3.7 shows a CSA-net which is composed of two acyclic nets such that $\mathrm{post}_{csan}(p_1) = \{a,c\}$, $\mathrm{pre}_{csan}(p_4) = \{b,d\}$, and $\mathrm{post}_{csan}(e) = \{q_1, p_6\}$. Moreover, transitions e and c are communicating *asynchronously*, so they can be executed at the same time, or e then c. However, c cannot be executed before e in *asynchronous* communication. The relation between e and c can be captured by the presence of $q_1 \in \mathrm{post}_{csan}(e) \cap \mathrm{pre}_{csan}(c)$. On the other hand, d and f must be executed simultaneously as they are involved in *synchronous* communication. ◇

Note that CSA-net can exhibit backward and forward non-determinism. Moreover, they can additionally contain synchronous cycles involving only buffer places.

Definition 3.5.2 (CSO-net) *A* communication structured occurrence net (or CSO-net) *is* $cson \in CSAN$ *such that:*

1. *The component acyclic nets belong to ON.*
2. $|\mathrm{pre}_{cson}(q)| = 1$ *and* $|\mathrm{post}_{cson}(q)| \leq 1$, *for every* $q \in Q_{cson}$.
3. *No place in* P_{cson} *belongs to a cycle in the graph of* $F_{cson} \cup W_{cson}$.

The set of all CSO*-nets is denoted by CSON.* ◇

A CSO-net provides a full record of all causal dependencies between the events involved in a single 'causal history'. In terms of behaviour defined later, CSO-nets exhibit both backward determinism and forward determinism. However, we also consider CSA-nets with only forward nondeterminism.

Example 8. Figure 3.8 depicts a CSO-net. ◇

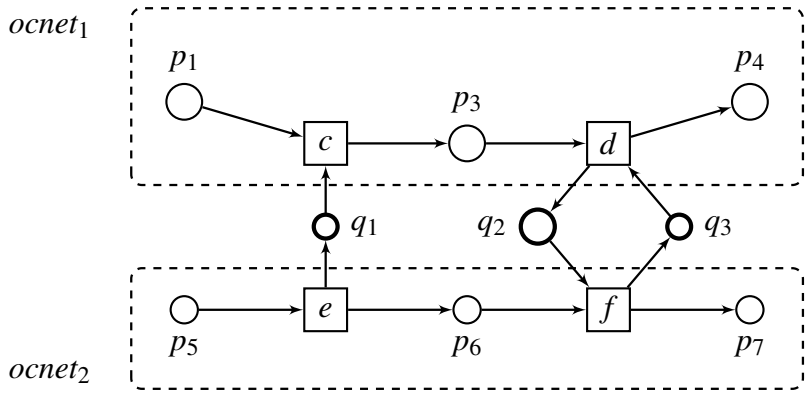

Fig. 3.8 Communication structured occurrence net.

Definition 3.5.3 (BDCSA-net) *A* backward deterministic communication structured acyclic net (or BDCSA-net) *is* $bdcsan \in CSAN$ *such that:*

1. *The component acyclic nets belong to BDAN.*

2. $|\mathrm{pre}_{bdcsan}(q)| = 1$, *for every* $q \in Q_{bdcsan}$.

The set of all BDCSA*-nets is denoted by BDCSAN.* ◇

BDCSA-net is a CSA-net providing full and unambiguous record of backward causal dependencies. BDCSA-nets exhibit backward determinism, but forward determinism is not required. Scenarios for CSA-nets can be defined similarly as for acyclic nets.

Below, for two CSA-nets,

$$csan = (acnet_1, \ldots, acnet_n, Q, W) \text{ and } csan' = (acnet'_1, \ldots, acnet'_n, Q', W'),$$

we denote $csan \sqsubseteq csan'$ if

1. $acnet_i \sqsubseteq acnet'_i$, for every $1 \leq i \leq n$.
2. $Q \subseteq Q'$.
3. $\mathrm{pre}_{csan}(t) = \mathrm{pre}_{csan'}(t)$ and $\mathrm{post}_{csan}(t) = \mathrm{post}_{csan'}(t)$, for every $t \in T_{csan}$.

Definition 3.5.4 (scenario and maximal scenario of CSA-net) *Let csan be a* CSA*-net. A* scenario *of csan is a* CSO*-net cson such that* $cson \sqsubseteq csan$*. Moreover, cson is* maximal *if there is no scenario* $cson' \neq cson$ *such that* $cson \sqsubseteq cson'$.
The set of all scenarios of csan is denoted by $\mathrm{scenarios}(csan)$*, and the set of all maximal scenarios of csan is denoted by* $\mathrm{maxscenarios}(csan)$. ◇

Intuitively, the scenarios of a CSA-net are its subnets that begin with the same initial marking and exhibit both backward and forward determinism. This means they represent executions with well-defined causal relationships. Note that scenarios depict potential executions (concurrent histories) that adhere to the dependencies imposed by the flow relation. Specifically, this implies that all transitions can be executed and places marked at some point during an execution. Moreover, maximum scenarios are considered complete, as they cannot be extended any further.

Proposition 3.5.2 ([8]) $\mathrm{maxscenarios}(cson) = \{cson\}$*, for every* CSO*-net cson.*

That is, an occurrence net has only one maximal scenario (itself), which can be understood as a precise depiction of a single execution history.

Example 9. Figure 3.8 shows a maximal scenario for the CSA-net in Figure 3.7. ◇

3.6 Step sequence semantics of CSA-nets

This section will introduce notions related to the behaviour of CSA-nets.

Definition 3.6.1 (step and marking of CSA-net) *Let csan be a* CSA*-net.*

1. $\text{steps}(csan) = \{U \in \mathbb{P}(T_{csan}) \mid \forall t \neq u \in U : \text{pre}_{csan}(t) \cap \text{pre}_{csan}(u) = \varnothing\}$ *are the* steps.
2. $\text{markings}(csan) = \mathbb{P}(P_{csan} \cup Q_{csan})$ *are the* markings.
3. $M^{init}_{csan} = P^{init}_{csan}$ *is the default* initial *marking.* ◇

That is, in CSA-net steps can involve events from one or more acyclic nets, provided that $\text{pre}_{csan}(t) \cap \text{pre}_{csan}(u) = \varnothing$ for every $t \neq u$. Furthermore, markings in CSA-net can consist of both places and buffer places. The initial marking in a CSA-net is made up of the initial markings of all acyclic nets participating in the CSA-net. Note that buffer places are excluded from initial marking.

Example 10. $\text{steps}(csan_1) = \{U \in \mathbb{P}(\{a,b,c,d,e,f\}) \mid a \in U \implies c \notin U\}$ and $M^{init}_{csan_1} = \{p_1, p_5\}$, for the CSA-net in Figure 3.7. ◇

Definition 3.6.2 (enabled and executed step of CSA-net) *Let M be a marking of a* CSA*-net csan.*

1. $\text{enabled}_{csan}(M) = \{U \in \text{steps}(csan) \mid \text{pre}_{csan}(U) \subseteq M \cup (\text{post}_{csan}(U) \cap Q)\}$ *are the steps* enabled *at M.*
2. *A step* $U \in \text{enabled}_{csan}(M)$ *can be* executed *yielding a new marking*

$$M' = (M \cup \text{post}_{csan}(U)) \setminus \text{pre}_{csan}(U) \,.$$

This is denoted by $M[U\rangle_{csan} M'$. ◇

That is, enabling a step in CSA-nets requires having all input places of the acyclic nets present in the marking (global state), including buffer places. Also, if an input buffer place is not present in the step, it has to be an output place for a transition in the step. In other words, a step in CSA-nets not only uses tokens locally for each acyclic net but also uses tokens from different acyclic nets that are present in buffer places. This mechanism allows for synchronising transitions from different acyclic nets.

The dynamic behaviour of *csan* is captured by step sequences and mixed step sequences, as follows.

Definition 3.6.3 (mixed step sequence and step sequence of CSA-net) *Let* $M_0, \ldots, M_k$ $(k \geq 0)$ *be markings and* $U_1, \ldots, U_k$ *be steps of a* CSA*-net csan such that* $M_{i-1}[U_i\rangle_{csan} M_i$*, for every* $1 \leq i \leq k$.

1. $\mu = M_0 U_1 M_1 \ldots M_{k-1} U_k M_k$ *is a* mixed step sequence from M_0 to M_k.

2. $\sigma = U_1 \ldots U_k$ *is a* step sequence from M_0 to M_k.

The above two notions are denoted by $M_0[\mu\rangle\rangle_{csan} M_k$ *and* $M_0[\sigma\rangle_{csan} M_k$*, respectively. Moreover,* $M_0[\sigma\rangle_{csan}$ *denotes that* σ *is a step sequence* enabled M_0*, and* $M_0[\rangle_{csan} M_k$ *denotes that* M_k *is* reachable from M_0. ◇

Note that if $k = 0$ then $\mu = M_0$ and the corresponding step sequence σ is the *empty* sequence denoted by λ.

Example 11. Figure 3.9 shows a CSA-net. Some of its mixed step sequences are:

$$\begin{aligned}
\mu_1 &= \{p_1, p_7\}\{d\}\{p_2, p_5, p_7\}\{a, b, c\}\{p_3, p_6, p_8\}\{e\}\{p_4, p_8\} \\
\mu_2 &= \{p_1, p_7\}\{d\}\{p_2, p_5, p_7\}\{a\}\{p_3, q_1, p_5, p_7\}\{b, c\}\{p_3, p_6, p_8\}\{e\}\{p_4, p_8\} \\
\mu_3 &= \{p_1, p_7\}\{d\}\{p_2, p_5, p_7\}\{a, b\}\{p_3, q_1, p_6, q_2, p_7\}\{e\}\{, q_1, q_2, p_4, p_7\}\{c\}\{p_4, p_8\}.
\end{aligned}$$

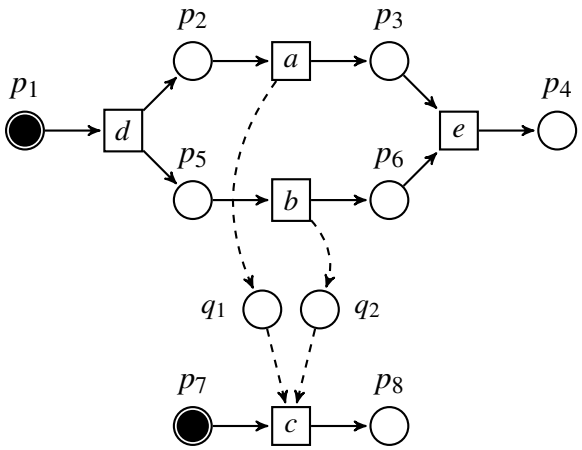

Fig. 3.9 A CSA-net model.

Definition 3.6.4 (behaviour of CSA-net) *The following sets capture various behavioural notions related to step sequences and reachable markings of a* CSA*-net csan. This definition assumes that the starting point is the default initial marking.*

1. $\text{sseq}(csan) = \{\sigma \mid M^{init}_{csan}[\sigma\rangle_{csan} M\}$ step sequences.
2. $\text{mixsseq}(csan) = \{\mu \mid M^{init}_{csan}[\mu\rangle\rangle_{csan} M\}$ mixed step sequences.
3. $\text{maxsseq}(csan) = \{\sigma \in \text{sseq}(csan) \mid \neg\exists U : \sigma U \in \text{sseq}(csan)\}$ maximal step sequences.
4. $\text{maxmixsseq}(csan) = \{\mu \in \text{mixsseq}(csan) \mid \neg\exists U, M : \mu U M \in \text{mixsseq}(csan)\}$ maximal mixed step sequences.
5. $\text{reachable}(csan) = \{M \mid M^{init}_{csan}[\rangle_{csan} M\}$ reachable markings.
6. $\text{finreachable}(csan) = \{M \mid \exists \sigma \in \text{maxsseq}(csan) : M^{init}_{csan}[\sigma\rangle_{csan} M\}$ final reachable markings.

Example 12. The following hold for the CSA-net in Figure 3.7.

1. $\text{sseq}(csan_1) = \{\lambda, \{a,e\}, \{a,e\}b, \cdots\}$.
2. $\text{mixsseq}(csan_1) = \{\{p_1,p_5\}, \{p_1,p_5\}\{a,e\}\{p_2,p_6,q_1\}, \ldots\}$.

3. maxsseq$(csan_1) = \{ab, aeb, abe, \{a,e\}b, a\{b,e\}, ec\{d,f\}, \{e,c\}\{d,f\}\}$.

4. maxmixsseq$(csan_1) = \{\{p_1,p_5\}\{a,e\}\{p_2,p_6,q_1\}b\{p_4,p_6,q_1\}, \dots\}$.

5. reachable$(csan_1) = \{\{p_1,p_5\}, \{p_2,p_6,q_1\}, \dots\}$.

6. finreachable$(csan_1) = \{\{p_4,p_6,q_1\}, \{p_4,p_7\}\}$. ◇

3.7 Well-formed CSA-nets

A fundamental requirement for CSA-net is well-formedness, which essentially guarantees a clear representation of causality in the behaviours they represent. The concept of a well-formed CSA-net is derived from the idea of a well-formed step sequence.

Definition 3.7.1 (well-formed step sequence of CSA-net) *A step sequence $U_1 \dots U_k$ of a* CSA*-net csan is* well-formed *if the following hold:*

1. $\text{post}_{csan}(t) \cap \text{post}_{csan}(u) = \varnothing$, *for every* $1 \leq i \leq k$ *and all* $t \neq u \in U_i$.

2. $\text{post}_{csan}(U_i) \cap \text{post}_{csan}(U_j) = \varnothing$, *for all* $1 \leq i < j \leq k$. ◇

Intuitively, in CSA-net, a well-formed step sequence means that no place or buffer place is filled by a token more than once in any given step sequence. In other words, no transition is executed and no token is consumed more than once. The order of execution of transitions does not influence the resulting marking.

Definition 3.7.2 (well-formed CSA-net) *A* CSA*-net is* well-formed *if each transition occurs in at least one step sequence and all step sequences are well-formed. The set of all well-formed* CSA*-nets is denoted by WFCSAN.* ◇

Example 13. Figure 3.10 shows a non-well-formed CSA-net. The reason is that $\text{post}_{csan}(c) \cap \text{post}_{csan}(e) \neq \varnothing$ and $a\{b,d\}\{c,e\}$ is a valid step sequence. In other words, it is not clear at place p_6 whether the token transition is from c or e. ◇

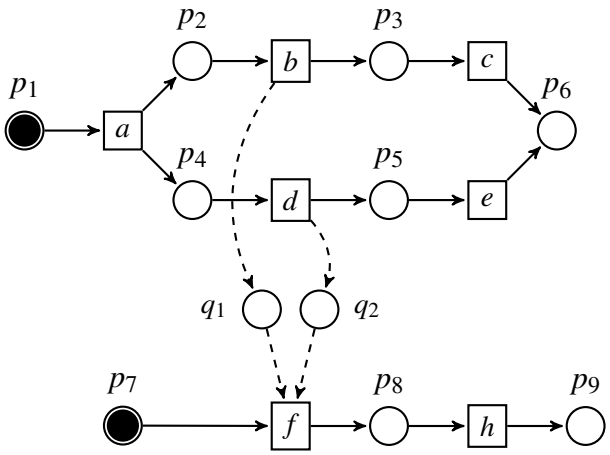

Fig. 3.10 A CSA-net model.

By satisfying backward and forward determinism, CSO-nets enjoy some essential behavioural properties 'for free'.

Proposition 3.7.1 ([8]) *Each* CSO*-net is well-formed.*

Proposition 3.7.2 ([8]) *A* CSA*-net is well-formed iff each transition occurs in at least one scenario, and each step sequence is a step sequence of at least one scenario.*

Proposition 3.7.3 ([8]) *Step sequences of a well-formed* CSA*-net do not contain multiple occurrences of transitions.*

Each step sequence σ of a well-formed CSA-net *csan* induces a scenario $\text{scenario}_{csan}(\sigma) = \text{scenario}_{csan}(\bigcup\sigma)$ such that $\sigma \in \text{maxsseq}(\text{scenario}_{csan}(\sigma))$. Thus, in a well-formed CSA-net, different step sequences may generate the same scenario, and conversely, each scenario is generated by at least one step sequence. Moreover, two maximal step sequences generate the same scenario iff their executed transitions are identical.

3.8 Conclusion

This chapter introduced the basic notions and examples related to acyclic nets and CSA-nets, including their structure, semantics, step sequences, executed steps, and well-formedness. Additionally, we discussed the use of CSA-nets as an extension of CSO-nets, enabling both

asynchronous and synchronous communication between different subsystems through buffer places. This makes them an appropriate tool for modelling and visualising event-based system behaviour, including concurrent and synchronisation behaviour. The next chapter will introduce a new extension of acyclic nets and CSA-nets through a new class called *parameterised* CSA*-nets*.

Chapter 4

Parametrisation of CSA-nets

Communication Structure Acyclic Nets (CSA-nets) are a Petri net-based formalism used to represent the behaviour of Complex Evolving Systems (CESs). CSA-nets consist of sets of acyclic nets that communicate with each other through a set of buffer places. However, this can generate an excessive number of buffer places, making the model hard to visualise and analyse. Additionally, having several components that perform similar tasks can result in the development of a complicated model. This chapter introduces two extensions for CSA-nets, which are used to simplify the depiction of the relationships between interacting system components represented by acyclic nets. Specifically, it introduces a way of folding buffer places to address the issue of a large number of buffer places. In addition, it introduces parametrisations for CSA-nets which uses multi-coloured tokens. The combination of these techniques should lead to improved visualisation and analysis of large and complex CSA-nets. We apply mechanisms found in the domain of coloured Petri nets.

4.1 Introduction

Complex evolving systems are composed of a large number of subsystems that interact with each other concurrently. This growth necessitates the integration and interaction of several subsystems, supporting, e.g., processing, and analysis. They are often characterised by dynamically changing structures and features, as well as intricate behavioral patterns.

Moreover, these system can have large amount of data stored on servers, hard drives, and in the cloud. In particular, there is a vast amount of data available, and this continues to grow due to advances in digital sensors, communications, and storage technologies. The addition of new features can affect the behavior of the entire system. Thus, these systems are highly complex in terms of interpreting and visualising their behavior [49]. Therefore, effective visualisation of systems comprising large amounts of data can greatly enhance the ability to analyse and understand them, thereby extracting useful knowledge which can assist in making the right decisions. One example of CESs with large data volumes is found in cybercrime investigation. In such applications, the generated data is crucial, necessitating high-quality analysis. A major specific advantage in cybercrime investigations is that such analysis enables the early detection of crimes, thus facilitating crime prevention [39]. For instance, it allows investigators to establish connections between criminal activities and locations through data representations, which in turn facilitates the detection of criminal activities [95]. This is achieved by extracting valuable knowledge from large and complex datasets and identifying useful information from various data sources.

Although CSA-nets provide a means to analyse and visualise complex systems, challenges arise as the data scale increases. Specifically, the growing number of nodes makes the model more difficult to understand. This growth leads to an increase in the number of places, transitions, and buffer places, which often complicates the model, making it challenging to analyse and visualise effectively. Furthermore, a limitation of CSA-nets is to only present one token in a place at any given time, which in turn limits their ability to represent several similar processes simultaneously (essentially, each process must be modeled individually). Thus, by abstracting or parametrising certain components and allowing places in CSA-nets to represent multiple tokens, we can enhance the representation and foster a clearer understanding of the problem under study. In particular, it could assist investigators in detecting causality and dynamic behaviors between events, which would aid in analysing such systems.

Parametrisation, a form of abstraction in modeling, is a method of representing a common part for all processes or objects in a model, which can be used later for different inputs [30]. This makes complex models easier and more effectively represented in the model through

simplified processes. That is, we can view parametrisations as a way to simplify processes while maintaining their validity. However, manipulation of models using abstractions is an important part of the model development process. The interesting point about parametrisation is that it simplifies the process of testing the effect of changing inputs on the outputs. This can be likened to a function definition in programming, where the programmer calls the function with different inputs to test the outputs. This facilitates a deeper understanding of the relationships between inputs and outputs. Additionally, it streamlines the system tracing process by automatically adjusting the results of one parameter in response to modifications in others. Another benefit of parametrisation is that it reduces the number of components in the process, which results in a much more pleasant visualisation, and this is where our contribution lies. With this in mind, parametrisation can be viewed as a approach for improving system visualisation. Usually, when modelling systems, each tiny piece of information is modelled. If CSA-nets are used for this task, the number of the components can be large, resulting in a complex model.

This chapter focuses on improving the visualisation of CSA-nets by addressing two main challenges. Firstly, it introduces the concept of 'Master Buffer Places' that fold potentially numerous buffer places into a *single node*. In essence, master buffer places offer a parametrisation for a set of buffer places. This extension provides a folding mechanism to CSA-nets, effectively tackling the concern of excessive buffer places. Secondly, we present a new class of CSA-nets, called 'parametrised CSA-nets', which extends the single-token concept. This approach aims to enable places to accommodate multiple tokens, each distinguished by specific parameters or colours.

4.2 Related work

Modelling complex and evolving systems can help in solving real-world problems safely and efficiently by providing an important method of system analysis that is easily understood. However, modelling such systems are challenging task due to the high volume of interrelated components. In other words, gaining a clear understanding of how CESs evolve over varying

variable values is not an easy task, as it might require modifications in the model. Moreover, as the process under study becomes more complex, the visual form of modelling also becomes complex, which may hinder understanding the model correctly.

Parametrisation has been viewed as an approach that can help in improving the models' complexity, in particular in Petri nets, by improving their visualisation. However, Petri nets encompass a wide range of different classes. For example, [52] extended the traditional Petri net approach by integrating a hierarchical structure into parametrised Petri nets (PPNs). This simplified model complexity by parametrising transitions and token types, enabling the management of systems with numerous places and transitions, and thus yielding a visually appealing and user-friendly graphical representation. Moreover, Ermel and Weber [43] accommodated the diversity of Petri net classes and developed the Parametrised Petri Net Kernel (PNK), a flexible framework for various Petri net-like structures. This kernel facilitates the modification of Petri nets, allowing users to create custom PNs tailored to their needs by defining specific net characteristics. It supports high-level abstraction for CESs while simplifying the representation of system component interactions.

The concept of colour is also proposed in the literature to facilitate the modelling and validation of systems where concurrency, communication, and synchronisation play major roles. In this context, Jensen et al. [60] introduced Coloured Petri Nets (CPNs), which organises a model as a set of modules, taking into account the time required to execute events in the modelled system. This approach aims to attach information at the token level, resulting in a compact representation of CESs such as [78]. In the same context, [131] using CPNs to model how social robots and users communicate continuously. They did this by showing the robots' behavior in graphs, making it easier to spot complex patterns of communication and understand events happening together or separately. This method helped them gain a deeper insight into where privacy might be at risk. Similarly, [51] used CPNs to create a model of a defense system that hides its identity to trick attackers. This use of CPNs has been effective in closely examining how data is captured, controlled, and collected. Moreover, [44] employed CPNs to model network-controlled systems characterised by high traffic that could be vulnerable to faults and cyber attacks. In particular, using CPNs, they

modelled a sensor fault and correction mechanism using to test the system's performance. In the same direction [96] improved the attack trees, a well-known graphical security model, using CPNs to focus on system scalability investigation.

CSA-nets are another robust modelling tool able to describe systems with interrelated components. However, a significant limitation lies in their inability to handle CESs with a high volume of components, and here is where our work comes in. Specifically, in this chapter, we introduce a novel class of CSA-nets that harnesses the combined power of parametrisation and colored tokens. This innovative approach enables CSA-nets to offer more comprehensive and meaningful insights into CESs while abstracting away intricate details.

4.3 Towards master buffer places

Despite the fact that CSA-nets rely on a robust and effective structure that reduces complexity compared to other representations, they lack the ability to visualise and analyse large amounts of data simultaneously. Specifically, only one token can be represented in one buffer place at any given time. In other words, a new token cannot be introduced into the same place until the previous one has been processed. That is, it needs at least one buffer place for each link between transitions in different acyclic nets. This approach can lead to a surge in buffer places when the CSA-nets scale up, making the model challenging to visualise and analyse. However, the complexity is accentuated when dealing with acyclic nets that represent many communication activities between them. In particular, these activities introduce additional layers of intricacy, demanding careful handling to maintain the fidelity of the model. One possible solution to address this issue is to fold buffer places into one node. This can be achieved by allowing buffer places to accept and represent more than one token at a time. Nevertheless, the primary challenge of handling scalability with large datasets will be addressed by leveraging the semantic information available due to the semantic structuring embedded in CSA-nets.

Example 14. Figure 4.1 shows a CSA-net in which two acyclic nets, $acnet_1$ and $acnet_2$, communicate through seven buffer places, q_0, q_1, ..., q_6. Each buffer place can represent

and hold only one token at a time. Consequently, if the depicted model were to expand, the number of buffer places could significantly increase. ◇

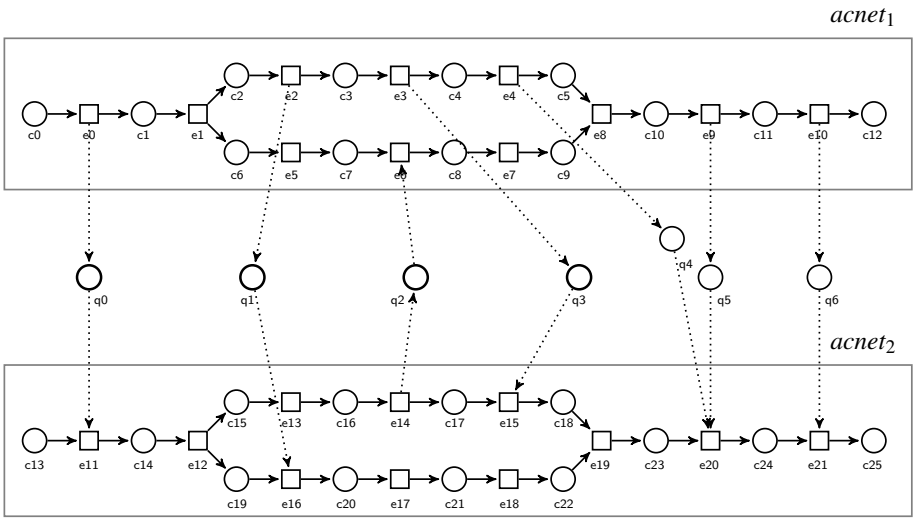

Fig. 4.1 A CSA-net model with 7 buffer places.

4.4 Master buffer places (MBPs)

In CSA-nets, buffer places link different acyclic nets with each other and allow the transfer of tokens yielding two types of communication: synchronous and asynchronous. The aim of this work is to extend and refine CSA-nets to effectively visualise large amounts of data, thereby enhancing model visualisation and analysis. Specifically, we propose introducing Master Buffer Places (MBPs) to bring conciseness to the CSA-net by collapsing (folding) multiple buffer places into MBPs. In essence, MBPs can hold and represent more than one token at a time, preventing the need for an excessive number of buffer places. This can enable the component acyclic nets to communicate through a unique master buffer place. Within an MBP, tokens will be differentiated by unique colors. A specific token will be presented in the MBP without interference from other tokens in each execution, ensuring it is 'color-safe', provided that the original CSA-net was safe. This improvement, reducing the number of buffer places, would optimise CSA-net visualisation, resulting in a model that is both more readable and comprehensible without affecting the behaviours they represent.

