Towards Content-Centric Control Plane Supporting Efficient Anomaly Detection Functions

Hoang Long Mai*†, Guillaume Doyen*, Wissam Mallouli†, Edgardo Montes de Oca†, Olivier Festor‡
*ICD - CNRS UMR 6281, Troyes University of Technology, 10004 Troyes - France
†Montimage, 39 rue Bobillot, 75013 Paris - France
‡LORIA - CNRS UMR 7503, TELECOM Nancy - University of Lorraine, 54506 Vandoeuvre-les-Nancy - France

***Abstract*—Anomaly detection remains a challenging task due to both the ever more complex functions that need to be executed and the evolution of current networking devices which induces limitation of computational resources such as the Internet of Things (IoT). Furthermore, results of anomaly function computations can be repeated gradually over time or executed in neighboring nodes, thus leading to a waste of such limited computing resources in constrained nodes. To tackle these issues, the content-centric paradigm enhanced with computing features offers a promising solution to reduce the computation resources and finally improve the scalability of anomaly detection functions. In this paper, we propose a first step toward a content-oriented control plane which enables the distribution of the processing and the sharing of results of anomaly detection functions in the network. We present the way we leverage NFN to support Bayesian Network inference to detect anomalies in network traffic. The relevance and performance of our proposed approach are demonstrated by considering the Content Poisoning Attack (CPA) through numerous experiment data.**

***Index Terms*—Distributed anomaly detection, Bayesian Network, Named Function Networking**

I. Introduction

Strong security is a prerequisite for any network today. With the evolution of technology, usage, and markets, a large number of types of attacks have emerged. The operational methods of such attacks have highly gained in sophistication, making them increasingly difficult to detect. Thanks to the rise of Machine Learning in recent years, new means for detection of attacks have been proposed. The deployment and operation of these functions grow in complexity as well as the number of metrics used to feed them. Besides, the trend of IoT is also remarkable, and the security for billions of devices with limited computational resources remains a challenge increased by the growing complexity of the processing functions. This evolution calls for novel approaches to effectively detect attacks using complex security functions in an environment of distributed and limited computational resource devices.

In the context of distributed system security, we found out that current node-centric solutions induce a waste of computing resources which further enlarge the gap between the complexity of security functions and the constraint resources of nodes. Especially, it appears that (1) each node executes identical security functions over time, especially in a normal condition without attack or a substantial change, and (2) results of security functions are likely to have already been calculated in remote nodes. Hence, the computation of security functions producing identical results are repeated gradually over time. We verified this hypothesis by finding out that in a normal condition of Named Data Networking (NDN) [1], 87% of computational security operations are repeated.

Inspired by the concept of both active networking and the content-centric communication model [2], the Named Function Networking (NFN) architecture has been introduced in [3], [4] to resolve names to computation functions. NFN also leverages the in-network caching feature of the content-centric paradigm which offers a substantial way to reduce the computation resources of security functions. In this paper we explore the opportunity of leveraging NFN as an execution environment for anomaly detection. This stands for a first attempt toward the design of content-centric control planes. As an anomaly detection framework, we consider Bayesian Network (BN) inference since it stands for a representative function that numerous security components considers. As a use-case, we have chosen an acknowledged scenario [5] which allows us to solely address computation issues: the Content Poisoning Attack (CPA) detection performed in a NDN environment.

The paper is organized as follows. Section II presents some background and the works related to distributed security function execution. Afterward, Section III provides background on Bayesian Networks. Section IV presents our main contribution which leverages NDN and consists in a naming scheme, data structures, and the proposal of concent-centric inference functions. In Section V, we provide numerical results that demonstrate our approach's relevance and performance in the context of CPA detection. Finally, section VI summarizes the paper and presents our plans for future work.

II. Related works

This section surveys the main previous works which are close to ours. These are distributed anomaly detection and NFN.

A. Distributed anomaly detection

The authors in [6] propose a novel approach based on a distributed, cluster-based anomaly detection algorithm. The proposed approach clusters the sensor measurements and merges clusters before sending a description of the clusters to the other nodes to minimize the communication overhead. Diro et al. in [7] propose a novel approach using deep learning for cyber-attack detection in IoT using a fog ecosystem. A

978-3-903176-24-9

distributed attack detection also proposed and evaluated at fog level to prove its effectiveness compared to the centralized one. Palmeri et al. [8] proposed an approach through which network anomaly can be detected by multiple distributed sensors located throughout the network. Then, they use Independent Component Analysis to collect necessary components to build the baseline traffic profiles needed, which are then classified by machine learning inferred decision trees to detect abnormal behaviors. In summary, several approaches are proposed in the research community to resolve the challenge of building a distributed framework for attack and anomaly detection. However, none of these proposals leverage the distributed systems to share the result of the detection function results and thus avoid computation overlaps.