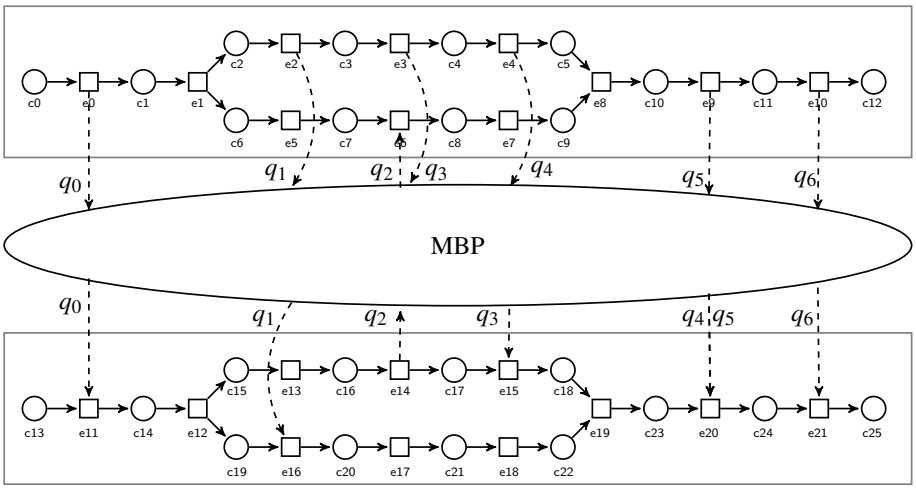

Fig. 4.2 A version of the CSA-net of Figure 4.1 with a master buffer place.

Example 15. Figure 4.2 represents an adaptation of the CSA-net in Figure 4.1, where the seven buffer places, q_0, q_1, ..., q_6, are folded into a single MBP q. Additionally, annotations have been added to the arcs to indicate which coloured tokens will be inserted into the MBP q when transitions are executed. ◇

In essence, Figure 4.2 folds seven buffer places, q_0, q_1, ..., q_6, into one MBP by collapsing them into a single node q, which can accommodate more than one coloured token at a time. We have also added annotations on the arcs adjacent to the new places and modified the execution rules accordingly. It is important to note that the annotations q_0, q_2, ..., q_6 represent a set of the original buffer places (now treated as coloured tokens). Note also that the execution rules follow the standard idea of coloured Petri nets, ensuring that the tokens 'flowing' along the arcs match the annotations. The tokens in master buffer places are simply the original buffer places. This approach could significantly enhance the visualisation, understanding, and analysis of CSA-nets by minimising their structural complexity.

Example 16. Figure 4.3(a) depicts an example of a CSA-net which intuitively combines two executions (*conflict*): one involving a and c, and the other involving b, d, c, and e. Additionally, there are two buffer places, q_1 and q_2: one between a, b and c, and another between d and e. Figure 4.3(b) shows the CSA-net of Figure 4.3(a), where the two buffer places have been replaced by a single MBP q. ◇

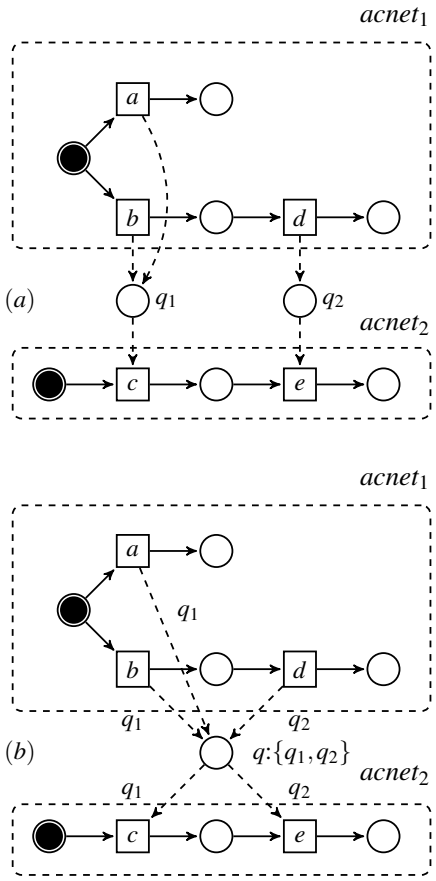

Fig. 4.3 CSA-net and a corresponding version with one master buffer place.

4.4.1 Managing MBPs

Although the folded buffer simplifies the visualisation by reducing the number of buffer places in the model, it introduces certain challenges. Specifically, folding multiple buffer places into a single MBP can lead to crossings between acyclic nets, especially when the model includes several acyclic nets (subsystems). A primary challenge, therefore, is to ensure a clear and well-structured display without undesirable arc crossings.

Example 17. Figure 4.5 shows a version of the CSA-net of Figure 4.4, where three acyclic nets communicate with each other through a single master buffer place. This MBP folds the four original buffer places: q_1, q_2, q_3, and q_4. ◇

The last example shows that introducing a single MBP (or a small number of MBPs) could lead to an excessive number of crossings. Specifically, unless it is placed 'outside' the outline of the underlying occurrence nets — which is generally not feasible due to the

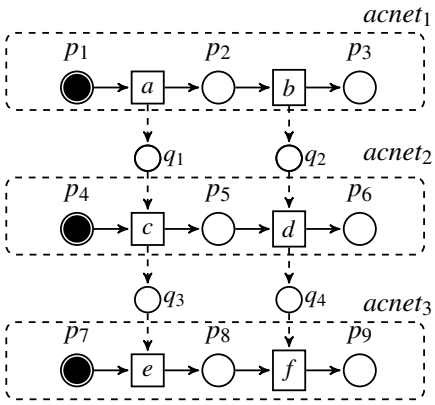

Fig. 4.4 CSA-net with four buffer places.

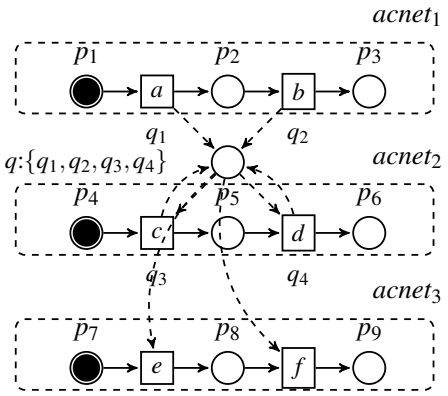

Fig. 4.5 A version of the CSA-net of Figure 4.4 with one master buffer place.

large size of the individual acyclic nets making them too large to fit on a single screen — such crossings would be unavoidable. Consequently, folding buffer places into MBPs can result in extensive crossings, thereby complicating the model's analysis and visualisation and making its understanding more challenging. One possible solution to this issue is to create sufficiently many MBPs. This leads to an important problem: ' how to determine the number and placement of MBPs (either automatically or semi-automatically) to optimise the visualisation of CSA-nets?' This is the topic of Chapter 5.

Example 18. Figure 4.6 shows a version of the CSA-net of Figure 4.4 after creating two MBPs. This clearly leads to a better visualisation than the attempt shown in Figure 4.5. ◇

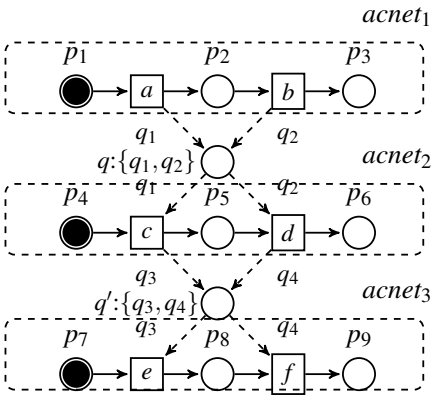

Fig. 4.6 A version of the CSA-net of Figure 4.4 with two master buffer places.

4.5 Parametrised CSA-nets (PCSA-nets)

Parametrised Petri nets fold places and transitions into a CPN model, resulting in a more compact model with fewer places and transitions [60]. In this section, we will apply the same concept of folding to the components of CSA-nets while retaining their primary rules (specifically, being acyclic). We will introduce a new class of CSA-nets, called *Parametrised* CSA-*nets* (PCSA-nets). PCSA-nets will allow places to accept multiple tokens, which are distinguished by 'colours' to differentiate tokens carrying various data types. This extension streamlines the modelling of complex systems by allowing individual places or transitions to handle multiple token types, rendering the model both more clear and concise.

Example 19. Figure 4.7(b) depicts a parametrised version of a simple CSA-net of Figure 4.7(a), where two (conflicting) parts of $acnet_2$ are folded (collapsed) into a single parametrised structure $acnet_2'$. Clearly, the two parts have the same net structure. Thus, the idea is to determine the set of components that are behaving identically, and then represent them as a single substructure. Figure 4.7(b) uses typed parameters to achieve the desired effect through passing tokens to parametrised transition. Examples of firing sequences in Figure 4.7(a) are: abdce, abdcf, abedc, and adfbc. Note that the execution of behaviours will not change after parametrising. ◇

That is, this process can allow for the reuse of the model multiple times and increase comprehension, making larger systems under investigation easier to handle.

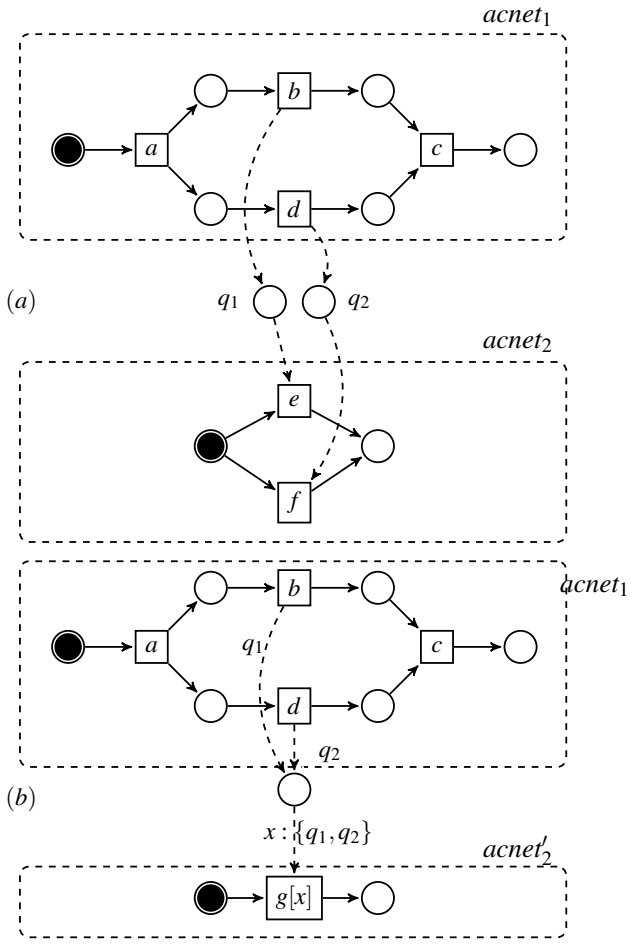

Fig. 4.7 A CSA-net with an arc adjacent to the MBP annotated with variable x with the domain $\{q_1, q_2\}$, and a mapping g defined so that $g[q_1] = e$ and $g[q_2] = f$.

4.6 Formalisation

In this section, we provide a formalisation of the approach proposed in this chapter. We first introduce parametrised acyclic nets, and then parametrised communication structured acyclic nets.

4.6.1 Parametrised acyclic nets

Definition 4.6.1 (parametrised acyclic net) *A* parametrised acyclic net (or PA-net) *is a tuple* $pacnet = (P, T, F, acnet, \iota, col)$*, where:*

- $acnet = (P', T', F')$ *is a well-formed acyclic net.*

- *col is a finite set of* colours *(or* parameters*).*
- *P ($P \neq \varnothing$) and T are disjoint finite sets of* places *and* transitions*, respectively.*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a* flow relation*.*
- *$\iota : P \cup T \to \mathbb{P}(P' \cup T') \setminus \{\varnothing\}$ is a* node annotation *mapping.*

We denote, for all $x \in P \cup T$, $p \in P$, and $u \in T'$:

$$\begin{array}{lcllcl} \mathrm{pre}_{pacnet}(x) & = & \{z \mid zFx\} & \mathrm{pre}_{pacnet}(u,p) & = & \iota(p) \cap \mathrm{pre}_{acnet}(u) \\ \mathrm{post}_{pacnet}(x) & = & \{z \mid xFz\} & \mathrm{post}_{pacnet}(u,p) & = & \iota(p) \cap \mathrm{post}_{acnet}(u)\ . \end{array}$$

We then assume that the following hold.

1. *$(P \cup T) \cap (P' \cup T') = \varnothing$, $\bigcup \iota(P) = P'$, and $\bigcup \iota(T) = T'$.*
2. *For all $t \in T$ and $u \in \iota(t)$ we have the following, where $\uplus$ is assumed to be applied to non-empty sets:*[1]

$$\begin{array}{lcl} \mathrm{pre}_{acnet}(u) & = & \uplus\{\mathrm{pre}_{pacnet}(u,p) \mid p \in \mathrm{pre}_{pacnet}(t)\} \\ \mathrm{post}_{acnet}(u) & = & \uplus\{\mathrm{post}_{pacnet}(u,p) \mid p \in \mathrm{post}_{pacnet}(t)\}\ . \end{array}$$

3. *If $u \in T'$ and $R \subseteq P$ are such that*

$$\mathrm{pre}_{acnet}(u) = \biguplus\{\mathrm{pre}_{pacnet}(u,p) \mid p \in R\}$$

then then there is $t \in T$ such that $u \in \iota(t)$ and $\mathrm{pre}_{pacnet}(t) = R$.

4. *For every $p \in P$,*

$$\iota(p) \setminus M^{init}_{acnet} = \bigcup\{\mathrm{post}_{pacnet}(u,p) \mid\ u \in \bigcup \iota(\mathrm{pre}_{pacnet}(p))\}\ .$$

5. *For every $p \in P^{init}_{acnet}$, there is exactly one place $p^{init} \in P$ such that $p \in \iota(p^{init})$.* ◇

[1] Thus, for example, $\{\mathrm{pre}_{pacnet}(u,p) \mid p \in \mathrm{pre}_{pacnet}(t)\}$ is a partition of $\mathrm{pre}_{acnet}(u)$.

Note that we do not assume that F is acyclic since the acyclity of the behaviour of *acnet* will lead to the acyclicity of the behaviour of *pacnet*.

Intuitively, $\iota(t)$ specifies different 'modes' of the execution of a transition $t \in T$, and $\iota(p)$ specifies the places of *acnet* which can be 'present' in $p \in P$ in the executions of *pacnet*. Definition 4.6.1(2) means that, for each mode $u \in \iota(t)$, the arcs incoming to t 'deliver' the input places of u, and similarly for the output arcs. Definition 4.6.1(3) means that, for any potential distribution of the 'preset' of $u \in T_{acnet}$, there is a transition $t \in T$ with the mode u which can input this ditribution of the preset. Definition 4.6.1(4) states that $\iota(p)$ is determined by the modes of the input transitions.

Example 20. Figure 4.8 shows a parametrised acyclic net $pacnet = (P,T,F,acnet,\iota,col)$ such that

$$acnet = (\{p_1,p_2,p_3,p_4\},\{a,b\},\{(p_1,a),(a,p_2),(p_3,b),(b,p_4)\}),$$

and

$$\begin{array}{llll} P & = \{r_1,r_2\} & T = \{t\} & F = \{(r_1,t),(t,r_2)\} \\ \iota(r_1) & = \{p_1,p_3\} & \iota(t) = \{a,b\} & \iota(r_2) = \{p_2,p_4\} \\ col & = \{1,2\}. & & \end{array}$$

We then have, for example, $\mathrm{pre}_{pacnet}(a,r_1) = \{p_1\}$ and $\mathrm{post}_{pacnet}(b,r_2) = \{p_4\}$. ◇

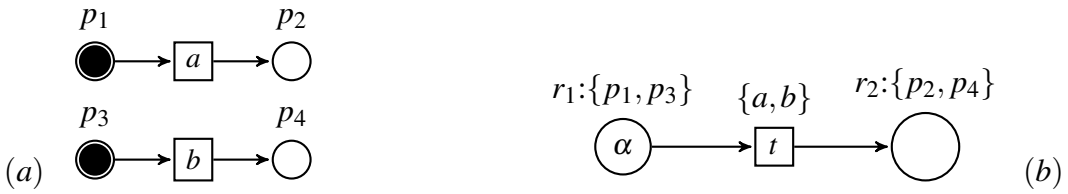

Fig. 4.8 An acyclic net ((a)); and its parametrised version (PA-net) with $col = \{1,2\}$, where $\alpha = \{p_1{:}1, p_1{:}2, p_3{:}1, p_3{:}2\}$ ((b)).

To capture the behaviour of *pacnet*, we will use parametrised places and transitions:

$$P_{pacnet} = \{(p,r,c) \in P' \times P \times col \mid p \in \iota(r)\}$$
$$T_{pacnet} = \{(u,t,c) \in T' \times T \times col \mid u \in \iota(t)\}\ .$$

Intuitively, (p,r,c) is an instance of place p of the original acyclic net coloured by c, which resides in the place r of *pacnet*. (Another way of interpreting (p,r,c) is that it represents coloured token (p,c) present in the place r.) Similarly, (u,t,c) represents mode in which transition t operates as if it were transition u of the original acyclic net coloured by c. Moreover, we denote for $(u,t,c) \in T_{pacnet}$ (and similarly for subsets of T_{pacnet}):

$$\begin{aligned}\mathrm{pre}_{pacnet}((u,t,c)) &= \bigcup\{\mathrm{pre}_{acnet}(u,p) \times \{p\} \times \{c\} \mid p \in \mathrm{pre}_{pacnet}(t) \wedge c \in col\} \\ \mathrm{post}_{pacnet}((u,t,c)) &= \bigcup\{\mathrm{post}_{acnet}(u,p) \times \{p\} \times \{c\} \mid p \in \mathrm{post}_{pacnet}(t) \wedge c \in col\}\ .\end{aligned}$$

That is, $\mathrm{pre}_{pacnet}((u,t,c))$ and $\mathrm{post}_{pacnet}((u,t,c))$ determine the tokens which transition t in mode u consumes and produces when fired.

Example 21. The PA-net on the right in the Figure 4.8 is supposed to be equivalent to two copies of the acyclic net on the left as indicated by col={1,2}. Therefore, there are two pairs of initial tokens (in total 4 tokens), one corresponding to mode 1 and the other to mode 2. That is, for the PA-net in Figure 4.8 we have, for example, $\mathrm{pre}_{pacnet}((a,t,1)) = \{(p_1,r_1,1)\}$ and $\mathrm{post}_{pacnet}((b,t,2)) = \{(p_4,r_2,2)\}$. ◇

4.6.2 Behaviour of PA-nets

In this section we will introduce notions related to the behaviour of parametrised acyclic net.

Definition 4.6.2 (marking and step of PA-net) *Let acnet be a* PA*-net.*

1. *The* markings *of pacnet, denoted by* markings(*pacnet*)*, are the subsets of* P_{pacnet}*.*
2. *The default* initial *marking is:*

$$M^{init}_{pacnet} = \{(p, p^{init}, c) \mid p \in M^{init}_{acnet} \wedge c \in col\}. \tag{4.1}$$

3. *The* steps *of pacnet, denoted by* steps(*pacnet*)*, are:*

$$\mathrm{steps}(pacnet) = \{U \in \mathbb{P}(T_{pacnet}) \mid \forall t \neq u \in U : \mathrm{pre}_{pacnet}(t) \cap \mathrm{pre}_{pacnet}(u) = \varnothing\}\ .$$

4. Given a marking M and $r \in P$, we denote

$$M(r) = \{(p,c) \mid (p,r,c) \in M\} = \{(p_1,c_1),\ldots,(p_k,c_k)\},$$

and in the diagrams place $p_1{:}c_1,\ldots,p_k{:}c_k$ inside p. ◇

That is, $p_1{:}c_1,\ldots,p_k{:}c_k$ are coloured tokens present in p at marking M. Moreover, each place p of *pacnet* is annotated by $p{:}\iota(p)$, and each transition t by $\iota(t)$. The same convention is used for the buffer places and CSA-nets later on. We also say that M is *colour-safe* if $M(r) \cap M(s) = \varnothing$, for all $r \neq s \in P$. Note that M^{init}_{pacnet} is colour-safe.

Example 22. The initial marking of the PA-net in Figure 4.8 is given by $M^{init}_{pacnet} = \{(p_1,r_1,1),(p_1,r_1,2),(p_3,r_1,1),(p_3,r_1,2)\}$. ◇

Definition 4.6.3 (enabled and executed step of PA-net) *Let M be a marking of a* PA*-net pacnet. A step U of pacnet is* enabled *at marking M if* $\mathrm{pre}_{pacnet}(U) \subseteq M$. *It can then be* executed *and yield*

$$M' = (M \cup \mathrm{post}_{pacnet}(U)) \setminus \mathrm{pre}_{pacnet}(U).$$

This is denoted by $M[U\rangle_{pacnet}M'$. ◇

Enabling a step in a PA-net is similar to that in an acyclic nets, where the global state requires all its pre-places to be marked. The execution of such a step first adds tokens to all its post-places and then removes tokens from all its pre-places.

Definition 4.6.4 (mixed step sequence of PA-net) *Let $(M^{init}_{pacnet} =)M_0,M_1\ldots,M_k$ $(k \geq 0)$ be markings and $U_1,\ldots,U_k$ be steps of pacnet such that we have $M_{i-1}[U_i\rangle_{pacnet}M_i$, for every $1 \leq i \leq k$. Then*

$$\mu = M_0U_1M_1\ldots M_{k-1}U_kM_k$$

is a mixed step sequence *of pacnet. This is denoted by $\mu \in \mathrm{mixsseq}(pacnet)$. Moreover, μ is* well-formed *if the following hold:*

- $\mathrm{post}_{pacnet}(t) \cap \mathrm{post}_{pacnet}(u) = \varnothing$, *for every $1 \leq i \leq k$ and all $t \neq u \in U_i$.*

- $\mathrm{post}_{pacnet}(U_i) \cap \mathrm{post}_{pacnet}(U_j) = \varnothing$, *for all* $1 \leq i < j \leq k$.

Note that in such a case M_i *is colour-safe, for every* $0 \leq i \leq k$. ◇

Example 23. Two of its mixed step sequences in Figure 4.8 are:

$$\begin{aligned}\mu_1 &= M^{init}_{pacnet} \quad \{(a,t,2)\} \quad M_1\ \{(b,t,2),(b,t,1)\}\ M_2\ \{(a,t,1)\}\ M_3\\ \mu_2 &= M^{init}_{pacnet}\ \{(b,t,1),(a,t,2)\}\ M_4\ \{(a,t,1),(b,t,2)\}\ M_3,\end{aligned}$$

where:

$$\begin{aligned}M_1 &= \{(p_1,r_1,1),(p_2,r_2,2),(p_3,r_1,1),(p_3,r_1,2)\}\\ M_2 &= \{(p_1,r_1,1),(p_2,r_2,2),(p_4,r_2,1),(p_4,r_2,2)\}\\ M_3 &= \{(p_2,r_2,1),(p_2,r_2,2),(p_4,r_2,1),(p_4,r_2,2)\}\\ M_4 &= \{(p_1,r_1,1),(p_2,r_2,2),(p_4,r_2,1),(p_3,r_1,2)\}.\end{aligned}$$

◇

To express full consistency between the behaviour of *pacnet* and that of the underlying acyclic net *acnet*, we need a notion of projection of the steps and markings of *pacnet* onto the steps and markings of *acnet*.

For every subset X of $P_{pacnet} \cup T_{pacnet}$ and $c \in col$, $X{\downarrow}c = \{x \mid (x,y,c) \in X\}$. Moreover, for every sequence $\xi = X_1 \ldots X_k$ of subsets of $P_{pacnet} \cup T_{pacnet}$, we denote $\xi{\downarrow}c = X_1{\downarrow}c \ldots X_k{\downarrow}c$.

Example 24. Let μ_1 be as in Example 23. Then

$$\begin{aligned}\mu_1{\downarrow}1 &= \{p_1,p_3\}\ \varnothing\ \{p_1,p_3\}\ \{b\}\ \{p_1,p_4\}\ \{a\}\ \{p_2,p_4\}\\ \mu_1{\downarrow}2 &= \{p_1,p_3\}\ \{a\}\ \{p_2,p_3\}\ \{b\}\ \{p_2,p_4\}\ \varnothing.\end{aligned}$$

Note that both $\mu_1{\downarrow}1$ and $\mu_1{\downarrow}2$ are mixed step sequences of the original acyclic net. ◇

Proposition 4.6.1 *Let pacnet be as in Definition 4.6.1.*

1. *If* $\mu \in \mathrm{mixsseq}(pacnet)$ *then* μ *is well-formed and* $\mu{\downarrow}c \in \mathrm{mixsseq}(acnet)$*, for every* $c \in col$.

2. *If* $\mu^c \in \text{mixsseq}(acnet)$*, for every* $c \in col$*, are sequences of the same length, then there is* $\mu \in \text{mixsseq}(pacnet)$ *such that* $\mu\downarrow c = \mu^c$*, for every* $c \in col$*.*

Proof 4.6.1 *(1) Let* $\mu = M_0U_1M_1\ldots M_{k-1}U_kM_k$ *and* $c \in col$*. The result is proven by induction on* k*.*

If $k = 0$ *then* $\mu = M_0 = M^{init}_{pacnet}$*. Then* μ *is well-formed and* $\mu\downarrow c \in \text{mixsseq}(acnet)$ *follow from Definition 4.6.1(5) and Eq. (4.1).*

The inductive case follows from the well-formedness of acnet, Definition 4.6.1(3), and the following fact: for every $(u,t,c) \in T_{pacnet}$*,*

$$\text{pre}_{pacnet}((u,t,c))\downarrow c = \text{pre}_{acnet}(u) \;\; \textit{and} \;\; \text{post}_{pacnet}((u,t,c))\downarrow c = \text{post}_{acnet}(u).$$

Also, it is important that the tokens and modes of transition firings based on different colours do not 'interfere' with each other in pacnet.

(2) Let n *be the common length of the mixed step sequences* μ^c *(for every* $c \in col$*).*

Let $c \in col$*. We observe that there is a mixed step sequence* $\nu^c = Z^c_1\ldots Z^c_n \in \text{mixsseq}(pacnet)$*, which only uses transitions and tokens based on the parameter c, such that* $\nu^c\downarrow c = \mu^c$*. (Note that Definition 4.6.1(3) ensures that* $u \in T'$ *can be 'executed' with parameter c when M is a colour-safe marking of pacnet such that* $\text{pre}_{acnet}(u) \subseteq M\downarrow c$*.)*

Let $\mu = Z_1\ldots Z_n$*, where* $Z_i = \bigcup_{c\in col} Z^c_i$*. Again, since the tokens and modes of transition firings based on different colours do not 'interfere' with each other in pacnet, we obtain* $\mu \in \text{mixsseq}(pacnet)$*. Moreover, by construction,* $\mu\downarrow c = \mu^c$*, for every* $c \in col$*.* ◇

The assumption that the mixed step sequences μ^c (for every $c \in col$) have the same length can be easily satisfied by adding extra empty steps to the 'shorter' ones.

Proposition 4.6.1(1) means that the behaviour of *pacnet* is well-formed and based on the behaviour of *acnet*. Conversely, Proposition 4.6.1(2) means that all the behaviours of *acnet* are represented by the behaviours of *pacnet*.

4.6.3 Parametrised communication structured acyclic nets

We now introduce a parametrised version of communication structured acyclic nets. Since the definitions closely follow those in the previous section, we keep explanations short.

Definition 4.6.5 (parametrised communication structured acyclic net) *A* parametrised communication structured acyclic net *(*PCSA*-net) is a tuple*

$$pcsan = (pacnet_1, \ldots, pacnet_n, Q, W, Q', W', \iota', col) \quad (n \geq 1)$$

such that:

- $pacnet_i = (P_i, T_i, F_i, acnet_i, \iota_i, col)$*, for* $1 \leq i \leq n$*, are* PA*-nets.*
- $csan = (acnet_1, \ldots, acnet_n, Q, W)$ *is a well-formed* CSA*-net.*
- Q' *is a set of* (master) buffer *places, and* $W' \subseteq Q' \times T \cup T \times Q'$*, where* $T = T_1 \cup \cdots \cup T_n$*, is a set of arcs adjacent to these places.*
- $\iota' : Q' \to \mathbb{P}(Q) \setminus \{\varnothing\}$ *is an annotation mapping such that* $Q = \biguplus\{\iota(q) \mid q \in Q'\}$*. For every* $q \in Q$*,* q_{pcsan} *will denote the unique buffer place in* Q' *such that* $q \in \iota'(q_{pcsan})$*.*
- *col is a finite set of* colours *(or* parameters*).*

We also assume that the sets of nodes of $pacnet_1, \ldots, pacnet_n, Q, Q'$ *are pairwise disjoint and*

$$W' = \{(x,y) \in Q' \times T \cup T \times Q' \mid (\iota(x) \times \iota(y)) \cap W \neq \varnothing\},$$

where $\iota = \iota_1 \cup \cdots \cup \iota_n \cup \iota'$*.* ◇

The idea of Definition 4.6.5 is similar to that of PA-net, and so the following notations are close to those introduced before :

$$T_{pcsan} = T_{pacnet_1} \cup \cdots \cup T_{pacnet_1} \;\; and \;\; P_{pcsan} = P_{pacnet_1} \cup \cdots \cup P_{pacnet_1} \cup \widetilde{Q},$$

where $\widetilde{Q} = \{(q, q_{pcsan}, c) \mid q \in Q \wedge c \in col\}$. Moreover, we denote, for all $1 \leq i \leq n$ and $(u,t,c) \in T_{pacnet_i}$:

$$\begin{array}{rcl} \mathrm{pre}_{pcsan}((u,t,c)) & = & \mathrm{pre}_{pacnet_i}((u,t,c)) \ \cup \{(q, q_{pcsan}, c) \in \widetilde{Q} \mid qWu\} \\ \mathrm{post}_{pcsan}((u,t,c)) & = & \mathrm{post}_{pacnet_i}((u,t,c)) \cup \{(q, q_{pcsan}, c) \in \widetilde{Q} \mid uWq\}. \end{array}$$

Example 25. Figure 4.9 shows an example of PCSA-net. ◇

We also denote $P = P_1 \cup \cdots \cup P_n \cup Q'$ and $F = F_1 \cup \cdots \cup F_n \cup W'$, and, for all $x \in P \cup T$, $p \in P$, and $u \in T_{csan}$, we denote:

$$\begin{array}{rclcrcl} \mathrm{pre}_{pcsan}(x) & = & \{z \mid zFx\} & \quad & \mathrm{pre}_{pcsan}(u,p) & = & \iota(p) \cap \mathrm{pre}_{csan}(u) \\ \mathrm{post}_{pcsan}(x) & = & \{z \mid xFz\} & & \mathrm{post}_{pcsan}(u,p) & = & \iota(p) \cap \mathrm{post}_{csan}(u)\ . \end{array}$$

Proposition 4.6.2 *For all $t \in T_{pcsan}$ and $u \in \iota(t)$:*

$$\begin{array}{rcl} \mathrm{pre}_{csan}(u) & = & \uplus\{\mathrm{pre}_{pcsan}(u,p) \mid p \in \mathrm{pre}_{pcsan}(t)\} \\ \mathrm{post}_{csan}(u) & = & \uplus\{\mathrm{post}_{pcsan}(u,p) \mid p \in \mathrm{post}_{pcsan}(t)\}\ . \end{array}$$

Example 26. Figure 4.9(a) shows a CSA-net where two acyclic nets (the upper one exhibiting concurrency and the lower exhibiting alternative) communicate through two buffer places. Figure 4.9(b) shows a corresponding representation after parametrising. Note that $\iota(r_1) = \{p_1\}$ and $\iota(t_1) = \{a\}$ for the upper acyclic net, while $\iota(r_7) = \{p_7\}$ and $\iota(t_5) = \{e, f\}$ for the lower acyclic net. Intuitively, $\iota(t_5)$ specifies different 'modes' of the execution of transition t_5 for each mode $u \in \iota(t_5)$, the arcs incoming to transition $\{t_5\}$ 'deliver' the input places of u, and similarly for the output arc. ◇

4.6.4 Behavioural properties of PCSA-nets

In this section, we will introduce the behaviors of PCSA-nets. Generally, these are similar to those in Chapter 3, so we will keep this discussion brief.

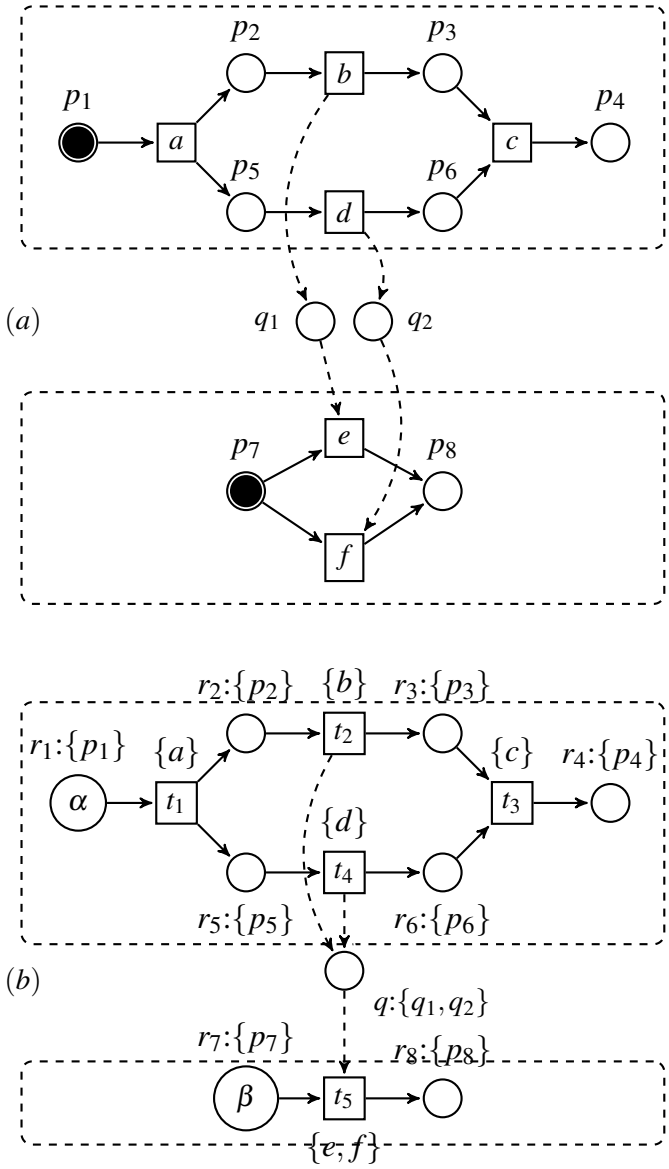

Fig. 4.9 A CSA-net (a); and its PCSA-net version (resulting introducing an MBP and folding the lower acyclic net) with $col = \{1,2\}$, where $\alpha = \{p_1{:}1, p_1{:}2\}$ and $\beta = \{p_7{:}1, p_7{:}2\}$ (b).

Definition 4.6.6 (marking and step of PCSA-net) *Let pcsan be as in Definition 4.6.5.*

1. *The* markings *of pcsan, denoted by* markings$(pcsan)$*, are the subsets of P_{pcsan} and Q_{pcsan}.*

2. *The default* initial *marking of pcsan is $M^{init}_{pcsan} = M^{init}_{pacnet_1} \cup \cdots \cup M^{init}_{pacnet_n}$.*

3. *The steps of pcsan are*

$$\text{steps}(pcsan) = \{U \in \mathbb{P}(T_{pcsan}) \mid \forall t \neq u \in U : \text{pre}_{pcsan}(t) \cap \text{pre}_{pcsan}(u) = \varnothing\}\ .$$

4. *A step U of pcsan is* enabled *at marking M if*

$$\text{pre}_{pcsan}(U) \subseteq M \cup (\text{post}_{pcsan}(U) \cap \widetilde{Q}),$$

where $\widetilde{Q} = \{(q, q_{pcsan}, c) \mid q \in Q \wedge c \in col\}$. It can then be executed *and yield*

$$M' = (M \cup \text{post}_{pcsan}(U)) \setminus \text{pre}_{pcsan}(U).$$

This is denoted by $M[U\rangle_{pcsan} M'$. ◇

Definition 4.6.7 (mixed step sequence of PCSA-net) *Let pcsan be as in Definition 4.6.5. Moreover, let $M_0, M_1 \ldots, M_k$ ($k \geq 0$) be markings and $U_1, \ldots, U_k$ be steps of pcsan such that $M_0 = M^{init}_{pcsan}$ and we have $M_{i-1}[U_i\rangle_{pcsan} M_i$, for every $1 \leq i \leq k$. Then*

$$\mu = M_0 U_1 M_1 \ldots M_{k-1} U_k M_k$$

is a mixed step sequence *of pcsan. This is denoted by $\mu \in \text{mixsseq}(pcsan)$. Also, μ is* well-formed *if the following hold:*

- $\text{post}_{pcsan}(t) \cap \text{post}_{pcsan}(u) = \varnothing$, *for every $1 \leq i \leq k$ and all $t \neq u \in U_i$.*
- $\text{post}_{pcsan}(U_i) \cap \text{post}_{pcsan}(U_j) = \varnothing$, *for all $1 \leq i < j \leq k$.*

Note that in such a case M_i is colour-safe, for every $0 \leq i \leq k$. ◇

We also use the same notion of projection as for PA-nets, and then obtain a result showing full consistency between the behaviour of a PCSA-net and its underlying CSA-net.

Proposition 4.6.3 *Let pcsan be as in Definition 4.6.5.*

1. *If* $\mu \in \text{mixsseq}(pcsan)$ *then* μ *is well-formed and* $\mu{\downarrow}c \in \text{mixsseq}(csan)$*, for every* $c \in col$*.*

2. *If* $\mu^c \in \text{mixsseq}(csan)$*, for every* $c \in col$*, are sequences of the same length, then there is* $\mu \in \text{mixsseq}(pcsan)$ *such that* $\mu{\downarrow}c = \mu^c$*, for every* $c \in col$*.*

Proof 4.6.2 *The proof is similar to that of Proposition 4.6.1.*

4.7 Conclusion

This chapter introduced two extensions of CSA-nets using the concepts of folding and parametrisation. The analysis and visualisation of CSA-nets is a challenging task, especially for large and complex systems. The proposed parametrisation technique can contribute to improving visualisation and increasing the ability to understand models under investigation. Future work will consider other forms of folding, e.g., merging together entire components of acyclic nets and subsequently the extension to behavioural structured acyclic nets.

Chapter 5

Improving Placement of CSA-nets

Communication Structured Acyclic Nets (CSA-nets) are a Petri net-based formalism for representing interactions between subsystems of complex evolving systems (CES). CSA-nets, composed of sets of acyclic nets, are suitable for modelling and visualising the behaviour of event-based systems. Each subsystem is represented using a separate acyclic net and is linked with other acyclic nets through a set of buffer places. These buffer places connect different acyclic nets and depict their interactions. The challenge arises as the growth of subsystems increases the number of arc crossings. This chapter focuses on improving the visualisation of CSA-nets by rearranging their component acyclic nets to minimise the number of crossing arcs. Specifically, this chapter takes inspiration from the main ideas behind three well-known sorting algorithms, viz., bubble sort, insertion sort, and selection sort, and integrates them into formulas to compute crossings. Experiments are carried out to compare the improvements resulting from following each of the three approaches. The outcomes point to the high effectiveness of the selection sort-inspired approach.

5.1 Introduction

The use of CSA-nets for example, in criminal investigations, can encounter significant hurdles due to the complexity and sheer volume of the information involved. These obstacles pose considerable challenges for investigators attempting to interpret CESs and making accurate

and timely decisions [24]. This becomes particularly critical as the volume of data increases because the complexity of the visualisation can significantly hinder effective analysis and decision-making. Specifically, an increasing number of subsystems that communicate with each other leads to an increase in the number of buffer places, resulting in a higher number of crossings in the model. However, minimising the crossing number is an NP-hard problem [63, 45]. To alleviate these issues, enhancing the visualisation of CSA-nets representations could play a crucial role, enabling much better understanding of the problem under investigation. Such an improvement would assist investigators in identifying, e.g., causality among the events, thereby aiding system analysis.

Reducing the crossing number for CSA-nets resembles a sorting problem where the goal is to find the most efficient arrangement of acyclic nets to reduce the total count of crossings. In other words, by sorting acyclic nets in a manner that minimises the number of crossing arcs or edges in the visualisation, the model's representation can be improved. One of the efficient approaches to enhance the clarity and readability of such representations is to employ sorting algorithms to arrange the acyclic nets. By doing so, we aim to make the graphical representation more comprehensible.

Sorting algorithms [72] are designed to reorder items in a list or array, either in increasing or decreasing sequence [19]. They are key to numerous computational activities, ranging from displaying data in a form understandable to humans to enhancing the efficiency of intricate algorithms. In the realm of layered graphs [28], also referred to as layered digraphs or hierarchies, sorting algorithms hold substantial significance. They contribute to various aspects of layered graph visualisation and are instrumental to their success. There are several sorting algorithms such as bubble sort, insertion sort, selection sort, quick sort, merge sort and heap sort [86]. Each of these algorithms has its strengths and weaknesses and is best suited to particular kinds of tasks and data types. The right algorithm to use depends on the specifics of the problem at hand, including the size of the list, the range and type of values, and the desired time and space complexity [80].

In this thesis, we investigate three elementary sorting algorithms, namely Bubble Sort, Insertion Sort, and Selection Sort. These sorting techniques, while simple, can be quite

effective in properly arranging the acyclic nets and ultimately enhancing the visualisation of CSA-nets. Given these algorithms' characteristics, they can be employed strategically depending on the specific requirements of the CSA-nets under consideration, such as the size of the acyclic nets and the degree of 'unsortedness'. The overall aim is to optimise the visual representation, reduce crossing arcs, and thus improve readability and analysis. This strategic utilisation of sorting algorithms in enhancing visualisation provides a significant step forward in the field, contributing to more effective cybercrime investigation [33], AI interpretation, clinical investigation, and other applications of CES.

This chapter proposes an innovative approach to enhance the visualisation of CSA-nets by reorganising the acyclic nets with the objective of minimising the crossing number.

5.2 Related work

The techniques of data visualisation, like directed graphs [31, 20], are essential for various applications, such as financial fraud detection, and the analysis of social and biological networks. This is due to the merits of such techniques for analysing complex data, identifying patterns, and extracting valuable insights. The main components of such techniques are the nodes and edges/arrows [23].

The effectiveness of visual representations is paramount when dealing with complex systems, where nodes and edges serve as crucial elements in conveying relationships and structures [34]. However, this results in an high number of edges that may present a common hindrance in visualisations due to their intersection. Such a problem is commonly known as *crossing arcs*. This problem is widely agreed to be an inhibitor in understanding graphs as it profoundly affects the clarity of visual representations and hinder the understandability of visualisations [100]. Moreover, the characteristics of complex systems usually make it very difficult to yield drawings free of crossings [93]. Consequently, the topic of reducing arc crossings has been a focal point for researchers with the aim of minimising the total number of arc crossings [32], and several strategies have been devised for this purpose. For example, Jianu et al. [61] propose to employ colouring schema for the crossing arcs. In particular,

they colour the intersecting edges with contrasting colours to maximise the viewer's ability to distinguish them. Another issue that arises from crossing arcs, especially with directed graphs, is when the arrowheads crosses the edges. This commonly occurs when several edges are incident to a common node. Then, it becomes hard to see whether a specific edge is incoming or outgoing to a specific node. Binucci et al. [23] proposed placing the arrowhead at different points along the arc. Specifically, they introduced a solution using integer linear programming that employs a heuristic approach to determine whether to place the arrowhead along the entire edge lines or at a specific point (for example, the midpoint) of these arcs. More recently, Francisco et al. [81] concentrated on radial layered graph which is a subclass of graphs where the nodes are arranged in a circular shape around a predefined node referred to as the root. Each successive circle is represented by a larger circle radiating outward. The nodes in each circle are of the same distance from the root node. This type of graph is of major importance in genomics visualisation. Similar to a directed graph, crossing edges limits the applicability of a radial layered graph. Therefore, they developed an algorithm to minimise edge crossings in radial layered graphs that do not require adding or removing nodes and edges. Using a two-step process of counterclockwise circle rotation and edge-to-segment path conversion, the algorithm optimises node placements within their circular layers.

A different attempt was carried out by Domrös et al. [38]. This includes using the dummy vertices technique between layered crossings which involves introducing a new path between two crossing layers. For example, a crossing arc between layer A and layer C through layer B can be replaced by new paths, one between layer A and layer B and the other between B and C. However, this technique will decrease the readability of the graph and might make the graph hard to follow because the number of vertices will increase.

Focusing on one emerging technology, the big graphs of NoSQL, which focus on data relationships rather than data values. The issue is that arrows cross with each other while trying to elaborate on the relationships in the data. Accordingly, Adoni et al. [1] proposed an efficient parallel and distributed algorithm for large-scale graph partitioning called Distributed Placement of Hub-Vertices (DPHV) to reduce the complexity of queries by dividing the

graph into multiple sub-graphs. However, it is an NP-complete problem and presents numerous challenges. Walter et al. [127] proposed a practical method for drawing cable plans of complex, custom-configured machines. They introduced 'port groups' to model plug sockets, allowing ports within a group to change their position for improved aesthetics, while maintaining contiguity. The method was tested on both real-world and synthetic data simulating real-world scenarios. Their experiment was compared with Kieler, a tool used to model and analyse complex systems. The approach resulted in 10-30% fewer crossings while performing similarly or slightly worse than Kieler in terms of bend count and computation time.

One popular technique in this area is Sugiyama-style [118], otherwise known as the layered or hierarchical graph drawing. It is a specific method employed for the visualisation of directed graphs which arranges the vertices of the graph across distinct horizontal tiers or layers, predominantly orienting the edges in a downward direction. The paradigmatic design of a layered drawing aspires to be an upward planar drawing, wherein all the edges align in a uniform direction, and no edge intersects another.

Going to modelling tools, CSA-net can be viewed as a directed graph that depicts the relations between complex system components. With no exceptions, CSA-net suffers from the crossing arc issue. The hurdle is that, to the best of our knowledge, no attempts have been carried out to reduce the crossing in the field of CSA-net. Accordingly, this chapter focuses on improving the placement of CSA-net components using sorting algorithm to mitigate this issue.

5.3 Placement of CSA-nets

CSA-nets are represented by subsystems that facilitate communication between them. While the component acyclic nets yield relatively simple graphs that are typically (very) long and have few crossings, the primary challenge arises from the arcs that represent communication between subsystems (acyclic nets). These arcs can cross several acyclic nets, thereby diminishing the comprehension of the overall graphical display. A potential solution is to

reorder the placement of the acyclic nets in a manner that minimises the total number of crossings. From this perspective, minimising the crossing number for CSA-nets can be seen as a kind of sorting problem, where the goal is to find the most effective placement of acyclic nets. In this chapter, we investigate how the principles behind basic sorting algorithms might aid in developing effective placement strategies for the component acyclic nets in a CSA-net. Specifically, this work aims to examine strategies to reduce the number of crossings between acyclic nets and arcs representing communication in a CSA-net by optimising the placement of the acyclic nets.

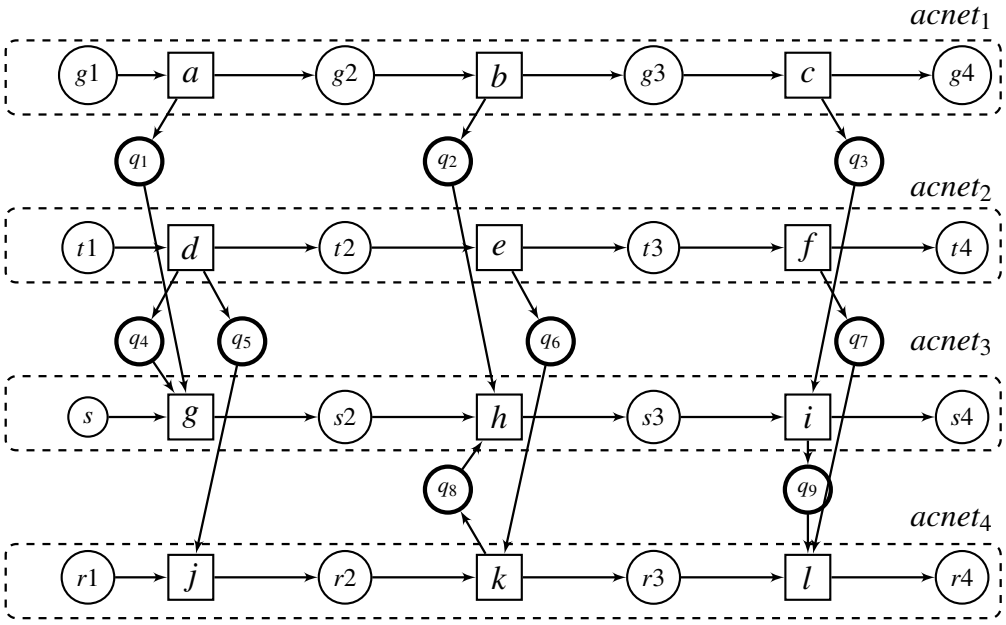

Fig. 5.1 A graphical representation of a CSA-net with 4 acyclic nets causing 6 crossings.

Throughout this chapter, we assume that we are given a CSA-net

$$csan = (acnet_1, \ldots, acnet_n, Q, W)$$

as in Definition 3.5.1. Moreover we assume that:

- $n \geq 3$ since only then it makes sense to consider different placements of the component acyclic nets.

- For each buffer place $q \in Q$ there is exactly one transition t and exactly one transition u such that tWq and qWu.

- For all $1 \leq i, j \leq n$ we denote by $m_{i,j}$ the number of buffer places q such that there are $t \in T_{acnet_i}$ and $u \in T_{acnet_j}$ satisfying tWq and qWu. To simplify some of the formulas, we will also use the notation $c_{i,j} = m_{i,j} + m_{j,i}$, for all $1 \leq i, j \leq n$. Note that $c_{i,j} = c_{j,i}$, for all $1 \leq i, j \leq n$.

Additionally, we assume that *csan* is represented graphically by placing the acyclic nets one above the other (that is, in the form of a stack). Initially, the placement is $acnet_1, \ldots, acnet_n$. In general, a placement can be represented by an integer array H with n entries indexed $0, 1, \ldots, n-1$. It represents the placement $acnet_{H(0)}, \ldots, acnet_{H(n-1)}$ of the acyclic nets which are the components of the CSA-nets *csan*, and so it is a permutation of the integers $1, 2, \ldots, n$. Initially, $H_{initial} = (1, 2, \ldots, n)$.

CSA-net consists of a number of disjoint acyclic nets that communicate through special buffer places. These buffer places can generate many arcs 'crossing' other acyclic nets, which makes the model hard to visualise and analyse. Therefore, our idea is to develop algorithms which will serve the placement purpose efficiently and accurately.

Consider, for example, a graphical representation of CSA-net in Figure 5.1, where there are 6 crossings generated by buffer-related arcs passing through acyclic nets. For example, the communication between $acnet_1$ and $acnet_3$ generates three crossings. The placement of acyclic nets shown in Figure 5.1 could be improved, for example, by Figure 5.4. The result would generate only one crossing.

In the initial placement, for all buffer places $q \in Q$ and transitions $t \in T_{acnet_i}$ and $u \in T_{acnet_j}$ such that tWq and qWu, the arcs adjacent to q pass through exactly $|i-j|-1$ acyclic nets placed in-between $acnet_i$ and $acnet_j$. We refer to each such passing as a 'crossing'.[1] That is, initially, when we place the acyclic nets according to $H_{initial} = (1, 2, \ldots, n)$, the number of

[1] Note that we assume that each buffer connection between $acnet_i$ and $acnet_j$ generates one crossing with any acyclic net in-between $acnet_i$ and $acnet_j$.