B. Named Function Networking

In recent years, researches have been interested in the content-centric communication model. Among them, content centric networks (CCNs) [2] is a proposal which received much attention from the community. In CCN, requests for content are routed by name instead of requiring the use of host IP address. The content can also be cached in intermediate nodes on the path from the content producer to its consumer in order to fasten subsequent requests sharing part of this path. The Named Function Networking (NFN) architecture, introduced in [3], [4], further extends this concept to resolve names to computation functions. In NFN, instead of requesting data, a client sends an Interest with the function expression to request the result of a function. The data returned to clients are the computed results, which also can be cached as a Data packet. The functional programs are expressed in λ-calculus [9], encoded in hierarchical CCN names that can be aggregated and carried in $Interest$ packets. A λ-calculus expression resolution engine is integrated into each NFN node, and optionally an application processing or the compute server can also be hosted in an NFN node. The routers use the longest-match name lookup to forward $Interests$. If an NFN node hosts a compute server, its named function is published as a λ-calculus expression in the network, and $Interests$ can be forwarded to the function host. If a copy of a function result is cached at an intermediate node, it will be returned to the requesting user along the reverse path. Otherwise, one of the NFN nodes, which has received the $Interest$ on the path to the data source, will attempt to compute the result based on the policies and the available processing resources.

A typical use of NFN consists in performing data processing in the network. In [10], [11], the authors use NFN for the orchestration of treatments on multimedia content and for the processing of data streams in the network. NFN has also been proposed as a query language [12] to improve the naming scheme for content in Information-Centric Networking (ICN). An In-Network Access Control for NDN is proposed in [13], [14] which also utilizes NFN as Data Processing Unit (DPU) between clients and providers.

NFN has also found its application in an Edge/Fog computing context. The function program can be stored in cache of nodes and migrated to the network according to the request of the users [15]. A study on NFN as an architecture for Mobile Edge Computing (MEC) systems is proposed in [16]. It shows how to identify network resources and to support the execution of their functions [17]. The authors in [18] have also explored the possibilities of an augmented reality network using NDN with NFN as an associated processing mechanism. In [19], authors propose another ICN-based service platform at the edge of the network with NFN as a protocol for managing service interactions.

In the context of the Internet of Things, several solutions have been proposed to exploit the mechanisms of NFN. The authors of [20] propose a management of computing services to assign and schedule computational tasks in an IoT network. In [21], the authors also use NFN in an IoT network where sensor data and processing functions are stored. In [22], the authors also propose an architecture and a naming scheme to cache data and processing within an IoT network. The authors of [23] also use NFN as the DPU for the IoT coupled to NDN. In [24], Lenord et al. use NFN in an IoT gateway to propose a new routing strategy in NDN.

In brief, NFN has drawn much attention from the research community in recent years, with different use-cases, in the context of NDN, IoT, and Fog computing and in the following, we explore the opportunity to consider it in a security context, for anomaly detection. We especially consider Bayesian Network inference which is commonly used as a substrate.

III. Background on Bayesian Networks

To ease the understanding of the methods integrated in our solution, we present in this section the necessary elements related to Bayesian Networks, classifiers and inference.

A. Terminology

We first explain definitions of terms used in the inference algorithm presented subsquently.

1) Variable: A random variable, called X_i, is a set of possible values of a random phenomenon. In the following, we only consider discrete ones. In the context of anomaly detection, a random variable can be an observed metric or an anomaly detection result.

2) Evidence: An $evidence\ E = e$ is a subset $E = (X_{e_1}, \ldots, X_{e_m})$ of random variables, where m is denoted as the number of variables in the $evidence$, standing for the observed phenomenon and an instantiation $e = (x_{e_1}, \ldots, x_{e_m})$ of these variables is an occurrence of these observed data.