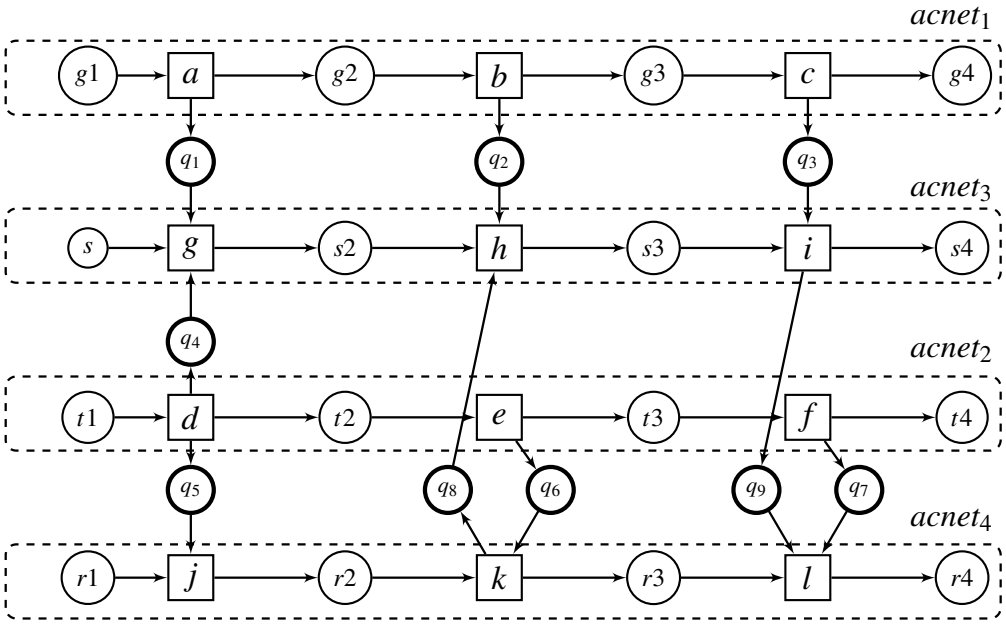

Fig. 5.2 A graphical representation of a CSA-net with 4 acyclic nets after applying bubble placement to the CSA-net from Figure 5.1 causing 2 crossings.

crossings is:

$$\begin{aligned} cross(H_{initial}) &= \sum_{i\neq j}(|i-j|-1)\cdot m_{i,j} \\ &= \sum_{i<j}(|i-j|-1)\cdot c_{i,j} \end{aligned} \tag{5.1}$$

In general, for a placement given by permutation H, we have:

$$\begin{aligned} cross(H) &= \sum_{i\neq j}(|i-j|-1)\cdot m_{H(i),H(j)} \\ &= \sum_{i<j}(|i-j|-1)\cdot c_{H(i),H(j)} \end{aligned} \tag{5.2}$$

Eq.(5.1) describes how to calculate the number of crossings in the initial placement. Eq.(5.2) describes how to calculate the number of crossings in the placement given by an arbitrary placement represented by H.

The aim of our work is to place the acyclic nets so that the number of these crossings is minimised. In other words, to find H such that $cross(H)$ is as small as possible. In this chapter, we investigate different algorithms inspired by three well-known sorting strategies, namely, bubble sort, insertion sort, and selection sort. The reason behind choosing these

algorithms is due to their simplicity and their reasonable running time compared to others. Moreover, they are known to be consistent sorting algorithms requiring constant space.

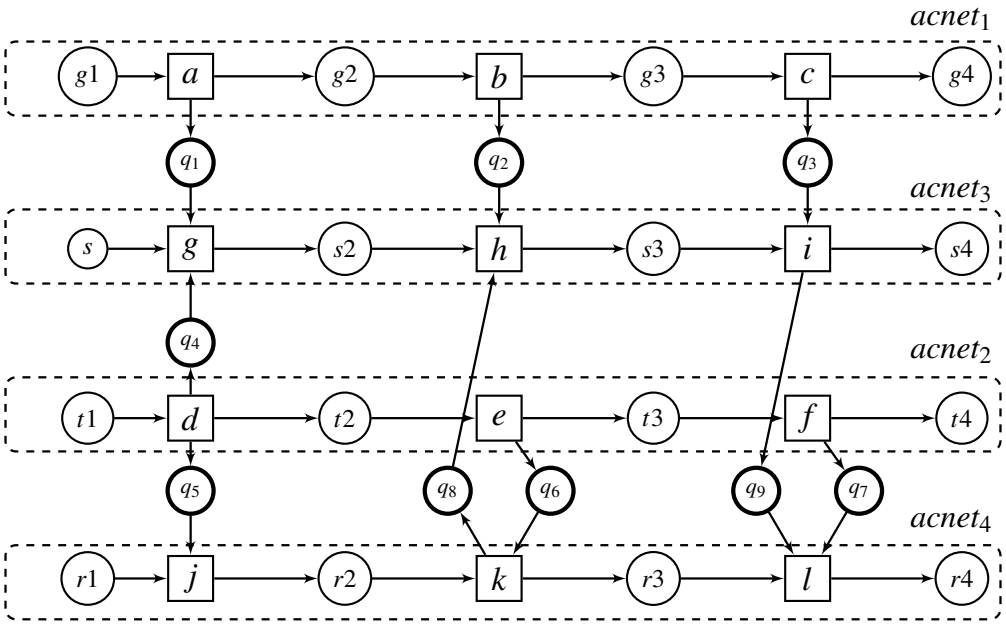

Fig. 5.3 A graphical representation of a CSA-net with 4 acyclic nets after applying insertion placement to the CSA-net from Figure 5.1 causing 2 crossings.

5.4 Bubble-sorted placement

Bubble sort [16] is a sorting algorithm that iterates over a list of n items while comparing each neighboring pair in the list. The strategy is to swap the pair's members if they are not in the correct order. The swapping is repeated in up to $n-1$ passes of the list [123]. While not suitable for large data sets due to its relatively high time complexity [42], it proves to be quite useful for smaller or nearly sorted lists, making it a good fit for CSA-nets with fewer acyclic nets or when the nets are nearly in the desired order. We apply this concept by carrying out the comparison step, taking into account the number of crossings. More precisely, we examine $acnet_{H(i)}$ and $acnet_{H(i+1)}$, and swap them if this reduces the total number of crossings.

Proposition 5.4.1 (difference crossing) *Let H be a placement, $0 \leq i < n-1$, and H' be the placement given, for every $0 \leq j \leq n-1$, by:*

$$H'(j) = \begin{cases} H(i+1) & \text{if } j = i \\ H(i) & \text{if } j = i+1 \\ H(j) & \text{otherwise} \end{cases}$$

Then $diffcross = cross(H) - cross(H')$ can be computed by:

$$diffcross = \sum_{j=0}^{i-1} (c_{H(i),H(j)} - c_{H(i+1),H(j)}) + \sum_{j=i+2}^{n-1} (c_{H(i+1),H(j)} - c_{H(i),H(j)}). \tag{5.3}$$

where:

- *If $i = 0$ then $\sum_{j=0}^{i-1}(c_{H(i),H(j)} - c_{H(i+1),H(j)})$ is omitted.*
- *If $i = n-2$ then $\sum_{j=i+2}^{n-1}(c_{H(i+1),H(j)} - c_{H(i),H(j)})$ is omitted.*

Proof 5.4.1 *Swapping $acnet_{H(i)}$ with $acnet_{H(i+1)}$ does not affect the numbers of crossings induced by buffers between the acyclic nets $acnet_{H(0)}, \ldots, acnet_{H(i-1)}, acnet_{H(i+2)}, \ldots, acnet_{H(n-1)}$. Moreover, we have that:*

- *each buffer place (in either direction) between $acnet_{H(0)}, \ldots, acnet_{H(i-1)}$ and $acnet_{H(i)}$ induces one more crossing;*
- *each buffer place (in either direction) between $acnet_{H(0)}, \ldots, acnet_{H(i-1)}$ and $acnet_{H(i+1)}$ induces one less crossing;*
- *each buffer place (in either direction) between $acnet_{H(i+2)}, \ldots, acnet_{H(n-1)}$ and $acnet_{H(i)}$ induces one less crossing;*
- *each buffer place (in either direction) between $acnet_{H(i+2)}, \ldots, acnet_{H(n-1)}$ and $acnet_{H(i+1)}$ induces one more crossing;*
- *buffer places (in either direction) between $acnet_{H(i)}$ and $acnet_{H(i+1)}$ induce no crossings (before and after swapping).*

It then follows that

$$cross(H) - cross(H') = \sum_{j=0}^{i-1} c_{H(i),H(j)} - \sum_{j=0}^{i-1} c_{H(i+1),H(j)} - \sum_{j=i+2}^{n-1} c_{H(i),H(j)} + \sum_{j=i+2}^{n-1} c_{H(i+1),H(j)}.$$

That is, crossings between the acyclic nets outside the swapped pair, $acnet_{H(i)}$ and $acnet_{H(i+1)}$, remain unchanged by the swap. This implies that the swap does not change crossings between $acnet_{H(0)},\ldots,acnet_{H(i-1)},acnet_{H(i+2)},\ldots,acnet_{H(n-1)}$. Moreover, for any buffer place connecting the sequence before $acnet_{H(0)},\ldots,acnet_{H(i-1)}$ and $acnet_{H(i)}$ as well as between $acnet_{H(i+2)},\ldots,acnet_{H(n-1)}$ and $acnet_{H(i)}$, there is an increase in one crossing. This is because swapping changes their relative positions. Also, buffer places connecting the sequence before $acnet_{H(0)},\ldots,acnet_{H(i-1)}$ and $acnet_{H(i+1)}$ as well as $acnet_{H(i+2)},\ldots,acnet_{H(n-1)}$ and $acnet_{H(i)}$, experience one less crossing, due to the change of relative positions. Note also that the buffer places between $acnet_{H(i)}$ and $acnet_{H(i+1)}$ do not change the number of crossings as a result of the swap.
Hence Eq.(5.3) is correct. ◇

The *diffcross* in (5.3) has two summations that are used to compute whether $acnet_{H(i)}$ is to be swapped with $acnet_{H(i+1)}$. The interpretation of each summation is as follows.

1. The first summation, $\sum_{j=0}^{i-1}(c_{H(i),H(j)} - c_{H(i+1),H(j)})$, checks if the number of communication arrows from $H(i)$ to the preceding nets is more or less than those from $H(i+1)$. If the summation is positive, $H(i)$ has more connections to the preceding nets compared to $H(i+1)$. If the summation is negative, $H(i+1)$ has more connections to the preceding nets. This helps determine if swapping $H(i)$ with $H(i+1)$ might reduce the total number of crossings.

   Example 27. Consider that we are focusing on the fifth acyclic net, and the total numbers of arrows from the fifth and sixth acyclic net to the preceding four acyclic nets are 10 and 15, respectively. This means that the sixth acyclic net is more related to the first four acyclic nets and it would be better to swap it with the fifth acyclic net. ◇

2. The second summation, $\sum_{j=i+2}^{n-1}(c_{H(i+1),H(j)} - c_{H(i),H(j)})$, checks if the number of communication arrows from $H(i+1)$ to the succeeding nets is more or less than those from $H(i)$. If the summation is positive, $H(i+1)$ has more connections to the succeeding nets compared to $H(i)$. If the summation is negative, $H(i)$ has more connections to the succeeding nets. This helps determine if swapping $H(i)$ with $H(i+1)$ might reduce the total number of crossings.

   Example 28. Consider that we are focusing on the fifth acyclic net, and the total numbers of arrows from the fifth acyclic net and sixth acyclic net to the four last acyclic nets are 7 and 5, respectively. This means that the fifth acyclic net is more related to the last four acyclic nets and it would be better to swap it with the sixth one. ◇

That is, if we treat each summation as a condition, we can infer that if both conditions are satisfied, the final result of the summations will be a negative value (less than zero), confirming the validity of the swapping.

Algorithm 1: bubble placement (see Figure 5.2)

```
1  Function BubblePlace():
2      H = (1,2,...,n)
3      repeat
4          stop = true
5          for i = 0 to n-2 do
6              Compute the crossing difference as per Eq.(5.3)
7              if diffcross < 0 then
8                  temp = H(i)
9                  H(i) = H(i+1)
10                 H(i+1) = temp
11                 stop = false
12             end
13         end
14     until stop;
15     return H
```

Algorithm 1 provides details of the implementation. Note that the algorithm will always terminate. This is because each successful swap, indicated by *diffcross* value, reduces the total number of crossings, ensuring that the algorithm converges. Note that *diffcross* calculates

the 'effect' of swapping $acnet_{H(i)}$ with $acnet_{H(i+1)}$ which are currently placed next to each other. If $diffcross < 0$ the swapping reduces the number of crossings. All we need to do is calculate the difference between the total number of crossings now, and after the swap. Note that to determine the effect of swapping one does not need to calculate the total number of crossings which is quadratic (see Eq.(5.2)). The calculation given in Eq.(5.3) is linear, and so much more efficient.

Example 29. To illustrate Algorithm 1 with an example, consider the CSA-net shown in Figure 5.1, which initially has 6 crossings. The algorithm begins by iterating through the acyclic nets as per the loop in line 5. Initially, the index i is set to 0 (representing the first acyclic net), and when substituting into Eq.(5.3), the term $\sum_{j=0}^{i-1}(c_{H(i),H(j)} - c_{H(i+1),H(j)})$ is omitted because the summation cannot run from 0 to -1. However, the term $\sum_{j=i+2}^{n-1}(c_{H(i+1),H(j)} - c_{H(i),H(j)})$ is calculated, resulting in a crossing value of 1 for the first acyclic net. Consequently, the condition in line 7 is not met, indicating that no swapping will occur, and the first acyclic net will remain in its position. Moving on to the second acyclic net, the term $\sum_{j=0}^{i-1}(c_{H(i),H(j)} - c_{H(i+1),H(j)})$ is executed. Furthermore, the term $\sum_{j=i+2}^{n-1}(c_{H(i+1),H(j)} - c_{H(i),H(j)})$ is calculated, resulting in a crossing value of -2 for the second acyclic net. Accordingly, the condition in line 7 is met, leading to the swapping of the second acyclic net with the third acyclic net. This process is repeated until all the acyclic nets have been visited. The final configuration of the CSA-net after sorting using bubble sort is shown in Figure 5.2. By examining the figure, we can observe that bubble sort has reduced the number of crossings from the initial 6 to 2 in the model. ◇

5.5 Insertion-sorted placement

Insertion Sort [115] is another straightforward sorting technique. This algorithm sorts a list by shifting each element to its correct position in the sorted portion of the list, one item at a time [107]. Although it shares the same time complexity limitations as Bubble Sort for larger datasets, it performs efficiently on nearly sorted lists. Thus, it can be a suitable choice for sorting CSA-nets in contexts akin to those mentioned for Bubble Sort. In this method, we

continuously insert the first unsorted element into its optimal location and shift subsequent elements following this position to accommodate the newly inserted element. Similar to the bubble placement approach, the insertion placement also considers the number of crossings.

Proposition 5.5.1 ((cost of insertion)) *Let H be a placement, $0 \leq j \leq i < n-1$, and H' be the placement given, for every $0 \leq m \leq n-1$, by:*

$$H'(m) = \begin{cases} H(i) & \text{if } m = j \\ H(m-1) & \text{if } j < m < i \\ H(m) & \text{otherwise} \end{cases}$$

Then $cost_j$ defined as:

$$\sum_{k,l \leq i} (|k-l|-1) \cdot c_{H'(k),H'(l)} - \sum_{k,l \leq i-1} (|k-l|-1) \cdot c_{H(k),H(l)}$$

can be computed as:

$$cost_j = \sum_{k=0}^{j-1} \sum_{m=j}^{i-1} c_{H(k),H(m)} + \sum_{k=0}^{j-1} (|j-k|-1) \cdot c_{H(k),H(i)} + \sum_{m=j}^{i-1} |j-m| \cdot c_{H(i),H(m)}. \quad (5.4)$$

where:

- *If $j = 0$ the we omit $\sum_{k=0}^{j-1} \sum_{m=j}^{i-1} c_{H(k),H(m)}$ and $\sum_{k=0}^{j-1} (|j-k|-1) \cdot c_{H(k),H(i)}$.*
- *If $j = i$ then we omit $\sum_{k=0}^{j-1} \sum_{m=j}^{i-1} c_{H(k),H(m)}$ and $\sum_{m=j}^{i-1} |j-m| \cdot c_{H(i),H(m)}$.*

Proof 5.5.1 *Inserting $acnet_{H(i)}$ at position j means that each buffer place (in either direction) between $acnet_{H(0)}, \ldots, acnet_{H(j-1)}$ and $acnet_{H(j)}, \ldots, acnet_{H(i-1)}$ induces one more crossing. Hence (5.4) is correct.* ◇

The cost of insertion placement is an elaboration of the cost in Eq.(5.4) associated with an insertion operation. This equation is divided into separate summations, each capturing a specific aspect of the cost incurred. The double summation $\sum_{k=0}^{j-1} \sum_{m=j}^{i-1} c_{H(k),H(m)}$ calculates the cost between every pair where k is before the insertion point j and m is between the

insertion point j and the position of the inserted element (i_1). It essentially measures how the insertion affects the cost between elements that were before j and those that were between j and i. The second summation $\sum_{k=0}^{j-1}(|j-k|-1)\cdot c_{H(k),H(i)}$ calculates the cost between each element before the insertion point j and the element being inserted at i. Note that the term $|j-k|-1$ modifies the cost based on how far each element k is from the insertion point j. This reflects how the insertion changes the relative positions of these elements. The summation $\sum_{m=j}^{i-1}|j-m|\cdot c_{H(i),H(m)}$ computes the cost between the inserted element originally at position i and each element between the insertion point j, and $|j-m|$ is a modifier that accounts for the distance between the insertion point and each element m in this range.

Algorithm 2: insertion placement (see Figure 5.3)

```
1  Function InsertionPlace():
2      H = (1,2,...,n)
3      for i=1 to n - 1 do
4          bestcost = maxint
5          for j = 0 to i do
6              Compute cost_j as per Eq.(5.4)
7              if cost_j < bestcost then
8                  bestcost = cost_j
9                  position = i
10             end
11         end
12         for j = position to i-1 do
13             H(j+1) = H(j)
14         end
15         H(position) = i
16     end
17     return H
```

Algorithm 2 provides details of the implementation. In this case, before each iteration for i, we assume that H is such that $H(0),\ldots,H(i-1)$ already contain the 'best' placement for the acyclic nets $acnet_{H(0)},\ldots,acnet_{H(i-1)}$. The iteration step then finds the 'best' place for the acyclic net $acnet_{H(i)}$ with respect to $H(1),H(2),\ldots,H(i-1)$ (the relative ordering of $acnet_{H(0)},\ldots,acnet_{H(i-1)}$ is unchanged). Then $acnet_{H(i)}$ is inserted. Note that the acyclic

nets $acnet_{H(i+1)}, \ldots, acnet_{H(n-1)}$ are not considered when placing $acnet_{H(i)}$. Note also that if $position = i$ then the loop **for** $j = position$ **to** $i-1$ is not executed.

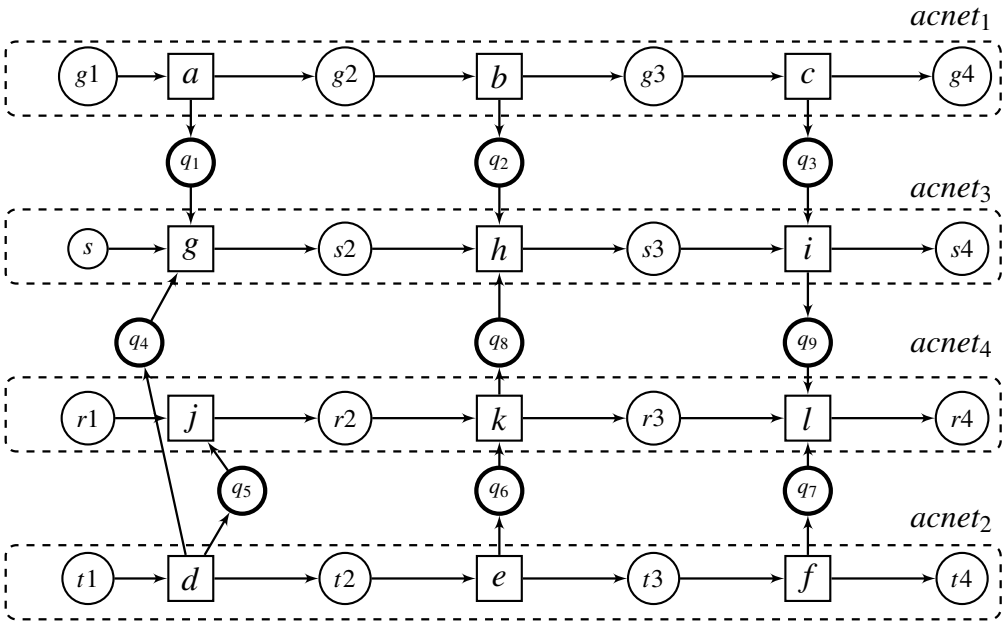

Fig. 5.4 A graphical representation of a CSA-net with 4 acyclic nets after applying selection placement to the CSA-net from Figure 5.1 causing only one crossings.

Example 30. An illustrative example of Algorithm 2 is provided using the CSA-net depicted in Figure 5.1, which initially contains 6 crossings. Similar to the bubble sort algorithm, the procedure begins by addressing the first acyclic net, as outlined in line 3. In line 4, a default high value, in this case, 1000, is assigned as the best cost. The reason is to guarantee that the conditional statement in line 7 is satisfied during every iteration, simplifying the determination of the optimal position for each acyclic net. The operational steps of the algorithm are as follows:

For all acyclic nets, excluding the first one, crossings are calculated as defined by Eq.(5.4). It is important to note that, in this context, the terms $\sum_{k=0}^{j-1}\sum_{m=j}^{i-1} c_{H(k),H(m)}$ and $\sum_{k=0}^{j-1}(|j-k|-1)\cdot c_{H(k),H(i)}$ are excluded. The reason for this exclusion is that it leads to a cost of 0 for the first acyclic net relative to the second acyclic net. Although this value is less than the predefined threshold of 1000, no swaps occur due to the unsatisfied condition in line 12,

which stipulates that i cannot be equal to j. This ensures that the first acyclic net remains in its original position in the sorted sequence.

Moving on to the second acyclic net, where $i = 2$, both terms $\sum_{k=0}^{j-1}\sum_{m=j}^{i-1} c_{H(k),H(m)}$ and $\sum_{k=0}^{j-1}(|j-k|-1)\cdot c_{H(k),H(i)}$ are considered. As a result, a cost of 0 is determined between the second and third acyclic nets. Since this value is below the 1000 threshold, a swap is required. Specifically, the positions of the second and third acyclic nets are exchanged. Continuing this methodology results in an insertion-sorted CSA-net, visually represented in Figure 5.3. A visual inspection of the figure confirms that the insertion sort algorithm successfully eliminates 4 of the initial 6 crossings present in the model. ◇

5.6 Selection-sorted placement

Selection sort maintains two lists, one contains the sorted values and the other contains the unsorted values [121]. Initially, the former is empty and in each iteration, the algorithm selects the smallest value in the unsorted list and migrates it to the sorted list. However, sorting the acyclic nets is different from sorting a list of elements, and we will use the number of crossings for each acyclic net with other acyclic nets to select the acyclic net which should be placed. Algorithm 3 shows the pseudocode of the proposed algorithm.

Each buffer place between $acnet_i$ to $anet_j$ generates $|i-j|-1$ crossings. Hence the number of crossings generated by the $acnet_i$, referred to as $k(i)$ is given by

$$k(i) = \sum_{j \neq i} (|i-j|-1)\cdot m_{i,j} \tag{5.5}$$

where $m_{i,j}$ is the number of buffer places between $acnet_i$ and $acnet_j$.

Eq.(5.5) could be rewritten by excluding the distance between the acyclic nets, as follows.

$$k'(i) = \sum_{j \neq i} c_{i,j} \tag{5.6}$$

Algorithm 3: selection placement (see Figure 5.4)

```
1  Function SelectionPlace():
2      Initialize array k of size n
3      Initialize empty H
4      for i = 1 to n do
5          Compute k(i) as per Eq.(5.5)
6      end
7      Let M = max(k)
8      H.add(M)
9      while there is acyclic net not added to H do
10         foreach acnet_i such that i ∉ H do
11             foreach acnet_j such that j ∈ H do
12                 Compute k(i) as per Eq.(5.6)
13             end
14         end
15         Let M = max(k)
16         H.add(M)
17     end
18     return H
```

Then, Algorithm 3 can be modified. This means that the criteria to detect the impact of crossings is computed just by looking at the number of buffer places between two acyclic nets.

Algorithm 3 aims to determine the optimal placement order for acyclic nets within a CSA-net to minimise crossings. It begins by initialising array k to store the computed crossing values for each acyclic net and the empty list H to represent the placement order. The core of the algorithm involves calculating the crossing values for each acyclic net. Initially, these values are computed using Eq.(5.5), which takes into account the relationships between acyclic nets in the net. Each crossing value indicates how many crossings an acyclic net would cause if placed at a particular position in the sequence. Once the initial crossing values are calculated, the algorithm selects the acyclic net associated with the highest crossing value and adds it to the placement order H. This process ensures that the acyclic net most likely to cause crossings is placed first, strategically reducing potential crossings. The algorithm then enters an iterative phase, where it continues to add acyclic nets to H. In each iteration, it calculates updated crossing values for the remaining acyclic nets based on their connections with the

already placed acyclic nets in H. These updated crossing values are computed using Eq.(5.6), which considers the influence of already placed acyclic nets on the potential crossings. The algorithm repeats this iterative process until all acyclic nets have been placed. The final placement order H represents the nearly optimal arrangement to minimise crossings within the CSA-net. By strategically selecting and placing acyclic nets based on their calculated crossing values, the algorithm effectively reduces the number of crossings in the net, resulting in an improved layout for visual representation.

Example 31. An illustrative example of Algorithm 3 is provided using the CSA-net depicted in Figure 5.1, which initially contains 6 crossings. The algorithm starts by computing the distance in line 5 using the equation, which yields the values of $k(i)$ for each acyclic net. Hence, the value of $k(i)$ is computed using the formula:

$$k(i) = \sum_{j \neq i} (|i-j|-1) \cdot m_{i,j}$$

where $|i-j|-1$ represents the number of crossings generated between the positions of nets $acnet_i$ and $acnet_j$, and $m_{i,j}$ is the number of buffer places between $acnet_i$ and $acnet_j$. This formula effectively calculates the crossings generated between $acnet_i$ and all other acyclic nets $acnet_j$. The calculated values are shown in Table 5.1.

Table 5.1 Explanation of k values for each acyclic net

Acyclic Net ($acnet_i$)	Calculation of $k(i)$	Resulting $k(i)$
1	$(\|1-2\|-1)\cdot m_{1,2}+(\|1-3\|-1)\cdot m_{1,3}+(\|1-4\|-1)\cdot m_{1,4}$	$k(1)=3$
2	$(\|2-1\|-1)\cdot m_{2,1}+(\|2-3\|-1)\cdot m_{2,3}+(\|2-4\|-1)\cdot m_{2,4}$	$k(2)=3$
3	$(\|3-1\|-1)\cdot m_{3,1}+(\|3-2\|-1)\cdot m_{3,2}+(\|3-4\|-1)\cdot m_{3,4}$	$k(3)=0$
4	$(\|4-1\|-1)\cdot m_{4,1}+(\|4-2\|-1)\cdot m_{4,2}+(\|4-3\|-1)\cdot m_{4,3}$	$k(4)=0$

That is, $k(i)$ represents the number of crossings generated by placing a specific acyclic net $acnet_i$ at a certain position in the sequence of acyclic nets. Specifically, the purpose of computing $k(i)$ is to determine the impact of each acyclic net on the overall number of crossings within the CSA-net.

Thus, by calculating $k(i)$ for each acyclic net, the algorithm can place the nets in a way that minimises the total number of crossings. Specifically, according to line 7 in the algorithm,

we set $M = 1$ as it is the maximum value in Table 5.1. Note that it is also possible to start with $M = 2$ as $k(2) = 3$. Then, acyclic net 1 is added to H; $H = (1)$, the sorted list. Now, we pin acyclic net 1 as we iterate using the loops in lines 9 and 10. Specifically, we re-compute the distance as in Eq. (5.5) from acyclic net 1. This yields $k(2) = 0$, $k(3) = 3$, and $k(4) = 0$. As $k(3) = 3$, acyclic net 3 is selected to follow acyclic net 1 in H; $H = (1,3)$.

In the next iteration, we recompute the distance from the remaining acyclic nets (2 and 4) with respect to those in H (1 and 3). This yields, $k(2) = 0$ and $k(4) = 1$. Therefore, acyclic net 4 is selected to follow acyclic net 3 in H; $H = (1,3,4)$. Intuitively, there is only one acyclic net remaining. Consequently, this yields a sorted list $H = (1,3,4,2)$. The outcomes of this approach are illustrated in Figure 5.4. Examination of the figure confirms that the selection sort algorithm successfully reduces 5 of the initial 6 crossings in the model.

◇

5.7 Experimental results and discussion

To test the placement algorithms proposed in the previous sections, we developed a Java code, which is available at [5]. The code was executed on a MacBook PC with a 2.2GHz Intel Core i7 processor and 8GB of RAM. We conducted experiments for each algorithm with varying numbers of acyclic nets, ranging from 5 to 100, running each test for 1000 iterations. The input matrices were randomly generated to specify the number of buffer places between each pair of acyclic nets. We calculated the number of crossings for the initially generated model and for the models resulting from the three placement algorithms. Then, we computed the average crossing number for each placement algorithm and the average for the initially generated crossings. From these averages, we derived the percentage of improvement using the formula:

$$\frac{(AvgInitialCrossing - AvgCrossing(A)) \cdot 100}{AvgInitialCrossing}$$

where $AvgCrossing(A)$ is the average crossing number for algorithm A. The results of this experiment are presented in Table 5.2. It indicates that selection placement is the most effective. For example, when $n = 100$, selection placement reduced the crossings by

approximately 81%, whereas the strongest competitor, bubble placement, managed to reduce the crossings by only about 9%. To further validate the results for the selection placement, we reran its outputs through the best competitor algorithm and recalculated the crossings. This is shown in the last column of Table 5.2. The results demonstrate that there is only about a 0.07% improvement for $n = 100$.

Table 5.2 Percentage of improvement for each algorithm and the splitting strategy. The first three columns show the results of bubble, insertion and selection placement, respectively. The fourth column shows the precentage of improvement after feeding the selection placement result to the bubble placements. The last column shows the effect of using the splitting strategy with selection placement.

n	BubblePlace	InsertionPlace	SelectionPlace	BubbleAfterSelection	Splitting
5	70.46	67.63	96.61	0.15	98.14
10	37.48	25.44	87.01	0.72	96.36
20	25.47	8.07	85.54	0.67	93.24
40	16.81	2.62	83.51	0.22	92.54
60	13.00	1.49	83.39	0.18	94.27
80	9.27	0.84	81.81	0.12	94.65
100	9.17	0.56	81.13	0.07	93.74

Table 5.3 The amount of improvement added as the number of items n increase.

Range	Bubble	Insertion	Selection
5-10	46.81	62.38	**9.94**
10-20	32.04	68.28	**1.69**
20-40	34.00	67.53	**2.37**
40-60	22.67	43.13	**0.14**
60-80	28.69	43.62	**1.89**
80-100	1.08	33.33	**0.83**
Mean	27.55	53.05	**2.81**

Moreover, we compared the percentage of improvement at each level with its predecessor. The outcome of this experiment is illustrated in Table 5.3. Consider, for example, the first row: '5-10' depicts the percentage of improvement at $n = 10$ with that at $n = 5$. Upon examining the last row, which represents the mean value, it becomes clear that the selection sort algorithm possesses the smallest mean, indicative of its steady and stable behavior. Moreover, closer inspection of the table reveals that as the number of acyclic nets n increases, the percentage of improvement tends to decrease. This is expected since, as the number of iterations

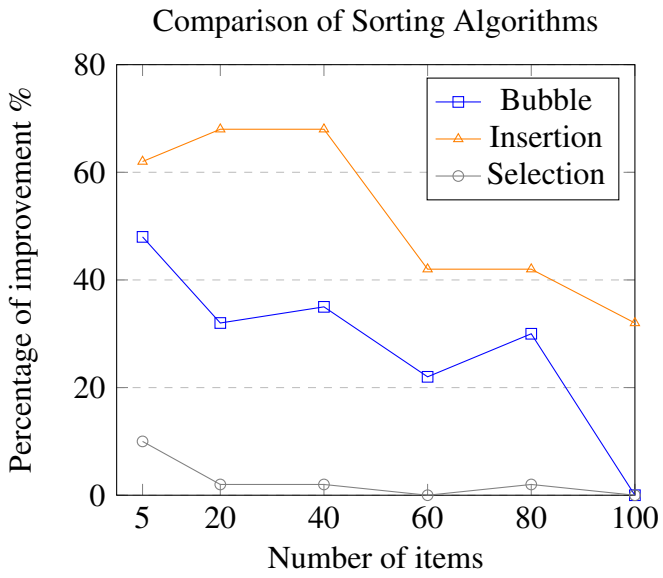

Fig. 5.5 Time taken by three sorting algorithms.

increases, the algorithms have less optimization to perform. An intriguing observation can be made when comparing these results with those in Table 5.2: at $n = 5$, the selection sort exhibits about 97% improvement. This accounts for its small mean improvement in Table 5.3, underscoring its high convergence speed. To visualise the behavior of the algorithm rather than simply quantify it, Figure 5.5 portrays the improvement percentage in relation to the number of iterations. A striking observation is the curve for selection sort, which tends towards the horizontal, denoting its resilience to the increase in the number of items.

Table 5.4 shows the comparison of three placement algorithms—bubble placement, insertion placement, and selection placement—across various metrics.

Table 5.4 Comparison of Bubble, Insertion, and Selection Placement Algorithms.

Metric	Bubble Placement	Insertion Placement	Selection Placement
Average Improvement (%)	25.38	15.23	85.71
Best Improvement (%)	70.46	67.63	96.61
Worst Improvement (%)	9.17	0.56	81.13
Stability (STD)	21.39	22.14	5.44

That is, the data in the table highlight the superior performance of the Selection Placement algorithm in reducing the number of crossings in acyclic nets. It consistently achieves the highest average and best improvements and performs reliably even in the worst-case

scenarios. The low standard deviation further underscores its stable performance. In contrast, Bubble Placement and Insertion Placement show higher variability and generally lower effectiveness in reducing crossings.

5.8 Result validation and evaluation

In the previous experiment, we assessed the algorithms' performance in terms of reducing crossings. To elaborate, we conducted a comparative analysis of the three sorting algorithms, focusing on the percentage reduction in crossings. This analysis led to the conclusion that the selection sort algorithm performed the best in terms of crossing reduction. In the subsequent experiment, our aim shifted to evaluating the practical applicability of a specific algorithm. To achieve this, we measured the time it takes to execute a specific operation, namely the replacement of acyclic nets (acyclic nets). These two experiments serve distinct research objectives, providing insights into different facets of the algorithms' functionality and usability.

5.8.1 Time evaluation

This experiment is focused on comparing the time required by the three sorting algorithms to determine the optimal placement. The results of this experiment are visually represented in Figure 5.6. The observation clearly indicates that the selection sort algorithm stands out as the fastest among the three algorithms in terms of execution speed. This finding underscores the practical advantages of selecting the selection sort algorithm, especially in scenarios where the prompt placement of elements is crucial.

5.9 Further improvement - splitting

The splitting strategy involves dividing the acyclic nets into two equal length lists. These two lists are then processed independently by the algorithm, allowing for a more specialised and tailored treatment of each group. The final placement, resulting from this dual processing,

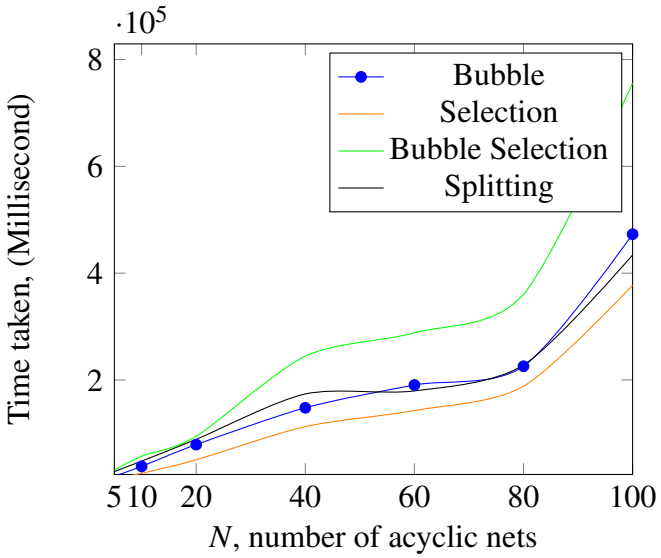

Fig. 5.6 A comparison of the time taken by each algorithm to reduce the crossing number. Note that the insertion placement running time is omitted as it falls in the range [0.02 - 23.46] millisecond.

becomes a combination of two stack-like placements, positioned side-by-side within the overall layout. This approach can potentially reduce the number of crossings further by isolating the nets into more manageable subsets.

The rationale behind the splitting strategy is to minimize crossings by dividing acyclic nets into smaller, more manageable groups. By processing these groups independently, the algorithm can optimize each subset, leading to a more organized and less cluttered overall layout. Crossings in the layout can be categorized into direct crossings and buffer-based crossings. Direct crossings occur between immediate connections, while buffer-based crossings involve intermediary buffer places. The splitting strategy primarily reduces direct crossings but can introduce buffer-based crossings.

Example 32. Consider the CSA-net shown in Figure 5.4 which is the selection sorting (resulting in 1 crossing) for the Figure 5.1 with initial 6 crossings. The idea is to split CSA-net in, for example, two columns with the aim of reducing the crossing further. This is depicted in Figure 5.7. That is, we can observe that we have zero crossing which in turn results in a much more pleasant and easy to track figure.

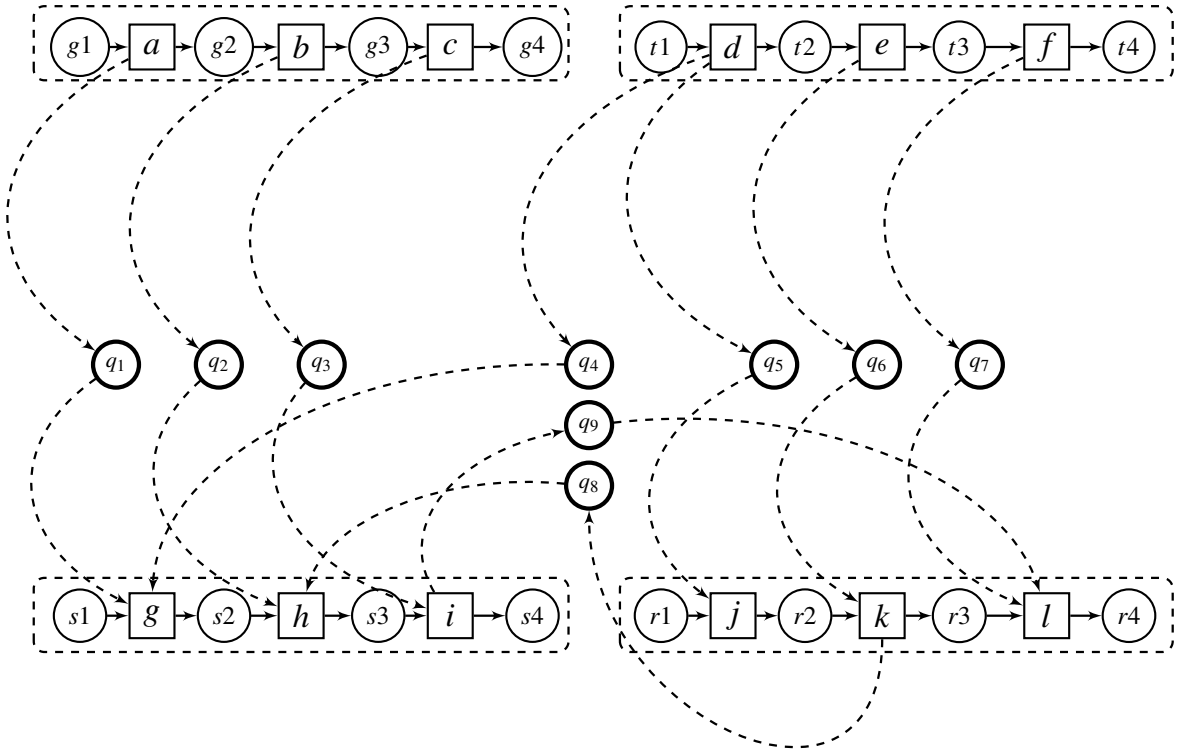

Fig. 5.7 A graphical representation of a CSA-net with 4 acyclic nets after applying splitting CSA-two equivalent lists.

Empirical data shows that the splitting strategy can significantly reduce the number of crossings. For example, in scenarios with 100 acyclic nets, the average reduction in crossings was approximately 40%. While the splitting strategy reduces direct crossings, it can introduce buffer-based crossings, which need to be managed carefully. Additionally, the increased complexity of managing two separate lists can impact computational efficiency. Future research could explore adaptive splitting strategies that dynamically adjust the size and number of splits based on the characteristics of the acyclic nets. Additionally, applying the splitting strategy to other domains, such as circuit design or network optimization, could yield further insights.

◇

Experimental results are shown in Table 5.2, where we can notice that there is a further improvement w.r.t. the results of selection placement. E.g., for $n = 100$, there is about 13% improvement which confirms the suitability of the splitting strategy to the placement of acyclic nets. However, this splitting strategy can also be adapted and applied with other placement algorithms, including bubble and insertion sort algorithms.

5.10 Conclusion

In this chapter, we focused on enhancing the visualisation of CSA-nets by optimising the placement of individual acyclic nets to minimise crossing numbers. We achieved this by adapting three well-known sorting algorithms. Our experimental results demonstrated the superiority of the selection placement strategy. Additionally, we discussed the splitting strategy to further enhance the placement process. This approach involves dividing the acyclic nets into two separate lists, each processed independently by the algorithm. The final placement is a combination of two stack-like arrangements positioned side-by-side. In the future we aim to refine these ideas by introducing dynamic placement strategies based on specific criteria such as time intervals or communication events. In essence, we intend to dynamically adjust the placement of acyclic nets, replacing them at predefined time intervals or after a set number of communications within a specific time frame.

Chapter 6

Detecting SYN Flood Attack Using CSA-nets

6.1 Introduction

Distributed Denial of Service (DDoS) attacks disrupt regular internet traffic by overwhelming servers with vast amounts of data, leading to resource depletion [73]. The diverse nature of malicious packets in these attacks complicates their analysis and makes detecting such attacks difficult [76]. Although DDoS attacks frequently target the transport layer, they can also exploit other layers using methods such as ICMP, SYN, and UDP flooding [58]. Attackers can manipulate the network layer by altering the IP packet header, flooding servers with irregular packets. Tracing the origin of these attacks is challenging due to the widespread use of IP spoofing. As a result, attackers leverage attributes like packet size, rate, bit rate, and arrival time to drain server resources [64]. The Transmission Control Protocol (TCP) — an essential internet protocol — initiates connections through a three-way handshake involving SYN and ACK messages [22]. TCP is widely used across the internet for various services, leveraging flags such as SYN, ACK, and RST to manage connection status and data transport. SYN-flooding attacks can be direct or involve IP address spoofing, with the most potent form being a distributed attack utilizing multiple zombie computers to inundate the target. Attackers exploit this by continuously sending SYN packets with fake IP addresses, leading

to network saturation and server unresponsiveness [10]. A TCP-SYN-flood attack, a subtype of DDoS attack, capitalizes on the TCP handshake process to overwhelm the targeted server, rendering it inoperable. In this assault, TCP connection requests flood in faster than the server can process, inducing network congestion. Such an attack not only disrupts server services but also compromises the security of the client-server communication channel. It is hard to detect, can arise without prior security alerts, and is considered permissible since it does not directly exploit network vulnerabilities or misuse resources [125].

CSA-nets as discussed in [69, 104, 13, 15, 6] are designed to analyse and visualise complex systems, such as those involving in cyber-attacks. They can aid in planning, tracing, and monitoring system efficiency. CSA-nets represent such a system as distinct acyclic nets interconnected by buffer places. These buffer places enable connections between subsystem components through both synchronous and asynchronous communications. Additionally, CSA-nets incorporate numerous properties in their formalisation, including causality, concurrency, and synchronisation, which can be invaluable for protocol analysis. In other words, they have the capability to model and analyse network communication protocols, aiding in the detection and understanding of abnormal network behaviours. Despite their potential, the application of CSA-nets for modelling network protocols and detecting connection abnormalities remains undiscovered. Thus, this mathematical model could assist researchers and practitioners in identifying the root causes of such anomalies, integrating the study and modelling of these behaviours with detection methodologies.

This chapter introduces a novel approach for analysing and visualising cybersecurity behaviours using CSA-nets. Specifically, it models clients and servers as acyclic nets, emphasising their communication via the three-way handshake. A new algorithm is introduced to discern communication between these nets. Essentially, the algorithm tracks packets that successfully complete the communication sequence and identifies any abnormal packets that fail in completing the process. That is, by using the capabilities of CSA-nets, this approach could offer deep insights into anomalous TCP activity and help detect malicious activities and uncover their root causes in the interactions between clients and servers.

6.2 Related work

A Distributed Denial of Service (DDoS) attack is a widespread network assault aimed at overwhelming computational resources and bandwidth, hindering the ability of legitimate users to access services. This attack typically involves substantial packet flooding, making it a more extensive version of a denial of service (DoS) attack. Attackers can effortlessly change their IP addresses, thus bypassing blacklisting. Over the years, numerous techniques for detecting and mitigating DDoS attacks have been developed, such as those targeting TCP flood attacks. Researchers [97] proposed a machine learning (ML) approach to identify DDoS attacks. This approach involves two main steps: feature extraction and model detection. The feature extraction process serves to remove superfluous features, isolating the most critical features of DDoS attack traffic. These features are then used as inputs in the model detection phase, which uses the random forest algorithm to train the attack detection model. Their experimental results suggest that the ML-based DDoS attack detection method yields a high detection rate for common DDoS attacks. In [111] argued that DDoS attacks from local networks pose greater threats than external ones due to detection complexities. Using a spacecraft simulator's real-time telemetry software, they analyzed both benign and malicious packets. Their detection method was designed for TCP and HTTP Flood DDoS attacks. They observed that the PSH&ACK flags, initially set low during regular data transfers, surged during attacks. Consequently, they introduced two algorithms. The primary one detects TCP floods by counting the PSH&ACK flags. If this count surpasses a predefined limit in a specific duration, it indicates an attack. A recent study [102] investigated machine learning (ML) algorithms for real-time DDoS attack detection, addressing prevalent attack types like UDP flood, ICMP ping flood and TCP-SYN-flood. A classification model was crafted based on ML, trained to distinguish benign from malicious network traffic. Decision Tree (DT), Random Forest (RF), and K-Nearest Neighbors (KNN) algorithms were found to be effective in detecting attack traffic. However, KNN presents computational complexity as it requires assessing the distance between the identified node and all other nodes in the training dataset, followed by computing these distances. Consequently, KNN can contribute to the computational workload of detection devices in the Software-Defined

Networking architecture and potentially result in significant detection delays. Furthermore, Decision Trees (DT) exhibit a shorter learning time compared to Random Forest (RF). That is, DT emerged as a more streamlined option, making it the favored classification model. A study [103] centered on detecting DoS attacks within the transmission control protocol (TCP) three-way handshake. The researchers introduced a detection and prevention mechanism for the TCP-SYN-flood attack through the use of an adaptive threshold. This threshold was calculated via the 'Adaptive threshold algorithm'. The outcomes highlight the proficiency of the proposed approach in identifying and preventing TCP-SYN-flood attacks by leveraging an adaptive thresholding technique. Moreover, a study [119] investigated ML and data mining algorithms for detecting DDoS attacks, particularly TCP-SYN-flood attacks, using the CAIDA dataset. The decision stump and the One-R (OR) algorithm were highlighted for their accuracy, with performance metrics supporting their effectiveness.

In the field of modeling, several studies have been undertaken. One such study [65] introduced a Petri net-based model to differentiate between DDoS attacks and legitimate communications, both on the server and client sides. When deployed on the server side, this model endeavors to reduce time complexity. It aims to identify and discard malicious packets, while allowing valid communications. Packets are categorised using a confidence index, specifically designed to prioritise against untrusted network attacks. Legitimate communications receive high index values and are positioned in a high-priority queue, whereas malicious communications are allocated to a lower-priority queue. This strategy enhances the detection and filtering of malicious attacks, leading to improved server efficiency. In addition, Structured Occurrence Nets (SONs) are another mathematical and graphical modelling formalism used to detect these types of behaviours. SONs were used to identify DNS tunneling attacks, a technique leveraged by adversaries to extract data from multiple accounts [9]. In this model, each packet is represented as an individual occurrence net. If a token securely transits from the start to the finish of the SON, the packet is recognized as legitimate. Conversely, if it does not, it is deemed indicative of a DNS attack.

Despite the widespread use of machine learning (ML) techniques in the development of intrusion detection systems (IDS) for detecting and classifying cyber-attacks, several challenges

arise. Malicious attacks are constantly evolving and occurring in large volumes, necessitating a adaptable solution [126]. As attacks adapt by altering their behaviors, the feature distributions may change, causing the trained models to perform poorly and fail to detect new attacks. This problem is referred to as domain-shift, which typically requires collecting new training data and retraining the model to adapt to changes in the target domain [48]. Thus, there is still a need for a deeper understanding of abnormal behaviors and the role of modeling techniques in detection. Developing novel anomaly detection techniques to improve learning, adaptation, and threat detection in diverse network environments is crucial [134]. Specifically, one could solution is combining the strengths of both methodologies can provide a more comprehensive and nuanced understanding of abnormal behaviors and enhance detection capabilities. By leveraging the predictive power of machine learning and the insights into structure and behavior from modeling, researchers can create a synergistic effect, resulting in more robust and reliable solutions.

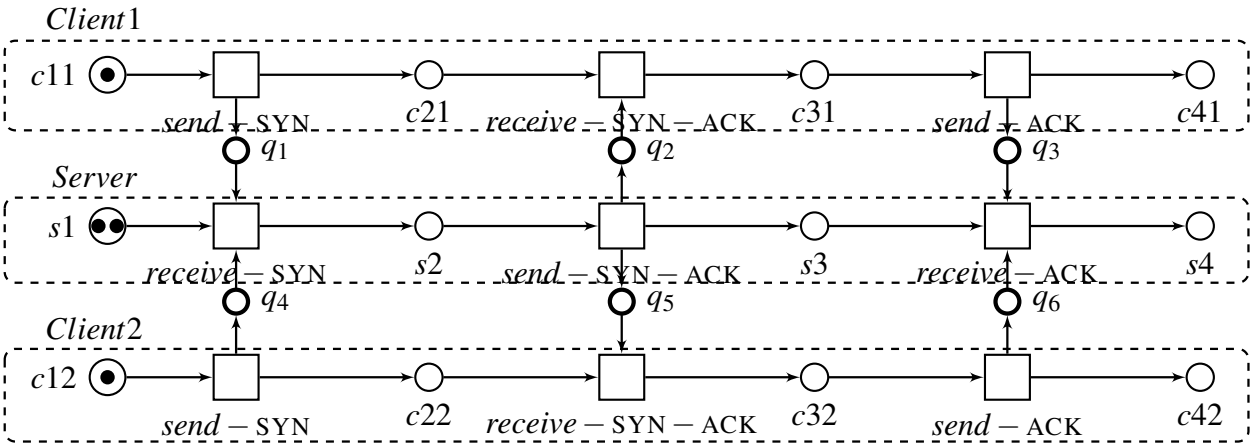

Fig. 6.1 Two clients ready to establish TCP Three-way handshake communication with the server.

6.3 TCP protocol

The CSA-net visualisation framework is designed to aid in the analysis of complex system such as cybercrime investigations. It uses acyclic nets to represent different systems, and these

individual acyclic nets are interconnected by buffer places that can model both asynchronous and synchronous communication between subsystems. The buffer places are employed to link events and connect the acyclic nets with each other. An example of this is depicted in Figure 6.1, which illustrates a CSA-net with two acyclic nets (CLIENT1, CLIENT2) representing clients connected to a server represented by an acyclic net (SERVER) through buffer places $q_1, q_2, \ldots, q_6$. CSA-net offers a set of promising built-in features that enable the analysis of systems under study and understanding their behaviours and interactions. Coloured tokens are one of these characteristics that aid in the non-overlapping of interacting tokens (e.g., packets), providing the ability to distinguish between packets based on unique features that prevent them from interfering with each other. Another feature of CSA-net is reachability, which is a verification property used to track the reachability of tokens from the initial marking. These properties allow for the modelling and tracking of both the normal behaviours of the TCP protocol for analysis and visualisation, as well as abnormal behaviours that may indicate a DDoS attack.

6.4 Coloured communication structured acyclic nets (ccsa-nets)

A coloured communication structured acyclic net consists of multiple disjoint acyclic nets that can communicate through buffer places, which allow instantaneous transfer of tokens and can involve a cycle only when synchronous communication is implemented. Given the material presented in the preceding chapters, some technical definitions are omitted or described informally.