3) Factor: To quantify the affinity of two random variables, a general-purpose function called a $factor$ is considered. Let $(X_1 \ldots X_n)$ be a set of random variables, a $factor\ \phi$ is defined as a function from $Val(X_1, \ldots, X_n)$ to $\mathbb{R}$. The set of variables $(X_1, \ldots, X_n)$ is called the scope of the $factor$ and denoted $Scope[\phi]$ [25]. An entry is a set of values of each variable and the related value of ϕ, which represents the affinity between values. The higher the value of ϕ, the more compatible these values are. A Conditional Probabilities

TABLE I
A FACTOR EXAMPLE

A	B	C	$\phi(A,B,C)$
1	1	1	0.25
1	1	2	0.35
1	2	1	0.08
1	2	2	0.16
2	1	1	0.05
2	1	2	0.07
2	2	1	0
2	2	2	0
3	1	1	0.15
3	1	2	0.21
3	2	1	0.09
3	2	2	0.18

Distribution (CPD) can be considered as a sort of $factor$ which is normalized.

Table I illustrates an example of a $factor$. The first three columns describe the overall joint distributions of three random variables A, B and C, while the last column holds the values of ϕ which correspond to each entry in the joint distribution of the $factor$. Each line in Table I represents an entry of the $factor$ $\phi(A,B,C)$, while the value of a $factor$ is the affinity between values in the entry. $\phi(A=1,B=2,C=2)=0.16$ while $\phi(A=1,B=2,C=1)=0.08$ means that $A=1,B=2,C=2$ is likely to be twice as compatible in comparison to the case where $A=1,B=2,C=1$.

B. Core Principles

A Bayesian Network (BN) [25] is a probabilistic graphical model that consists of nodes and directed edges. Each node represents a random variable X_i and an edge from node X_i to node X_j represents a statistical conditional dependence between the corresponding variables. As such, X_i is called a parent of X_j (i.e. $X_i \in Pa(X_j)$) and X_j is called a child of X_i. The relationship between variables is defined by the CPDs $\mathbb{P}[X_j|X_i]$ and the prior distribution of parent X_j. A Bayesian Network Classifier (BNC) is a BN used to classify one of its nodes in a finite set of values. In other words, BNC is a BN with a set of discrete random variables $X=(X_1,\ldots,X_n)$ and a set of observed data $E=e$ called $evidence$, a query variable X_q, where one needs to calculate the conditional probability of $P(X_q|E=e)$. This means that the conditional probability of $P(X_q|E=e)$ is the sum of all possible combinations of values of the other variables $X_i \in X - X_q$ of the joint probability of all values X, knowing $E=e$.

C. Bayesian Inference

The *inference* designates an algorithm which consists in calculating, for each value $x_q \in Val(X_q)$, the joint distribution probability $P(X_1,...,X_n)$ and then to sum out the instantiations that are consistent with $X_q = x_q$:

$$P(X_q) = \sum_{X_1} \cdots \sum_{X_i \neq X_q} \cdots \sum_{X_n} P(X_1,\ldots,X_n) \qquad (1)$$

Thanks to Bayes' rule, the joint probability distribution can be expressed as follows:

$$P(X_1,...,X_n) = \prod_{i=1}^{n} P(X_i|Pa(X_i)) \qquad (2)$$

which can be expressed in form of $factors$:

$$P(X_1,...,X_n) = \frac{1}{Z}\prod_{i=1}^{n} \phi_i(X_i, Pa(X_i)) \qquad (3)$$

$$where \quad Z = \sum_{X_1,\ldots,X_n} \prod_{i=1}^{n} \phi_i(X_i, Pa(X_i)) \qquad (4)$$

Φ is denoted as the set of $factors$ in a given BN. Hence, from (1) and (3), the goal of the inference algorithm is to compute:

$$P(X_q) = \frac{1}{Z}\sum_{X_n} \phi_q \cdot (\ldots(\sum_{X_2} \phi_3 \cdot (\sum_{X_1} \phi_2 \cdot \phi_1))) \qquad (5)$$

As we can see, the key of the inference algorithm is to compute the following expression $\sum \prod_{\phi\in\Phi} \phi$. This operation is called the sum-product procedure. Calculating this expression requires significant computational resources. Therefore, to calculate it effectively, we must perform the product in a subset of $factors$.

D. Variable Elimination Algorithm

There are various algorithms able to perform the inference of a BN. The Variable Elimination (VE) algorithm, as illustrated in pseudo-code 1, which calculates some subexpressions and caches the result of intermediate computations to avoid generating an exponentially high number of computation steps, is the most basic one. We use this algorithm to demonstrate the benefit of caching and distribution of results in NFN. More specifically, the input to the VE algorithm consists of two parts: the $factors$ and the $evidence$. The $factors$ are established by CPDs and are not changed when the algorithm performs the inference at different times, while the $evidence$ $E=e$ is retrieved after each iteration of the execution. The output is the conditional probability $P(X_q|E=e)$.