Definition 6.4.1 (coloured acyclic net) *A* coloured acyclic net *is a tuple*

$$acnet = (P, T, F, col, ex, gd),$$

where P and T are disjoint finite sets of places *and* transitions *respectively, and $F \subseteq (P \times T) \cup (T \times P)$ is the* flow relation *such that:*

- *P is nonempty and F is acyclic.*
- *For every $t \in T$, there are $p,q \in P$ such that pFt and tFq.*
- *col is a mapping assigning nonempty finite set of colours to every place.*
- *ex is an arc expression function that assigns an arc expression to each arc $x = (p,t)$ or $x = (t,p)$ so that $Type[ex(x)] = col(p)$.*
- *gd is a mapping assigning a boolean guard to each transition. (In this case guards are very simple and ensure that all the input tokens used in transition firing are of the same colour and all the tokens produced have the same colour too.)*

Definition 6.4.2 *A* coloured communication structured acyclic net (or CCSA-net) *is a tuple*

$$ccsan = (cacnet_1, \ldots, cacnet_n, Q, W, col', ex') \quad (n \geq 1)$$

such that:

- *$cacnet_1, \ldots, cacnet_n$ are colured acyclic nets.*
- *$csan = (acnet_1, \ldots, acnet_n, Q, W)$ is a well-formed* CSA*-net (as defined in [6]), where each $acnet_i$ is an acyclic net obtained from $cacnet_i$ after deleting the last three components.*
- *col' is a mapping assigning nonempty finite set of colours to every buffer place in Q.*
- *ex' is an arc expression function that assigns an arc expression to each arc $x = (q,t)$ or $x = (t,q)$, where $q \in Q$, so that $Type[ex(x)] = col(q)$.*

The execution semantics of CCSA-nets follows the standard coloured net rules as well as the execution rules of CSA-nets [6] and is omitted in this Chapter. Markings of a CCSA-net assign sets of suitable coloured tokens to the places of *csan* as well as the buffer places. The execution rule is basically the same as for CSA-nets assuming that all the input tokens used in transition firing are of the same colour and all the tokens produced have the same colour

Table 6.1 Detailed description of client and server acyclic nets

Content	meaning
c_1	Client is ready to initiate a connection SYN
snd-SYN	Client sends request to server to establish connection
c_2	Client waits for response from Server
rcv-SYN-ACK	Client receives response from Server ACK
c_3	Client is ready to send SYN/ACK to Server
snd-ACK	Client sends SYN/ACK to Server
c_4	Client is ready to establish connection and push data to Server
s_1	Server is ready to initiate connection with Client
rcv-SYN	Server receives request from Client to establish connection
s_2	Server is ready to send ACK to Client
snd-SYN-ACK	Server sends ACK to Client
s_3	Server waits for response from Client
rcv-ACK	Server receives SYN/ACK from Client
s_4	Server is ready to establish connection and push data to Client
x,y	represent the expression on the arc, which can be a function or operator.

too. Intuitively, this means that the executions corresponding to different colours do not interfere with each other (see Figures 6.3-6.9) Thus, coloured tokens ensure non-overlapping of interacting tokens (e.g., packets), allowing differentiation based on unique features to prevent interference [6]. In the CCSA-net case, a well-formed step sequence means that no place or buffer place is filled by the same coloured token more than once in any given step sequence. This mechanism allows for synchronising transitions from different acyclic nets (e.g., clients and server), which then aids in modelling the behaviours of the three-way handshake process. *ccsan* is 'well-formed' and so its executions (scenarios) allow to represent well-defined causal relationships and properties. These properties can assist in detecting and recording the entire causal history and so help in detecting abnormal behaviours that might occur during the three-way handshake process.

6.5 Analysing three-way handshake by CCSA-net

In this section, we analyse the three-way handshake using CCSA-net representation. Specifically, we rely on the structures and behavioural properties to model and analyse such kinds of cybercrime investigations. In other words, we use acyclic nets to represent different

subsystems (Client and server) as individual acyclic nets and connect them through buffer places to model both asynchronous and synchronous communication between them.

6.5.1 Structure of TCP model

Structurally, the three-way handshake process can be represented by two different acyclic nets linked by buffer places. Figure 6.2 demonstrates the three-way handshake process between clients and the server using a CCSA-net, which contains separate acyclic nets representing clients (the upper one) and the server (the lower one). The places/status are represented by circles, showing the current status of the process (e.g., SYN, SYN-ACK or ACK), while transitions — represented by squares — are responsible for transferring tokens from one place/status to another. Moreover, buffer places q_1, q_2 and q_3 — represented by bold circles — are responsible for transferring tokens between different acyclic nets synchronously or asynchronously. Expressions and guards can be displayed on arcs and transitions (e.g., x and y in Figure 6.2), and these expressions can be functions or operators written in ML (a programming language). This adds additional constraints on the arcs and transitions, e.g., determining the amount of traffic the server can process at a time or the time limit the server waits for an acknowledgment as we used in classifier model in Figure 6.10. Table 6.1 provides a detailed description of the Client and Server nodes. Initially, tokens representing different clients can be placed in c_1 and be moved by firing transitions (from place to another place by firing a transition). Each token has different colour to distinguish between different clients during communication. Table 6.2 gives detailed description of the features of each token. Therefore, we can analyse the handshake process based on the model's structural properties to ensure its soundness. Moreover, the structural properties of CCSA-net such

Table 6.2 Detailed description of colour sets of token

colour set	description
IP-source	contains the IP address of the source machine
IP-destination	contains the IP address of the destination machine
source-port	contains the source port number of the source machine
destination-port	contains the destination port number of the destination machine
TCP-flags	contains the flags of TCP packet (SYN, ACK or RST)

as causality can aid our understanding of how packets are transmitted between Client and Server through three-way handshake communication. This can provide administrators and investigators with insights into the causes of DDoS attacks and help them understand how to prevent future attacks. Specifically, this mechanism allows recording the causality between executed transitions which helps in detecting and tracing the process of such communication. Thus, investigators can benefit from extracting the causality of fired transitions to visually understand abnormal behaviours during the connection before delving into the cause and effect of such behaviour. That is, the three-way handshake can be effectively represented (structurally) using a properly structured CCSA-net.

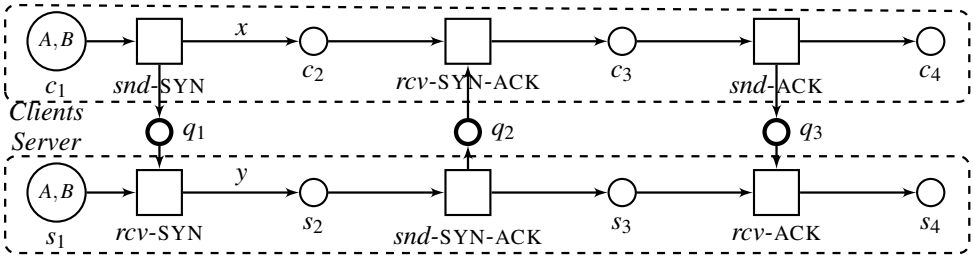

Fig. 6.2 CCSA-net model for communication between clients and server showing two clients (tokens) ready to establish connection with server.

6.5.2 Behaviour of TCP model

To analyse the three-way handshake behaviourally, we will leverage the built-in features and formal properties inherent in the semantics of CCSA-nets [6]. These can be used to analyse communication behaviour and detect abnormal situations that might arise during a connection. Such properties can be discovered by the reachability analysis, which verifies the traceability of tokens from the initial marking, and well-formedness, guaranteeing a clear representation of causality. Both these mechanisms allow us to model and trace the standard behaviours of the TCP protocol and to pinpoint abnormal behaviours, such as SYN-flood attacks. Note that well-formedness is a fundamental criterion for CSA-nets and also CCSA-nets, and it essentially guarantees a clear representation of causality in the behaviours of the net. The CSA-net in Figure 6.2 is well-formed and so it ensures a clear representation for each token

during the connection, assisting in detecting specific abnormal behaviours in communication between clients and servers.

6.6 TCP three-way handshake and CCSA-nets

In the preceding section, we analysed the three-way handshake based on the structure and behavioural properties of CSA-nets, discussing specific properties of this model. This section demonstrates and utilises the behaviours (steps) of the three-way handshake by employing CCSA-net to analyse and detect abnormal behaviours (SYN-flood).

6.6.1 Normal behaviour

The TCP protocol employs a three-way handshake to establish a reliable connection between two devices. This procedure is captured in three principal steps using the CCSA-net. As depicted in Figure 6.2, tokens A and B, which represent two packets, reside at c_1. Through the *snd*-SYN event, the client sends a SYN token, transmitting, for example, token A to c_2 and q_1, as illustrated in Figure 6.3. Upon receiving this, the server's *rcv*-SYN action is activated, transferring token A from s_1 to s_2 (as shown in Figure 6.4). This progression signifies the client's anticipation of the server's response to make the initial SYN communication. In the second handshake phase, with token A positioned at s_2, the SYN-ACK action is triggered, sending token A to s_3 and q_2 (as shown in Figure 6.5). Subsequently, the client acknowledges the server's SYN-ACK action through the *rcv*-SYN-ACK event, moving token A to c_3, as captured in Figure 6.6. In the final handshake step, the client responds via the *snd*-ACK action, shifting token A from c_3 to c_4 and q_3 (as represented in Figure 6.7). The server's subsequent *rcv*-ACK action in Figure 6.8 finalises the handshake, ensuring an established connection between both entities.

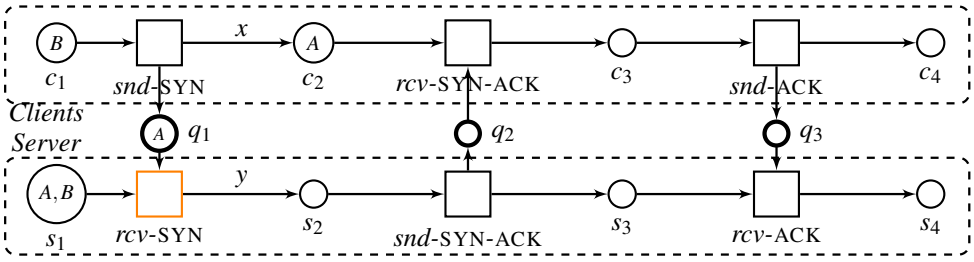

Fig. 6.3 Server received client's request and preparing to respond to this request.

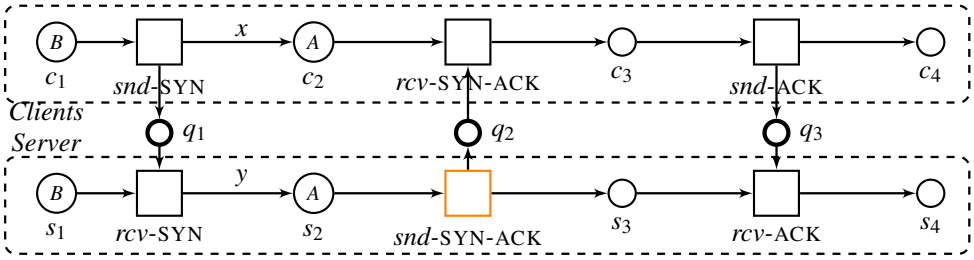

Fig. 6.4 Second handshake SYN-ACK using *snd*-SYN-ACK and *rcv*-SYN-ACK.

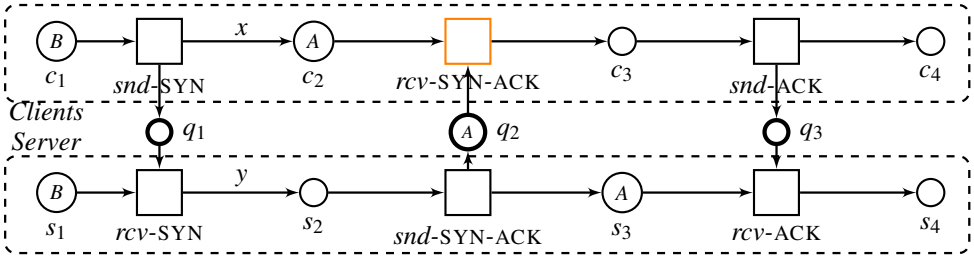

Fig. 6.5 Client received server's request and preparing to response for server's request.

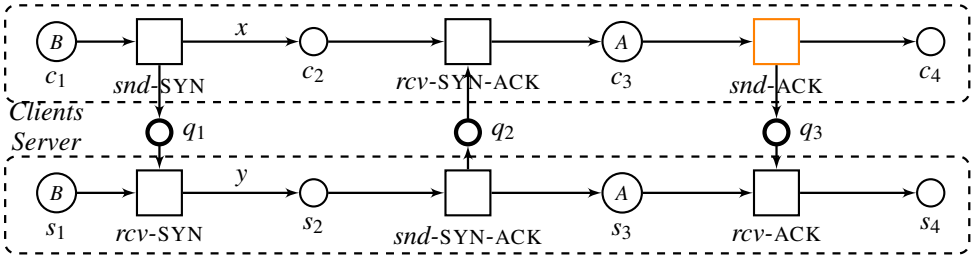

Fig. 6.6 Connection for third-handshake using *snd*-ACK and *rcv*-ACK events.

6.6.2 Abnormal behaviour

In the abnormal behaviour of the SYN-flood attack, attackers manipulate the standard behaviour to inundate a server, thereby making detection increasingly challenging. As depicted in Figure 6.9, the server fails to receive the third handshake response (ACK) from the client.

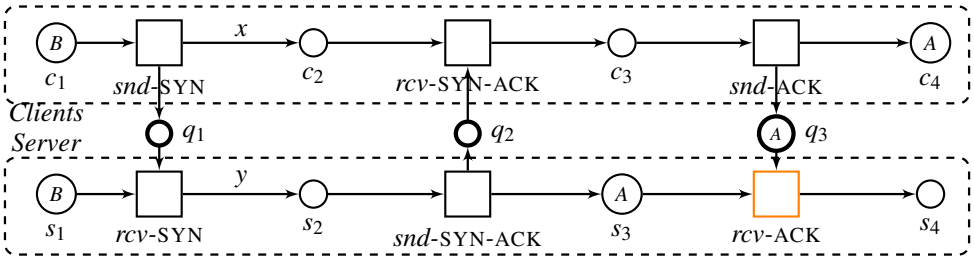

Fig. 6.7 Server received client's acknowledgment and preparing to start reliable connection with client.

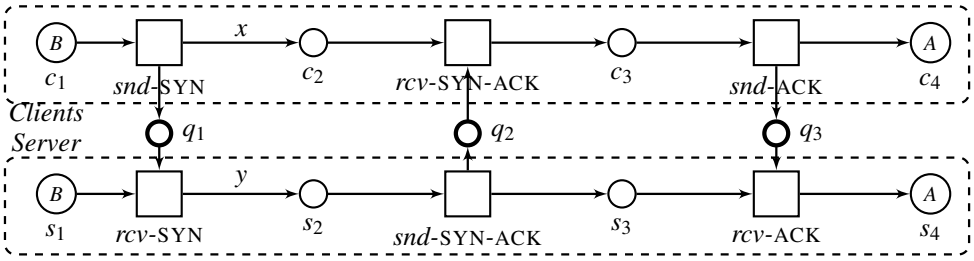

Fig. 6.8 Both client and server are ready to push data between each other.

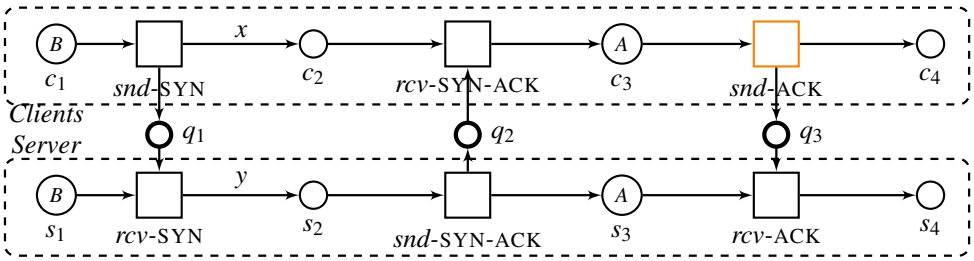

Fig. 6.9 Abnormal behaviour as token A is frozen in c_3.

Even though the *snd*-ACK transition is enabled, token A remains frozen at c_3. This strategy overwhelms the server with a surge of SYN requests, leading to numerous half-open connections and consequently making the server unresponsive.

6.7 Timed-CCSA-nets

This section introduces the method for integrating timing information into CCSA-nets. This integration enables TCCSA-nets to elaborate on the system's performance, capturing the system's operations in real-time. Moreover, with this timing feature, CCSA-nets become suitable for modelling systems where correctness depends on event timing. In contrast to

CCSA-nets, this timed variant offers deeper insights into behaviours, such as the average system runtime and adherence to deadlines for real-time processes. The primary distinction between timed and untimed CCSA-nets lies in their token structure, as tokens now include additional values denoting time. The token time (timestamp) quantifies the duration of execution for each token. This timestamp is set to zero for each token in the initial markings. Token timestamp is denoted by T_s. For instance, $T_s = 0$ indicates that the token time is zero, signifying that the token remains in its initial marking. Once the transition $snd-\text{SYN}$ is executed, the token time starts and begins to increment based on the firing of transitions. Figure 6.10 provides an illustrative example of a TCCSA-net. The formal notation for TCCSA-nets is provided in Definition 6.7.1.

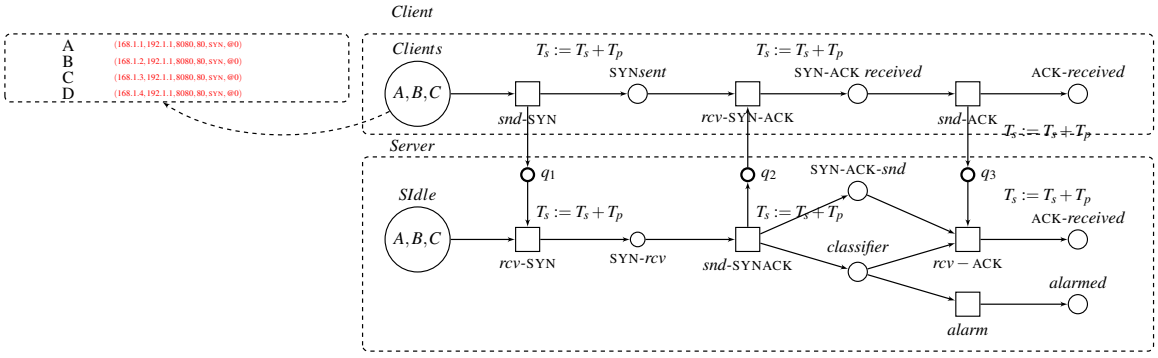

Fig. 6.10 CSA-net model after applying classification for normal and abnormal behaviours.

Definition 6.7.1 *A* Timed coloured communication structured acyclic net (or TCCSA-net) *is a tuple*

$$ccsan = (cacnet_1, \ldots, cacnet_n, Q, W, col', ex') \quad (n \geq 1)$$

such that:

- $cacnet_1, \ldots, cacnet_n$ *are coloured acyclic nets.*
- $csan = (acnet_1, \ldots, acnet_n, Q, W)$ *is a well-formed* CSA*-net (as defined in [6]), where each* $acnet_i$ *is an acyclic net obtained from* $cacnet_i$ *after deleting the last three components.*

- *col' is a mapping assigning nonempty finite set of colours to every buffer place in Q. Each colour set is now can be timed or untimed.*
- *ex' is an arc expression function that assigns an arc expression to each arc $x = (q,t)$ or $x = (t,q)$, where $q \in Q$, so that $Type[ex(x)] = col(q)$.*

That is, Definition 6.4.2 is extended by introducing a timed colour set for each token, enabling it to carry a time value during execution. Specifically, the colour set *col* can be either timed or untimed. All places associated with a timed colour set are termed 'timed places', and the arcs connected to these places are named accordingly. Additionally, these arcs can carry an expression function or operator, expressed in ML (for instance, a programming language). This introduces further constraints on the arcs. In ML, transitions can also feature a time delay expressed as a type TIME or guard associated with transitions that evaluate to either true or false based on the current marking (place). Thus, the colour set of a token t can be described as a vector composed of six types, detailed as follows:

$$(IP_{src}, IP_{dst}, Port_{src}, Port_{dst}, Flag, T_s).$$

Time values are updated during the model's execution (i.e., upon each event firing). At any given point, we can assess the status of the token by inspecting its values. Once the token's execution is finished, the value T_s represents the duration of a token has spent within the model (e.g., completing a three-way handshake). Algorithm 4 outlines the concept of initialising timing for CCSA-nets tokens. Intuitively, time increments with the firing of a token at each place, which indicates the duration taken for the token to transition from its current place to the next.

The process time T_p quantifies the duration of consuming the token for each transition. This facilitates the computation of metrics, such as the average time of processing transitions, which provides deeper insight into the analysis of such communication systems. Moreover, it is assumed that each transition can be fired once its input places are present, depending on the time needed to consume a token by the transition (e.g., no restriction on firing transitions). In other words, the time required for a transition to consume a token can vary based on

Algorithm 4: TCCSA-nets

Input : CCSA-net
Output : TCCSA-net

Initialization
for ***each token*** i ***in the initial marking*** **do**
    Initialize the token colour set. Initialize the token timestamp $T_s[i]$ to zero.
end
Execution
while CCSA-*net is executing* **do**
    for ***each token in the initial marking*** i **do**
        Once the token is fired in the first event Update the token timestamp $T_s[i]$ by adding the processing time T_p of the transition.
    end
end

different criteria, such as network latency or bandwidth. However, delays may occur within the process. Specifically, in attack scenarios, a server could wait for an acknowledgment from the client for an indeterminate period. In other words, attackers can force client machines to delay or stop sending acknowledgments to the server, resulting in the process time of the (*snd-ack*) event not being completed (i.e., creating a half-open connection). Thus, in the model, transitions may include delay expressions, represented as expressions of the TIME type. However, it is assumed that there are no time delays on transitions $snd - \text{SYN}$ and $snd - \text{SYN} - \text{ACK}$ and output arcs. In other words, we assume the first and second handshake are completed within a normal timeframe between clients and the server. This assumption simplifies the analysis at this stage, ignoring certain types of delays that could occur during the first and second handshakes, and focuses on computing the token's age within the model and comparing it with a predefined threshold to distinguish between normal and abnormal activities.

That is, the cumulative time within the model includes all timestamps resulting from the execution of all transitions. For example, if the initial event ($snd - \text{SYN}$) starts with a timestamp of zero (the initial marking at the beginning of the model's execution), and the process time T_p for this transition is five seconds (i.e., the duration required to consume the token in the $snd - \text{SYN}$ event), this indicates that the token takes (0 + 5) seconds to transition

to the subsequent place ($rcv - \text{SYN}$). In other words, we add the current timestamp of the token to the process time for each transition. Consequently, we compute the cumulative time throughout each token's execution to discern its behavior. Our aim is to identify any anomalies by detecting tokens that exceed the specified threshold for receiving acknowledgment from the client in each communication.

6.8 Classification and simulation tokens by TCCSA-net

The extension of TCCSA-net provides a beneficial framework for identifying unusual behaviours, such as those exhibited in SYN-flood attacks. It uses the timing attribute of tokens in combination with a predefined threshold to analyze and detect abnormal patterns by classifying tokens. Specifically, we rely on a predefined threshold to classify tokens. These tokens are identified using the flow ID/colour set, which includes IP source, IP destination, Source port, Destination port, Flags, and Timestamp. The algorithm assesses each token/packet independently, determining whether it is indicative of abnormal behaviour or standard communication. For instance, as illustrated in Figure 6.15, tokens can be classified as abnormal if their time values exceed the predefined threshold set for the alarm transition.

Formally, in TCCSA-nets, events can only be enabled if their input tokens are present in the preceding places. We rely on these properties, combined with time, to detect any abnormal activities that can occur during communication. In practical terms, for example, the classifier will send the token to $rcv - \text{ACK}$ when the server acknowledges it. In other words, the $rcv - \text{ACK}$ event on the server side will only be enabled when all its inputs are in place to fire $rcv - \text{ACK}$ and complete the three-way handshake. This means the $rcv - \text{ACK}$ event will not be activated until the client sends their acknowledgment (see Figure 6.15). That is, an *alarm* which is enabled event is executed if the token's time exceeds the threshold, indicating a potential SYN-flood attack. This means, when the *alarm* transition is fired, the $rcv - \text{ACK}$ transition will not be executed and $\text{ACK} - \textit{received}$ place will not be reachable. The reason is that $rcv - \text{ACK}$ will not have all its input places available to fire. Specifically, the token in the classifier place will be consumed by the *alarm* transition.

The threshold, denoted as τ, is a predefined threshold representing the maximum allowable time the server waits to *rcv-ack* from the client. We assumed threshold to be 30 seconds, which represents the maximum duration the server will wait to receive an *rcv-ack* from the client, completing the three-way handshake. It is important to note that the server's waiting duration can vary based on the criteria set by the network administrator. The procedure for identifying abnormal behaviour is outlined in Algorithm 5. Specifically, the algorithm starts by defining the maximum allowable time τ. Moreover, every communication is portrayed as a token. The timestamp T_s of token i is expressed as:

$$T_s := T_s + \Sigma(T_{p_i}) \tag{6.1}$$

That is, T_s denotes the timestamp for a token at any specific time, and T_p signifies the time needed to consume token by transition. For instance, if the timestamp at a specific point,(e.g., SYN-*rcv* place) is 5 seconds, and the process time, which represents the time needed to consume the *send*-SYN-ACK transition, is 3 seconds, then the timestamp upon firing *send*-SYN-ACK will be $5+3=8$ seconds. In other words, we add the process time for consuming each token to the timestamp.

The equation computes the token's timestamp for each token independently, combined time required to complete all transitions inside the model. Note that, the goal of computing process time (time needed by transition to consume token) is to give accurate measure of time needed by each transition which can help in analysis such a communication. For example, we can benefit from process time T_p to calculate the average time for server to consume client token. Note also that this approach can be further extended and refined by examining various communication statuses and packet properties, such as response averages, traffic bandwidths, or server capacities, which is a topic for future work and could improve the detection of such behaviours. However, detecting a SYN-flood attack can be modelled as a classification problem that differentiates between the network flow states of 'attack' and 'normal'. In this context, we rely on our new TCCSA-net extension to discern and differentiate between normal and abnormal behaviours. Specifically, we allow server to operate within designated timeframes to process client requests.

Example 33. Let T_p be the processing time for each step (transition), and T_s be the cumulative sum at each step. The cumulative summation can be expressed as:

$$T_s = 0 + T_{p1}$$
$$T_s = T_s + T_{p2}$$
$$T_s = T_s + T_{p3}$$
$$T_s = T_s + T_{p4}$$
$$T_s = T_s + T_{p5}$$
$$T_s = T_s + T_{p6}$$

where:

- $T_{p1} = 4$ seconds ($snd - \text{SYN}$),
- $T_{p2} = 3$ seconds ($rcv - \text{SYN}$),
- $T_{p3} = 4$ seconds ($snd - \text{SYN} - \text{ACK}$),
- $T_{p4} = 3$ seconds ($rcv - \text{SYN} - \text{ACK}$),
- $T_{p5} = 5$ seconds ($snd - \text{ACK}$),
- $T_{p6} = 2$ seconds ($rcv - \text{ACK}$).

Thus, the summation simplifies to: $T_s = (0+4)+3+4+3+5+2$ and so $T_s = 21$. This represents a cumulative approach where the initial T_s starts at 0, and each subsequent processing time is added to the running total of T_s, demonstrating a continuous build-up on the last total. Note that this method will be applied to each token, ensuring that we compute the timestamp for all tokens within the model.

$\diamond$

Furthermore, T_t quantifies the timer that starts once the *alarm* transition is enabled for each token. If a token reaches $rcv - \text{ACK}$ before the T_t timer expires, then the $rcv - \text{ACK}$ transition will be triggered; otherwise, the *alarm* transition will be executed. It is important to note that

the total timestamp of a token may vary upon reaching the classifier. This variation occurs because packets (tokens) can experience delays or latency within the network. However, the classifier will add an additional 30 seconds to the token's timestamp before classifying it as abnormal. For instance, if the token's timestamp is 19 when it reaches the classifier place, then the maximum total waiting time for such a token is 49 seconds before it is classified as abnormal. This approach will be applied to each token within the model. It is crucial to understand that this approach is employed to demonstrate that TCCSA-net can be used as promising tool to analyse and visualise the three-way handshake in order to detect abnormal activities that could occur during communication.

Algorithm 5 is designed for classifying tokens which represent communication packets, into categories of either normal or abnormal based on their adherence to a predefined threshold. Specifically, it processes each token independently using a time threshold τ and a timer T_t that initiates upon a token's arrival at a classifier place. The primary objective is to determine the legitimacy of each token's communication pattern within a specified duration. Upon each token's arrival at the classifier, a timer specific to that token starts, measuring the elapsed time since its classification process began. The algorithm employs a straightforward decision-making criterion to ensure the status of each token: if a token reaches the designated $rcv-\text{ACK}$ transition before its associated timer exceeds the τ threshold, it is deemed normal. This classification implies that the token's communication is within expected parameters, suggesting legitimate traffic. Conversely, tokens failing to meet this condition within the allowed timeframe are classified as abnormal, indicating potential anomalies or security concerns, such as participation in a SYN-flood attack. This methodological approach enables the effective identification of communication packets that deviate from expected temporal patterns, offering a mechanism for flagging potential threats.

Example 34. Let us revisit the scenario with the cumulative time $(T_s + T_p)$ required for a token to complete its process, and introduce a classification condition $\tau \leq T_t$, where τ is set to 30 seconds. The dynamic timer T_t starts from zero upon the token's arrival at the classifier place.

Algorithm 5: Token Classification Algorithm for Detecting Abnormal Tokens

Input : N tokens from m clients representing communication packets,
τ // Time threshold in seconds,
T_t // Timer for each token to measure elapsed time since processing started

Output : Classification of each token as either normal or abnormal

Initialization:
Start processing tokens
for $i = 1$ *to* N **do**
  if *token i reaches classifier place* **then**
    Start timer T_t for token i
    if $rcv - \text{ACK}$ *transition is enabled before* T_t *exceeds* τ **then**
      Token i is classified as normal
    end
    else
      Token i is classified as abnormal
    end
  end
end

For this example, consider the processing times leading to a cumulative timestamp T_s as follows:

- $T_{p1} = 4$ seconds $(snd - \text{SYN})$,
- $T_{p2} = 3$ seconds $(rcv - \text{SYN})$,
- $T_{p3} = 4$ seconds, $(snd - \text{SYN} - \text{ACK})$

resulting in a cumulative processing time:

$$T_s = (0+4)+3+4 = 11$$

seconds.

Upon reaching the classifier, we evaluate the classification condition with $\tau = 30$ seconds and a dynamic timer T_t that counts up from zero. Assuming that T_t reaches 10 seconds by the time the classification decision is made, we insert these values into the classification condition:

$$30 \leq 10$$

In this case, the condition $\tau \leq T_t$ is not satisfied, indicating that the token does not meet the classification criteria based on the set threshold within the given time. In other words, the packet (token) still in the normal time-frame. Thus, *alarm* transition will not fire until the $\tau \leq T_t$ is satisfied. Conversely, assuming that T_t reaches 31 seconds by the time the classification decision is made, we insert these values into the classification condition:

$$30 \leq 31$$

That is, in the case the condition $\tau \leq T_t$ is satisfied, indicating that the token meets the classification criteria based on the set threshold within the given time. In other words, *alarm* transition will be fired.

$\diamond$

This example demonstrates how the classification of tokens is determined by comparing their processing time against a fixed threshold τ and the elapsed time T_t since the token's arrival at the classifier. The condition provides a straightforward method for classifying tokens based on their processing timelines and the dynamic context of their arrival at critical checkpoints.

6.8.1 Simulation of SYN-flood attack by Timed-CSA-net

In this section we will provide more details about Timed-CSA-net by simulating firing steps. Initially, when client is ready to send a request to the server, transition $snd - \text{SYN}$ is enabled and ready to fire. Once $snd - \text{SYN}$ fires, token A will be sent to q_1 and $\text{SYN} - sent$, and the timestamp will be updated based on process time needed to consume $snd - \text{SYN}$ transition as illustrated in Figure 6.11.

Figure 6.12 demonstrates that transition $rcv - \text{SYN}$ on server side is enabled and ready to fire. Consequently, $rcv - \text{SYN}$ is executed, and token A will be sent to the $\text{SYN} - rcv$ place. Additionally, the timestamp is updated.

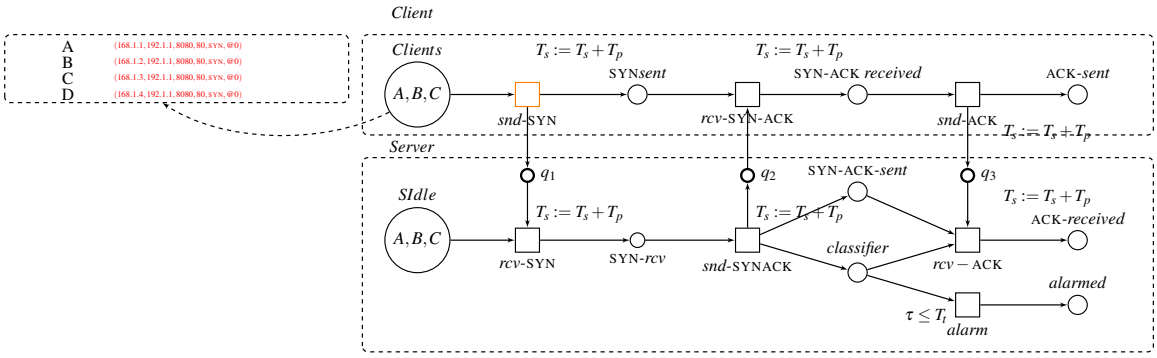

Fig. 6.11 *snd* − SYN is enabled and ready to fire with the timestamp 0.

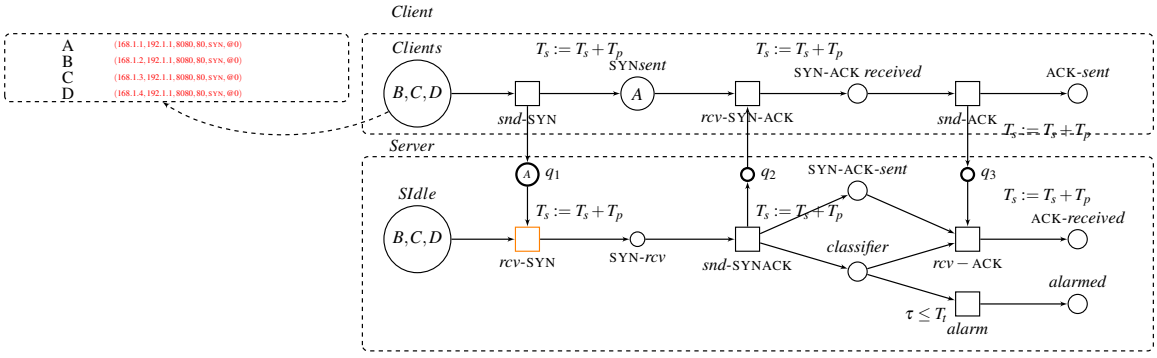

Fig. 6.12 transition *rcv* − SYN on server side is enabled and ready to fire

In Figure 6.13, the transition *snd* − SYN − ACK is enabled. Consequently, it executes, sending the token to the *classifier*, SYN − ACK − *snd* places, and the buffer place q_2.

In Figure 6.14, the *alarm* transition is enabled; however, it will not execute due to a guard condition on the transition, meaning it will only execute if the condition is satisfied. At the same time, the *rcv* − SYN − ACK transition is enabled and executed, sending token A to the SYN − ACK − *received* place.

Figure 6.15 demonstrates that the client is ready to send an acknowledgment to the server. Therefore, by sending token A to q_3 and ACK − *received*, the client sends its acknowledgment to the server, waiting for the server to accept the acknowledgment.

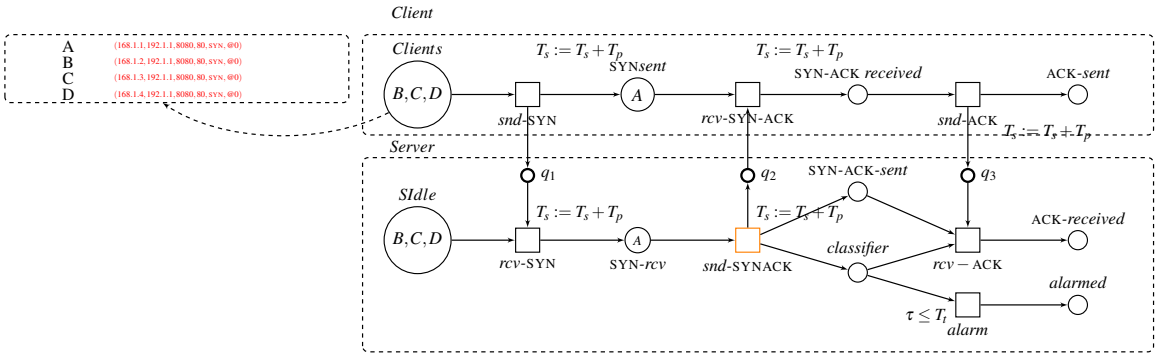

Fig. 6.13 transition *snd* − SYN − ACK on the server side is enabled and ready to fire.

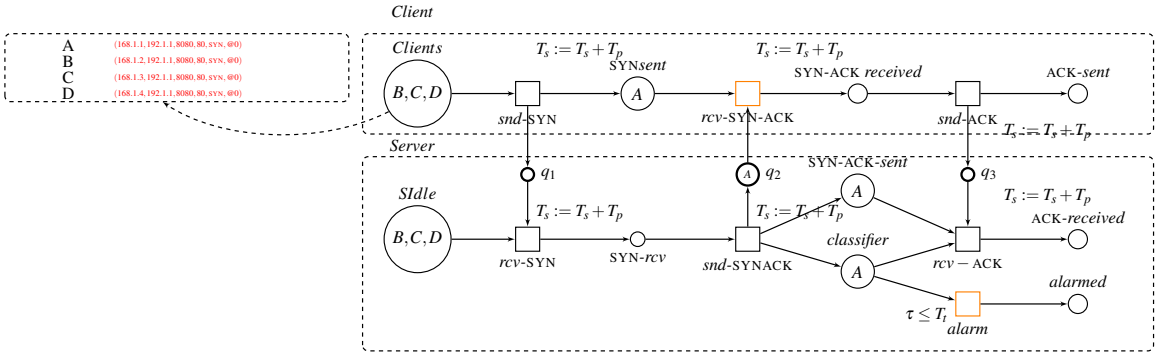

Fig. 6.14 *alarm* transition is enabled and timer T_t is started.

In Figure 6.16, the transition *rcv* − ACK is enabled as all its input places are present. Consequently, it will execute, confirming that the acknowledgment from the client has been received, and data transmission between them will commence.

Figure 6.17 shows that both side received acknowledgment and data transmission between them will start.

In Figure 6.18, potential attack behaviors can occur. Specifically, even though the transition *snd* − ACK on the client side is enabled, attackers may force client machines to refrain from sending acknowledgments to the server. However, the timer in the condition on the *alarm* transition evaluates this communication based on the condition assigned to the *alarm* transition. In other words, the *alarm* transition will wait until the threshold time is exceeded; then, it will send token *A* to the *alarmed* place, indicating a potential attack.

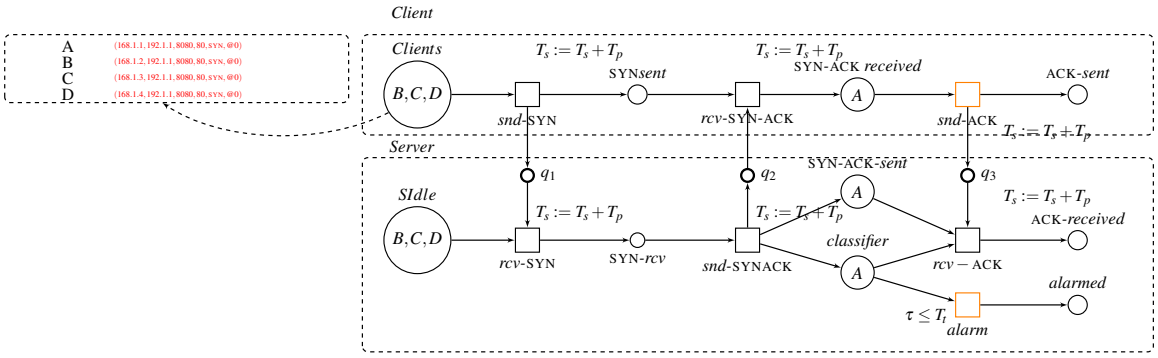

Fig. 6.15 client is ready to send an acknowledgment to the server by sending token A to q_3 and ACK − *received*

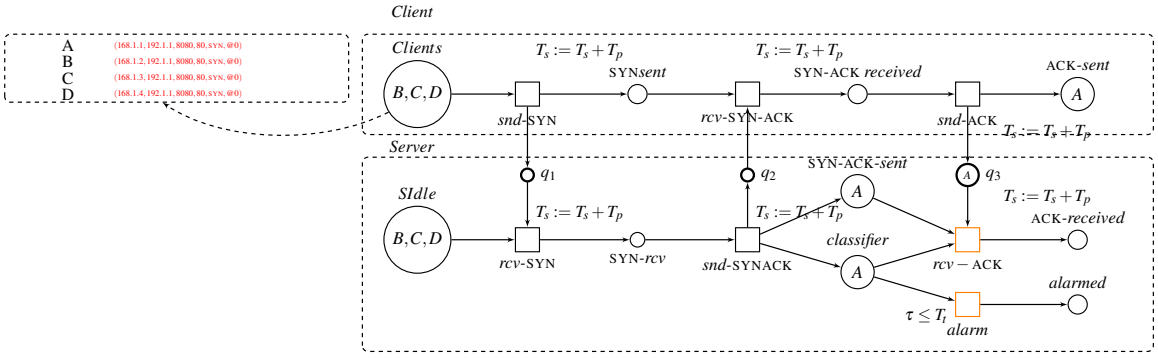

Fig. 6.16 transition *rcv* − ACK is enabled as all its input places are present.

6.9 Experimental Setup

The proposed algorithm aims to enhance the detection of DDoS attacks by integrating the analysis and visualisation of client-server communication. This approach not only focuses on achieving high detection rates but also provides network administrators with detailed insights into network traffic, enabling the identification of potential DDoS attack patterns. In other words, this approach will combine the detection process with the power of modelling which provides more insight for the analysis and detection of such attacks. The experiment is designed to evaluate the effectiveness of our algorithm using the CICIDS 2019 dataset. The performance of the algorithm will be assessed through various metrics, including accuracy, True Positive (TP) rate, True Negative (TN) rate, False Negative (FN) rate, and False Positive (FP) rate.

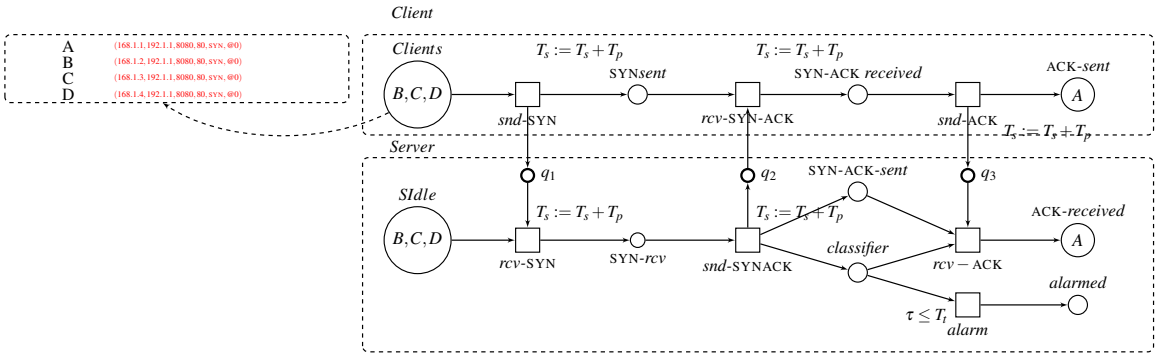

Fig. 6.17 Both side received acknowledgment and data transmission between them will start

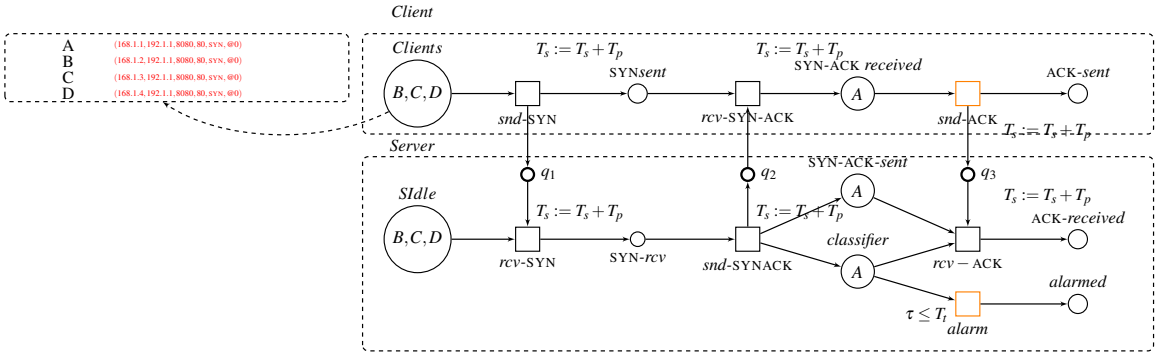

Fig. 6.18 potential attack behaviors occur

Proposed Algorithm

Algorithm 5 is designed for classifying tokens which represent communication packets, into categories of either normal or abnormal based on their adherence to a predefined threshold. Specifically, it processes each token independently using a time threshold τ and a timer T_t that initiates upon a token's arrival at a classifier place. The primary objective is to determine the legitimacy of each token's communication pattern within a specified duration. Upon each token's arrival at the classifier, a timer specific to that token starts, measuring the elapsed time since its classification process began. The algorithm employs a straightforward decision-making criterion to ensure the status of each token: if a token reaches the designated $rcv - \text{ACK}$ transition before its associated timer exceeds the τ threshold, it is deemed normal. This classification implies that the token's communication is within expected parameters, suggesting legitimate traffic. Conversely, tokens failing to meet this condition within the allowed timeframe are classified as abnormal, indicating potential anomalies

or security concerns, such as participation in a SYN-flood attack. This approach enables the effective identification of communication packets that deviate from expected temporal patterns, offering a mechanism for flagging potential threats.

Experimental Procedure

The experiment involves the following steps:

1. **Dataset**: We utilised the CICIDS 2019 dataset [53], which is publicly accessible. This dataset comprises over a million tokens, each representing a packet described by 27 conditional features. These tokens were categorised into two groups: normal and abnormal.

2. **Data Preprocessing**: Handle missing values by applying suitable imputation techniques to ensure the dataset's integrity for accurate analysis.

3. **Hardware and Software Configuration**: The proposed algorithm was evaluated using Python on a Mac PC equipped with an Intel Core i7 CPU and 16 GB of memory. The code for the algorithm is publicly available at [7].

4. **Performance Evaluation**: Assess the algorithm's performance using a confusion matrix to determine accuracy, TP rate, TN rate, FN rate, and FP rate.

6.9.1 Assumptions

In our experiments, we made several key assumptions: The CICIDS 2019 dataset is assumed to be representative of real-world network traffic, including both normal and abnormal packets. While all 27 features provided in the dataset are relevant for distinguishing between normal and abnormal packets, we will specifically rely on six key features, as detailed in Table 6.2, to serve as identifiers for each packet. Additionally, the dataset may have an imbalanced distribution of normal and abnormal packets, which our algorithm needs to handle appropriately.

6.10 Comparative Analysis

Although our algorithm's accuracy of 90% is slightly lower compared to the top-performing methods such as Decision Tree (97.38%) and Random Forest (97.34%) reported in [102], it offers significant advantages in practical applications. One major benefit is its ability to visualise network interactions, aiding network administrators in gaining a deeper understanding of network traffic and identifying potential DDoS attack patterns more effectively. This detailed insight is crucial for timely and efficient mitigation strategies, a feature less emphasised in other methods that primarily focus on detection rates.

Algorithm	**Detection Rate (%)**
Our Proposed Algorithm	**90.00**
K-Nearest Neighbours (KNN) [102]	94.93
Naïve Bayes (NB) [102]	71.41
Decision Tree (DT) [102]	97.38
Random Forest (RF) [102]	97.34

Table 6.3 Comparison of Detection Rates with State-of-the-Art Methods

That is, our approach could assist in understanding and adapting to new attack behaviours without the need for extensive retraining required by traditional machine learning approaches. This adaptability is primarily due to the integration of analysis and visualisation of client-server communication. Moreover, this approach could be improved to enable dynamic pattern recognition, enhance situational awareness, and provide a proactive defence mechanism. For example, by continuously monitoring and visualising network traffic, the approach can identify deviations from normal patterns that may indicate a new type of attack. Specifically, real-time adaptability ensures that network administrators can swiftly understand and respond to emerging threats, providing a robust defence against evolving DDoS scenarios. Future work will aim to validate these capabilities further, as current results primarily demonstrate the efficacy of our method in existing attack scenarios.

6.11 Conclusions

This chapter presented a novel approach for modelling SYN-flood TCP DDoS attacks using TCCSA-nets. The approach includes a detection algorithm that distinguishes between normal and attack communications, with a specific focus on TCP-SYN-flood flags attacks. Our approach extends the advantages of CSA-net's concept through the use of the timing feature, which enables the algorithm to determine whether a packet is normal or not. One of the noteworthy features of this approach is its ability to provide and visualise detailed information on TCP DDoS attacks, which can be utilised further to prevent the attack. The algorithm was tested on publicly available data, and the results were impressive. Specifically, the algorithm achieved a 90% discrimination accuracy between normal and attack communications. Moving forward, we aim to improve this approach by considering additional possible scenarios that cause TCP attacks. Moreover, we plan to enhance this model to operate in real-time situations for real-time detection.

Chapter 7

Conclusions

7.1 Conclusion

This thesis introduced extensions to the CSA-nets framework, which enhance the management and visualisation of complex systems such cyber-crime investigations. Specifically, Chapter 2 provides essential background about the challenges posed by the proliferation of big data and the complexity of CESs. It highlights the need for effective modelling and visualisation techniques to handle vast and intricate data within such systems. The chapter also emphasises the importance of structure and visual representation in simplifying the understanding of complex systems, particularly in the context of cyber-crime modelling and investigation. While mentioning limitations of the existing tools, it elaborates on CSA-nets as a promising formal notation for modelling CESs. Moreover, it outlines both the theoretical research and practical implementation adopted in the thesis, with a focus on improving the visual representation of CSA-nets and their application in cyber-attack detection.

In Chapter 3, we discussed acyclic nets and CSA-nets in detail, explaining their structure and functionality. CSA-nets proved useful in a variety of domains, notably in cyber-crime investigations, due to their capability to model diverse forms of communication within complex systems. Chapter 4 presented innovative approaches to enhancing the comprehensibility and usability of CSA-nets, a modelling framework designed to illustrate the operations of complex systems. These systems are often challenging to understand, given their multifaceted

components and interactions, which can lead to overly complex models. We addressed this issue through two primary strategies. Firstly, we introduced a technique for minimising buffer places caused by high volume of communication between subsystems, by folding them. Secondly, we proposed the use of colours or parameters for tokens to represent distinct types of information within the same framework. These extensions are intended to simplify the visualisation and understanding of these intricate models.

Chapter 5 focused on enhancing the visualisation of CSA-nets by arranging the component acyclic net to reduce the frequency of line crossings. We employed ideas from three straightforward yet effective sorting methods for this purpose. Our evaluations indicated that, selection sort, was particularly effective for positioning the acyclic nets. Additionally, we investigated a splitting strategy to further refine this process. This technique involves dividing the acyclic nets into two groups, addressing each group independently, and then reassembling them.

Chapter 6 introduced a new approach to model and detect SYN-flood TCP DDoS attacks using CSA-nets. The approach focused on modelling SYN-flood attacks with CSA-nets, including a detection algorithm that differentiates between normal and attack communications, with a particular emphasis on TCP-SYN-flood attacks. This method enhances the functionalities of CSA-nets by integrating timing features to assess packet normality. A significant aspect of this approach is its ability to provide detailed insights and visualisation of TCP DDoS attacks, contributing to attack prevention efforts. Experimental outcomes demonstrated a 90% accuracy rate in distinguishing between normal and attack communications.

7.2 Future Work

Following the work presented in this thesis, we see several areas for future research. For Chapter 4, our aim is to explore additional forms of folding, such as integrating entire components of acyclic nets, and to enhance the behaviour of CSA-nets by implementing parameterisation and employing coloured tokens for Behavioural Structured Acyclic Nets (BSA-nets). This methodology is designed to improve the intelligibility of BSA-nets for

facilitating comprehension of the stages within a system's evolution. We plan to continue refining and simplifying these models for dealing with complex and voluminous data, such as those encountered in cyber-security challenges.

In Chapter 5, we explored ways to improve visualisation through sorting algorithms. For future, our efforts will focus on enhancing these techniques by adopting dynamic placement strategies based on specific criteria, such as time intervals or communication events. Moreover, we will further explore the splitting strategy we introduced to find the ideal number of splits that maximises good results and addresses any crossing it could generate.

Focusing on cyber-attack detection, as a continuation of the work carried out in Chapter 6. We will explore extending our research beyond TCP-SYN-flood attacks to encompass a broader spectrum of TCP flags and packet attributes and diversifying the scope of attack techniques to explore other types of attacks. Additionally, we aim to develop a tool for detecting cyber-attacks in real-time, enabling users and investigators to benefit from our detection methodology by analysing and visualising data, helping them understand potential abnormal behaviours.

References

[1] Adoni, W. Y. H., Nahhal, T., Krichen, M., byed, A. E., and Assayad, I. (2020). DHPV: a distributed algorithm for large-scale graph partitioning. *J. Big Data*, 7(1):1–25.

[2] Agostinelli, S., Chiariello, F., Maggi, F. M., Marrella, A., and Patrizi, F. (2023). Process mining meets model learning: Discovering deterministic finite state automata from event logs for business process analysis. *Inf. Syst.*, 114:102180.

[3] Akoglu, L., Tong, H., and Koutra, D. (2014). Graph-based Anomaly Detection and Description: A Survey. *CoRR*, abs/1404.4679.

[4] Al-Duwairi, B., Al-Quraan, E., and AbdelQader, Y. (2020). ISDSDN: Mitigating SYN Flood Attacks in Software Defined Networks. *J. Netw. Syst. Manag.*, 28(4):1366–1390.

[5] Alahmadi, M. (2022). . https://github.com/b7023807/AN_Sorting.git. Accessed: 2023-02-13.

[6] Alahmadi, M. (2023). Parameterised CSA-nets. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Proceedings of the 2023 International Workshop on Petri Nets and Software Engineering (PNSE 2023) co-located with the 44th International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2023), June 27, 2023, Lisbon, Portugal*, volume 3430 of *CEUR Workshop Proceedings*, pages 167–182. CEUR-WS.org.