Given the observed data ($evidence$) $E=e$, we perform a *factor reduction* to reduce the complexity of each $factor$ in the set of $factors$ Φ by removing the joint distributions that are not matching up the $evidence$ (line 1-3). Several variables remain in the query probability after the *factor reduction* as we may observe only a subset of variables $E \subset X$. These variables will be eliminated by the sum-out procedure. For this purpose, the next step consists in eliminating these variables according to an Elimination Order (line 4). For each variable in the Elimination Order, we first multiply all the $factors$ that include this variable, generating a product $factor$ (line 6-10). Then, we sum up the value of the variable and eliminate it out of this combined $factor$, generating a new $factor$ that we enter into our set of $factors$ to be processed (line 11). Afterward, when all variables have been eliminated, the only variable which remains is the query variable X_q. At this moment, we

perform once again a *factor product* to get the final $factor$ over the distribution of X_q (line 13-15). Finally, we normalize the $factor$ by dividing each value by their sum (line 16-17).

Input: *initial factors* (Φ) and $evidence$ (E=e)
Output: Conditional probability $P(X_q|E=e)$

1 **foreach** $\phi_i \in \Phi$ **do**
2 | $\phi_i \leftarrow \phi_i(E=e)$ `// Factor reduction`
3 **end**
4 Select Elimination Order (σ);
5 **foreach** $x_i \in \sigma$ **do**
6 | **foreach** $\phi_j \in \Phi$ **do**
7 | | **if** $x_i \in Scope[\phi_j]$ **then**
8 | | | $\psi_i \leftarrow \psi_i * \phi_j$ `// Factor product`
9 | | **end**
10 | **end**
11 | $\phi_i \leftarrow \sum_{X_i} \psi_i$ `// Factor marginalization`
12 **end**
13 **foreach** $\phi \in \Phi$ **do**
14 | $\varphi \leftarrow \varphi * \phi$ `// Factor product`
15 **end**
16 $Z \leftarrow \sum_{X_1 \ldots X_n} \varphi$
17 $P \leftarrow \varphi / Z$ `// Factor normalization`

Algorithm 1: Variable Elimination algorithm

IV. A CONTENT-CENTRIC BAYESIAN INFERENCE ALGORITHM

The application of a Bayesian inference algorithm can be resource consuming. Several research initiatives propose various ways to accelerate it. Lu et al. [26] propose a parallel Message passing approach which distributes the computation of each message passing in the inference algorithm and accelerates the algorithm using GPU. Yinglong et al in [27] propose an exact inference algorithm by decomposing and merging junction trees and distributing the subset of junction trees in the network. These works both demonstrate the possibility to distribute the computation effort of BNC inference but they do not build on a content-centric approach which can substantially reduce it. In this section, we explain how we leverage NFN to support BNC; more specifically, we propose the data structure, naming scheme, and transformation of BNC functions into λ-calculus.

A. Naming scheme and data structure

As depicted in the VE algorithm, the $factor$ is the core of the inference algorithm. A $factor$ is not only the input but also the output of functions. Besides, the $evidence$ parameter can also be used to calculate the *factor reduction*. A variable is also a parameter in the VE algorithm but it does not carry additional information except its name. As such, we deliberately do not name the variable and use it as a string. The structures for $factor$ and $evidence$, their naming schemes and the naming scheme of functions are described in the following section.

1) Factor: The $factor$ packet includes three parts: the list of the names of random variables, their dimensions and the list of all values of ϕ. The two first parts refer to the meta-data of the packet, while the third part is the actual data. The list of all values is sorted in the order of a $factor$. In other words, the third part of the data structure is a serialization of values of ϕ when the $factor$ is expressed in form of a table. Figure 1 illustrates this structure.

More specifically, our proposed data structures include two types of $factors$: *initial factors* and *temporary factors*.

The *initial factors*, which are established by CPDs, are named as follows: */data/fac/initial/<name of variables>*. As an example, */data/fac/initial/AzBzC*, with "z" being the separator character, is the name for the $factor$ shown in Table I. This naming scheme is not the sole alternative but it is the simplest one. One can choose to name each entry of a $factor$ as a data. Nonetheless, since a $factor$ operation uses all $factor$'s entries, this means that a large number of parameters will be required to feed the operation, thus making the approach more complex.