[7] Alahmadi, M. (2024). https://github.com/b7023807/DDoS. Accessed: 2023-09-24.

[8] Alahmadi, M., Alharbi, S., Alharbi, T., Almutairi, N., Alshammari, T., Bhattacharyya, A., Koutny, M., Li, B., and Randell, B. (2024). Structured Acyclic Nets. *CoRR*, abs/2401.07308.

[9] Alharbi, T. and Koutny, M. (2019). Domain name system (dns) tunnelling detection using structured occurrence nets (sons). In *Proceedings of the International Workshop on Petri Nets and Software Engineering (PNSE 2019)*. Newcastle University.

[10] Alibrahim, T. S. and Hendaoui, S. (2022). DDoS attacks prevention in cloud computing through Transport Control protocol TCP using Round-Trip-time RTT. *IJCSNS*, 22(1):276.

[11] Allen, P. M. (2001). A complex systems approach to learning in adaptive networks. *International Journal of Innovation Management*, 5(02):149–180.

[12] Almutairi, N. (2022). Probabilistic Communication Structured Acyclic Nets. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022*, volume 3170 of *CEUR Workshop Proceedings*, pages 168–187. CEUR-WS.org.

[13] Almutairi, N. and Koutny, M. (2021). Verification of Communication Structured Acyclic Nets Using SAT. In Köhler-Bussmeier, M., Kindler, E., and Rölke, H., editors, *Proceedings of the International Workshop on Petri Nets and Software Engineering 2021 co-located with the 42nd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2021), Paris, France, June 25th, 2021 (due to COVID-19: virtual conference)*, volume 2907 of *CEUR Workshop Proceedings*, pages 175–194. CEUR-WS.org.

[14] Alrehily, A., Fallatah, R., and Thayananthan, V. (2015). Design of vending machine using finite state machine and visual automata simulator. *International Journal of Computer Applications*, 115(18).

[15] Alshammari, T. (2022). Towards Automatic Extraction of Events for SON Modelling. In Köhler-Bussmeier, M., Moldt, D., and Rölke, H., editors, *Petri Nets and Software Engineering 2022 co-located with the 43rd International Conference on Application and Theory of Petri Nets and Concurrency (PETRI NETS 2022), Bergen, Norway, June 20th, 2022*, volume 3170 of *CEUR Workshop Proceedings*, pages 188–201. CEUR-WS.org.

[16] Appiah, O. and Martey, E. M. (2015). Magnetic Bubble Sort Algorithm. *International Journal of Computer Applications*, 122(21).

[17] Arbab, F. (2004). Reo: a channel-based coordination model for component composition. *Math. Struct. Comput. Sci.*, 14(3):329–366.

[18] Aslan, Ö. and Samet, R. (2020). A Comprehensive Review on Malware Detection Approaches. *IEEE Access*, 8:6249–6271.

[19] Aung, H. H. (2019). Analysis and comparative of sorting algorithms. *International Journal of Trend in Scientific Research and Development (IJTSRD)*, 3(5):1049–1053.

[20] Bang-Jensen, J. and Gutin, G. (2018). *Classes of directed graphs*, volume 11. Springer.

[21] Bansal, B., Jenipher, V. N., Jain, R., Dilip, R., Kumbhkar, M., Pramanik, S., Roy, S., and Gupta, A. (2022). Big Data Architecture for Network Security. *Cyber Security and Network Security*, pages 233–267.

[22] Bauer, S., Jaeger, B., Reimann, M., Fromm, J., and Carle, G. (2022). Towards the Classification of TCP Throughput Changes. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE.

[23] Binucci, C., Didimo, W., Kaufmann, M., Liotta, G., and Montecchiani, F. (2022). Placing Arrows in Directed Graph Layouts: Algorithms and Experiments. *Comput. Graph. Forum*, 41(1):364–376.

[24] Bogen, C. and Dampier, D. (2005). Preparing for Large-scale Investigations with Case Domain Modeling. In *Refereed Proceedings of the 5th Annual Digital Forensic Research Workshop, DFRWS 2005, Astor Crowne Plaza, New Orleans, Louisiana, USA, August 17-19, 2005*.

[25] Bruni, R. and Montanari, U. (2000). Zero-safe Nets: Comparing the Collective and Individual Token Approaches. *Inf. Comput.*, 156(1-2):46–89.

[26] Castellani, I. (2001). Process Algebras with Localities. In Bergstra, J. A., Ponse, A., and Smolka, S. A., editors, *Handbook of Process Algebra*, pages 945–1045. North-Holland / Elsevier.

[27] Chao, L., Wu, C., Yoshinaga, T., Bao, W., and Ji, Y. (2021). A Brief Review of Multipath TCP for Vehicular Networks. *Sensors*, 21(8):2793.

[28] Chitturi, B., Balachander, S., Satheesh, S., and Puthiyoppil, K. (2018). Layered Graphs: Applications and Algorithms. *Algorithms*, 11(7):93.

[29] Christensen, S. and Hansen, N. D. (1994). Coloured Petri Nets Extended with Channels for Synchronous Communication. In Valette, R., editor, *Application and Theory of Petri Nets 1994, 15th International Conference, Zaragoza, Spain, June 20-24, 1994, Proceedings*, volume 815 of *Lecture Notes in Computer Science*, pages 159–178. Springer.

[30] Christensen, S. and Mortensen, K. H. (1997). Parametrisation of coloured petri nets. *DAIMI Report Series*, 26(521).

[31] Commoner, F. G., Holt, A. W., Even, S., and Pnueli, A. (1971). Marked Directed Graphs. *J. Comput. Syst. Sci.*, 5(5):511–523.

[32] Corberán, Á., Eglese, R., Hasle, G., Plana, I., and Sanchis, J. M. (2021). Arc routing problems: A review of the past, present, and future. *Networks*, 77(1):88–115.

[33] Dalal, S., Lilhore, U. K., Foujdar, N., Simaiya, S., Ayadi, M., Almujally, N. A., and Ksibi, A. (2023). Next-generation cyber attack prediction for IoT systems: leveraging multi-class SVM and optimized CHAID decision tree. *J. Cloud Comput.*, 12(1):137.

[34] Das, R. and Soylu, M. (2023). A key review on graph data science: The power of graphs in scientific studies. *Chemometrics and Intelligent Laboratory Systems*, page 104896.

[35] Dastres, R. and Soori, M. (2020). Secure socket layer (SSL) in the network and web security. *International Journal of Computer and Information Engineering*, 14(10):330–333.

[36] DeCusatis, C. M., Lynch, R. M., Kluge, W., Houston, J., Wojciak, P. A., and Guendert, S. (2020). Impact of Cyberattacks on Precision Time Protocol. *IEEE Trans. Instrum. Meas.*, 69(5):2172–2181.

[37] Döhmen, T., Bruntink, M., Ceolin, D., and Visser, J. (2016). Towards a Benchmark for the Maintainability Evolution of Industrial Software Systems. In Heidrich, J. and Vogelezang, F. W., editors, *2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IWSM-MENSURA 2016, Berlin, Germany, October 5-7, 2016*, pages 11–21. IEEE Computer Society.

[38] Domrös, S. and von Hanxleden, R. (2022). Preserving Order during Crossing Minimization in Sugiyama Layouts. In Hurter, C., Purchase, H. C., and Bouatouch, K., editors, *Proceedings of the 17th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2022, Volume 3: IVAPP, Online Streaming, February 6-8, 2022*, pages 156–163. SCITEPRESS.

[39] Dupont, B. (2019). Enhancing the effectiveness of cybercrime prevention through policy monitoring. *Journal of crime and justice*, 42(5):500–515.

[40] Dutta, N., Jadav, N., Tanwar, S., Sarma, H. K. D., Pricop, E., Dutta, N., Jadav, N., Tanwar, S., Sarma, H. K. D., and Pricop, E. (2022). Introduction to cybersecurity. *Cyber Security: Issues and Current Trends*, pages 1–16.

[41] Eades, P. and de Mendonça Neto, C. F. X. (1995). Vertex Splitting and Tension-free Layout. In Brandenburg, F., editor, *Graph Drawing, Symposium on Graph Drawing, GD '95, Passau, Germany, September 20-22, 1995, Proceedings*, volume 1027 of *Lecture Notes in Computer Science*, pages 202–211. Springer.

[42] Ekowati, M. A. S., Nindyatama, Z. P., Widianto, W., and Dananti, K. (2022). Comparative Analysis of the Speed of the Sorting Method on Google Translate Indonesian-english Using Binary Search. *International Journal of Global Operations Research*, 3(3):108–115.

[43] Ermel, C. and Weber, M. (2001). Implementation of Parameterized Net Classes with the Petri Net Kernel of the "Petrinetz-Baukasten". *Unifying Petri Nets: Advances in Petri Nets*, pages 79–102.

[44] Farah, K., Chabir, K., and Abdelkrim, M. N. (2023). High level Petri nets-based proposal of an integrated intrusion detection and prevention mechanism in network controlled systems. *IET Commun.*, 17(4):469–477.

[45] Fink, M. and Pupyrev, S. (2013). Metro-line Crossing Minimization: Hardness, Approximations, and Tractable Cases. *CoRR*, abs/1306.2079.

[46] Furat, F. G. (2016). A comparative Study of Selection Sort and Insertion Sort Algorithms. *International Research Journal of Engineering and Technology (IRJET)*, 3(12):326–330.

[47] Ganapathi, P. and Chowdhury, R. (2022). Parallel Divide-and-conquer Algorithms for Bubble Sort, Selection Sort and Insertion Sort. *Comput. J.*, 65(10):2709–2719.

[48] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., March, M., and Lempitsky, V. (2016). Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35.

[49] Gitelman, L., Ryzhuk, O., and Kozhevnikov, M. (2019). Visual analysis for conceptual design of complex systems. *Management and applications of complex systems*.

[50] Giua, A. and Suárez, M. S. (2018). Petri nets and Automatic Control: A historical perspective. *Annu. Rev. Control.*, 45:223–239.

[51] Gokhale, S., Siddavatam, I. A., Dalvi, A., Shaikh, M., and Patil, S. (2022). Formal modelling and verification of high interactive honeypot using coloured Petri nets. *Int. J. Crit. Comput. Based Syst.*, 10(3):227–247.

[52] Gracanin, D., Srinivasan, P., and Valavanis, K. P. (1994). Parameterized Petri Nets and Their Application to Planning and Coordination in Intelligent Systems. *IEEE Trans. Syst. Man Cybern. Syst.*, 24(10):1483–1497.

[53] Hamor, T. (2021). CICIDS 2019 Dataset.

[54] Hayfron-Acquah, J., Appiah, O., and Riverson, K. (2015). Improved selection sort algorithm. *International Journal of Computer Applications*, 110(5).

[55] Heiding, F., Katsikeas, S., and Lagerström, R. (2023). Research communities in cyber security vulnerability assessments: A comprehensive literature review. *Comput. Sci. Rev.*, 48:100551.

[56] Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall.

[57] Hughes, D. T. (2015). SmartFiles–ICT Innovation in Complex Criminal Investigations. *International Journal of Innovative Computing*, 5(1).

[58] Jangjou, M. and Sohrabi, M. K. (2022). A comprehensive survey on security challenges in different network layers in cloud computing. *Archives of Computational Methods in Engineering*, pages 1–22.

[59] Jensen, K. and Kristensen, L. M. (2009). *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer.

[60] Jensen, K., Kristensen, L. M., and Wells, L. (2007). Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *Int. J. Softw. Tools Technol. Transf.*, 9(3-4):213–254.

[61] Jianu, R., Rusu, A., Fabian, A. J., and Laidlaw, D. H. (2009). A Coloring Solution to the Edge Crossing Problem. In Banissi, E., Stuart, L. J., Wyeld, T. G., Jern, M., Andrienko, G. L., Memon, N., Alhajj, R., Burkhard, R. A., Grinstein, G. G., Groth, D. P., Ursyn, A., Johansson, J., Forsell, C., Cvek, U., Trutschl, M., Marchese, F. T., Maple, C., Cowell, A. J., and Moere, A. V., editors, *13th International Conference on Information Visualisation, IV 2009, 15-17 July 2009, Barcelona, Spain*, pages 691–696. IEEE Computer Society.

[62] Kalaivani, A. and Swetha, K. (2021). An enhanced bidirectional insertion sort over classical insertion sort. *International journal of image and graphics*, 21(02):2150024.

[63] Katheder, J., Kobourov, S. G., Kuckuk, A., Pfister, M., and Zink, J. (2023). Simultaneous Drawing of Layered Trees. *CoRR*, abs/2302.11952.

[64] Kautish, S., Reyana, A., and Vidyarthi, A. (2022). SDMTA: Attack Detection and Mitigation Mechanism for DDoS Vulnerabilities in Hybrid Cloud Environment. *IEEE Transactions on Industrial Informatics*.

[65] Kavitha, K., Babitha, T., Praveena, V., and Devika, P. (2022). Identifying legitimate user in DDoS attack using Petri net. *Materials Today: Proceedings*.

[66] Khan, N., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Mahmoud Ali, W. K., Alam, M., Shiraz, M., Gani, A., et al. (2014). Big data: survey, technologies, opportunities, and challenges. *The scientific world journal*, 2014.

[67] Kleijn, J. and Koutny, M. (2011). Causality in Structured Occurrence Nets. In Jones, C. B. and Lloyd, J. L., editors, *Dependable and Historic Computing - Essays Dedicated to Brian Randell on the Occasion of His 75th Birthday*, volume 6875 of *Lecture Notes in Computer Science*, pages 283–297. Springer.

[68] Köhler-Bußmeier, M. and Kudlek, M. (2008). Linear Properties of Zero-safe Nets with Debit Tokens. *Fundam. Informaticae*, 85(1-4):329–342.

[69] Koutny, M. and Randell, B. (2009a). Structured Occurrence Nets: A Formalism for Aiding System Failure Prevention and Analysis Techniques. *Fundam. Informaticae*, 97(1-2):41–91.

[70] Koutny, M. and Randell, B. (2009b). Structured Occurrence Nets: A Formalism for Aiding System Failure Prevention and Analysis Techniques. *Fundam. Informaticae*, 97(1-2):41–91.

[71] Koutny, M. and Randell, B. (2009c). Structured occurrence nets: incomplete, contradictory and uncertain failure evidence. Technical report, School of Computing Science, Newcastle University.

[72] Kristo, A., Vaidya, K., Çetintemel, U., Misra, S., and Kraska, T. (2020). The Case for a Learned Sorting Algorithm. In Maier, D., Pottinger, R., Doan, A., Tan, W., Alawini, A., and Ngo, H. Q., editors, *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, pages 1001–1016. ACM.

[73] Kumari, P. and Jain, A. K. (2023). A Comprehensive Study of DDoS Attacks over IoT Network and Their Countermeasures. *Computers & Security*, page 103096.

[74] Kummer, O. (1999). A Petri Net View on Synchronous Channels. *Petri Net Newsletter*, 56:7–11.

[75] Kuntz, P., Pinaud, B., and Lehn, R. (2006). Minimizing crossings in hierarchical digraphs with a hybridized genetic algorithm. *J. Heuristics*, 12(1-2):23–36.

[76] Lent, D. M. B., Novaes, M. P., Carvalho, L. F., Lloret, J., Rodrigues, J. J., and Proença, M. L. (2022). A gated recurrent unit deep learning model to detect and mitigate distributed denial of service and portscan attacks. *IEEE Access*, 10:73229–73242.

[77] Lin, H.-D. and Messerschmitt, D. G. (1991). Finite state machine has unlimited concurrency. *IEEE Transactions on Circuits and Systems*, 38(5):465–475.

[78] Liu, F., Heiner, M., and Gilbert, D. R. (2019). Coloured Petri nets for multilevel, multiscale and multidimensional modelling of biological systems. *Briefings Bioinform.*, 20(3):877–886.

[79] Liu, S., Cui, W., Wu, Y., and Liu, M. (2014). A survey on information visualization: recent advances and challenges. *Vis. Comput.*, 30(12):1373–1393.

[80] Lobo, J. and Kuwelkar, S. (2020). Performance analysis of merge sort algorithms. In *2020 International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 110–115. IEEE.

[81] Madera-Ramírez, F. A., Trejo-Sánchez, J. A., López-Martínez, J. L., and Ríos-Martínez, J. (2023). Crossing edge minimization in radial outerplanar layered graphs using segment paths. *Optim. Methods Softw.*, 38(6):1142–1162.

[82] Martí, R., Campos, V., Hoff, A., and Peiró, J. (2018). Heu CD ristics for the min–max arc crossing problem in graphs. *Expert Systems with Applications*, 109:100–113.

[83] Maylawati, D., Darmalaksana, W., and Ramdhani, M. A. (2018). Systematic design of expert system using unified modelling language. In *IOP Conference Series: Materials Science and Engineering*, volume 288, page 012047. IOP Publishing.

[84] McGraw, G. (2012). Software Security - Building Security In. *Datenschutz und Datensicherheit*, 36(9):662–665.

[85] Milner, R. (1989). *Communication and concurrency*. PHI Series in computer science. Prentice Hall.

[86] Mishra, A. D. and Garg, D. (2008). Selection of best sorting algorithm. *International Journal of intelligent information Processing*, 2(2):363–368.

[87] Moere, A. V. and Purchase, H. C. (2011). On the role of design in information visualization. *Inf. Vis.*, 10(4):356–371.

[88] Mohammed, A. S., Amrahov, S. E., and Celebi, F. V. (2016). Bidirectional Conditional Insertion Sort algorithm; An efficient progress on the classical insertion sort. *CoRR*, abs/1608.02615.

[89] Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proc. IEEE*, 77(4):541–580.

[90] Narasimhan, S., El-Farra, N. H., and Ellis, M. J. (2022). Active multiplicative cyber-attack detection utilizing controller switching for process systems. *Journal of Process Control*, 116:64–79.

[91] Nashat, D. and Hussain, F. A. (2021). Multifractal detrended fluctuation analysis based detection for SYN flooding attack. *Comput. Secur.*, 107:102315.

[92] Nickel, S., Nöllenburg, M., Sorge, M., Villedieu, A., Wu, H., and Wulms, J. (2022). Planarizing Graphs and their Drawings by Vertex Splitting. *CoRR*, abs/2202.12293.

[93] Nöllenburg, M. (2020). Crossing Layout in Non-planar Graph Drawings. In Hong, S. and Tokuyama, T., editors, *Beyond Planar Graphs, Communications of NII Shonan Meetings*, pages 187–209. Springer.

[94] Nuiaa, R. R., Manickam, S., and Alsaeedi, A. H. (2021). Distributed reflection denial of service attack: A critical review. *International Journal of Electrical and Computer Engineering*, 11(6):5327.

[95] Oatley, G. C. (2022). Themes in data mining, big data, and crime analytics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(2):e1432.

[96] Pasandideh, S., Pereira, P., and Gomes, L. (2022). Attack Tree Refinements Analysis and Verification by applying Coloured Petri Nets. In *IECON 2022 - 48th Annual Conference of the IEEE Industrial Electronics Society, Brussels, Belgium, October 17-20, 2022*, pages 1–6. IEEE.

[97] Pei, J., Chen, Y., and Ji, W. (2019). A DDoS attack detection method based on machine learning. In *Journal of Physics: Conference Series*, volume 1237, page 032040. IOP Publishing.

[98] Pozár, J. (2014). Modelling of the Investigation of Cybercrime. *Science & Military Journal*, 9(1):63.

[99] Prayudi, Y., Ashari, A., and Priyambodo, T. K. (2015). A proposed digital forensics business model to support cybercrime investigation in Indonesia. *International Journal of Computer Network and Information Security*, 7(11):1–8.

[100] Purchase, H. (1997). Which aesthetic has the greatest effect on human understanding? In *International Symposium on Graph Drawing*, pages 248–261. Springer.

[101] Qu, Z., Shi, H., Wang, Y., Yin, G., and Abu-Siada, A. (2022). Active and Passive Defense Strategies of Cyber-physical Power System against Cyber Attacks Considering Node Vulnerability. *Processes*, 10(7):1351.

[102] Rajesh, S., Clement, M., SB, S., SH, A. S., and Johnson, J. (2021). Real-time DDoS Attack Detection Based on Machine Learning Algorithms. *Available at SSRN 3974241*.

[103] Ramkumar, B. and Subbulakshmi, T. (2021). Tcp Syn Flood Attack Detection and Prevention System using Adaptive Thresholding Method. In *ITM Web of Conferences*, volume 37, page 01016. EDP Sciences.

[104] Randell, B. (2011). Occurrence Nets Then and Now: The Path to Structured Occurrence Nets. In Kristensen, L. M. and Petrucci, L., editors, *Applications and Theory of Petri Nets - 32nd International Conference, PETRI NETS 2011, Newcastle, UK, June 20-24, 2011. Proceedings*, volume 6709 of *Lecture Notes in Computer Science*, pages 1–16. Springer.

[105] Randell, B. and Koutny, M. (2007). Failures: Their Definition, Modelling and Analysis. In Jones, C. B., Liu, Z., and Woodcock, J., editors, *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, Macau, China, September 26-28, 2007, Proceedings*, volume 4711 of *Lecture Notes in Computer Science*, pages 260–274. Springer.

[106] Randell, B. and Koutny, M. (2009). Structured Occurrence Nets: Incomplete, contradictory and uncertain failure evidence. *School of Computing Science Technical Report Series*.

[107] Salihu, A., Hoti, M., and Hoti, A. (2022). A Review of Performance and Complexity on Sorting Algorithms. In *2022 International Conference on Computing, Networking, Telecommunications & Engineering Sciences Applications (CoNTESA)*, pages 45–50. IEEE.

[108] Sari, R. F., Rosyidi, L., Susilo, B., and Asvial, M. (2021). A Comprehensive Review on Network Protocol Design for Autonomic Internet of Things. *Inf.*, 12(8):292.

[109] Schneier, B. (2007). *Applied cryptography: protocols, algorithms, and source code in C*. john wiley & sons.

[110] Scholten, J. G., Arbab, F., de Boer, F. S., and Bonsangue, M. M. (2006). A Component Coordination Model Based on Mobile Channels. *Fundam. Informaticae*, 73(4):561–582.

[111] Shaaban, A. R., Abdelwaness, E., and Hussein, M. (2019). TCP and HTTP Flood DDOS Attack Analysis and Detection for space ground Network. In *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6. IEEE.

[112] Sheyner, O., Haines, J. W., Jha, S., Lippmann, R., and Wing, J. M. (2002). Automated Generation and Analysis of Attack Graphs. In *2002 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 12-15, 2002*, pages 273–284. IEEE Computer Society.

[113] Sindhu, G., Thanusha, K., Ganesh, G., Reddy, K., and Pujitha, M. (2023). Design of Vending Machine Using Visual Automata Simulator & Finite State Machine. In *2023 International Conference on Communication, Circuits, and Systems (IC3S)*, pages 1–7. IEEE.

[114] Singha, M. F. and Patgiri, R. (2023). A comprehensive investigation on attack graphs. In *Advances in Computers*, volume 128, pages 251–272. Elsevier.

[115] Sodhi, T. S., Kaur, S., and Kaur, S. (2013). Enhanced insertion sort algorithm. *International journal of Computer applications*, 64(21).

[116] Stallings, W., Brown, L., Bauer, M. D., and Howard, M. (2012). *Computer security: principles and practice*, volume 3. Pearson Upper Saddle River.

[117] Stewart, R., Tüxen, M., and Nielsen, K. (2022). RFC 9260: Stream Control Transmission Protocol.

[118] Sugiyama, K., Tagawa, S., and Toda, M. (1981). Methods for Visual Understanding of Hierarchical System Structures. *IEEE Trans. Syst. Man Cybern.*, 11(2):109–125.

[119] Sumathi, S. and Rajesh, R. (2021). Comparative study on TCP-SYN-flood DDoS attack detection: A machine learning algorithm based approach. *WSEAS Transactions on Systems and Control*, 16:584–591.

[120] Sutopo, H. (2011). Selection sorting algorithm visualization using flash. *The International Journal of Multimedia & Its Applications (IJMA)*, 3(1):22–35.

[121] Thabit, K. (2022). An Integer Prefix-based Methodology for Enhancing the Execution Performance of Any String Sorting Algorithm. In *Proceedings of the Future Technologies Conference (FTC) 2021, Volume 3*, pages 518–524. Springer.

[122] Thankappan, M., Rifà-Pous, H., and Garrigues, C. (2022). Multi-channel Man-in-the-middle Attacks Against Protected Wi-fi Networks: A State of the Art Review. *CoRR*, abs/2203.00579.

[123] Thomas, P. (2023). Enhanced Efficiency in Sorting: Unveiling the Optimized Bubble Sort Algorithm. *J Robot Auto Res*, 4(3):424–428.

[124] Tonkal, Ö., Polat, H., Başaran, E., Cömert, Z., and Kocaoğlu, R. (2021). Machine learning approach equipped with neighbourhood component analysis for DDoS attack detection in software-defined networking. *Electronics*, 10(11):1227.

[125] Toyeer-E-Ferdoush, Rahman, H., and Hasan, M. (2022). A convenient way to mitigate DDoS TCP-SYN-flood attack. *Journal of Discrete Mathematical Sciences and Cryptography*, 25(7):2069–2077.

[126] Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., Al-Nemrat, A., and Venkatraman, S. (2019). Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access*, 7:41525–41550.

[127] Walter, J., Zink, J., Baumeister, J., and Wolff, A. (2020). Layered Drawing of Undirected Graphs with Generalized Port Constraints. *CoRR*, abs/2008.10583.

[128] Whitman, M. E. and Mattord, H. J. (2021). *Principles of information security*. Cengage learning.

[129] Wu, X., Xiong, C., Deng, N., and Xia, D. (2021). A variable depth neighborhood search algorithm for the Min-max Arc Crossing Problem. *Comput. Oper. Res.*, 134:105403.

[130] Xiong, W., Legrand, E., Åberg, O., and Lagerström, R. (2022). Cyber security threat modeling based on the mitre enterprise att&ck matrix. *Software and Systems Modeling*, 21(1):157–177.

[131] Yankson, B., Iqbal, F., and AlMaeeni, F. (2023). Social Robots Privacy Enhancement Using Colored Petri Net (CPN) for Behavior Modeling: A Case Study of Asus Zenbo Robot. In *International Conference on Cyber Warfare and Security*, volume 18, pages 457–464.

[132] Yerpude, P. (2020). Predictive Modelling of Crime Data Set Using Data Mining. *International Journal of Data Mining & Knowledge Management Process (IJDKP) Vol*, 7.

[133] Zeebaree, S. R., Jacksi, K., and Zebari, R. R. (2020). Impact analysis of SYN flood DDoS attack on HAProxy and NLB cluster-based web servers. *Indones. J. Electr. Eng. Comput. Sci*, 19(1):510–517.

[134] Zhao, J., Shetty, S., Pan, J. W., Kamhoua, C. A., and Kwiat, K. A. (2019). Transfer learning for detecting unknown network attacks. *EURASIP J. Inf. Secur.*, 2019:1.