Besides, $factors$ are also the outputs of operations in the inference algorithm and then the inputs for subsequent ones. Therefore, *temporary factors* are used to distribute the computation between nodes in an NFN network. These $factors$ are named using their hash: */data/fac/temporary/<hash(factor)>*. An alternative for naming a *temporary factor* consists in using its values as its name, however, when it is complex, the name may become too large to be stored in an NFN packet. Besides, another choice consists in generating the name of the *temporary factor* randomly or incrementally. However, if we name the $factors$ in each NFN node randomly or incrementally, when a node asks others to compute an operation while another node already uses this name for a different $factor$, the result will be inexact. Hash functions avoid such a collusion since the same hash means that the same data is stored in the cache.

2) Evidence: Similarly, a packet of $evidence$ encompasses three parts: the name of variables, their dimensions, and values of the $evidence$. The data structure of $evidence$ is designed through the following ideas. Firstly, the $evidence$ of a variable $X_e = x_e$ shows that the value of the variable X_e is known (x_e). As a consequence, entries that encompass values $X_e \neq x_e$ will be dropped while introducing the $evidence$. For this purpose, the value of the $evidence$ for a variable in a packet consists of a chain of 0's and 1's. Value 0 signifies that the corresponding entry will be dropped while value 1 means that the entry will remain while introducing the $evidence$. As an illustration of the $evidence$ $A = 2$, the value in the $evidence$ packet is 010.

Secondly, an $evidence$ is composed of a multitude of variables. Consequently, the data of the $evidence$ is the merging of the values of each variable in the $evidence$. As an illustration of the $evidence$ $A = 2, C = 1$, the value in the $evidence$ packet is $010z10$.

Finally, an $evidence$ is designed to be computed with a $factor$. Some variables belong to E and other variables belong to $factor$. They both should be present in the value of the $evidence$. The values of these variables are still unknown. This means that they will remain while introducing the $evidence$, hence, they are marked at a chain of values 1. For example,

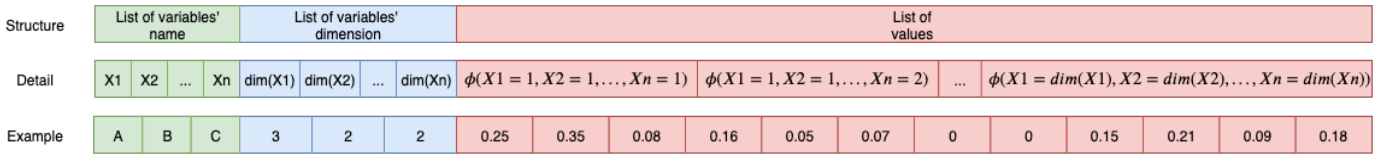

Fig. 1. Data structure of a $factor$

TABLE II
LIST OF HELPER FUNCTIONS

Function	Description
serialize	Serializes an array of $factor$'s values to a string
deserialize	Deserializes a string to an the array of $factor$'s values
productEviVal	Multiplies values of $evidence$ of two variables
verifyValidIndex	Verifies the index of values of product of two $factors$ is valid
mergeVarDim	Merges the set of random variables and their dimensions of two $factors$
removeVar	Removes a variable from the set of random variables in case of factor reduce or factor marginalization
getVarDimFac	Retrieves the set of random variables and their dimensions of a $factor$ as a string
getVarDimEvi	Retrieves the set of random variables and their dimensions of $evidence$ as a string
getValFac	Retrieves the values of a $factor$ as a string
getValEvi	Retrieves the values of $evidence$ as a string
listIdxFac	Generates the list of index for values of a $factor$

the value of the $evidence$ $A = 2, C = 1$ in the context of the $factor$ $\phi(A, B, C)$ is $010z11z10$.

On the other hand, the meta-data encompasses only names and dimensions of variables which belongs to the $evidence$ E. As an example, for the $evidence$ $A = 2, C = 1$ in context of the $factor$ $\phi(A, B, C)$, the meta-data of this $evidence$ is $AzCw3z2$, while the full packet of the $evidence$ is $AzCw3z2w010z11z10$, with "w" and "z" being the separator characters.

In the proposed approach, an $evidence$ is named using the list of names of variables and the value of the $evidence$: */data/evi/<name of variables>/<values of* $evidence$*>*. As the $evidence$ should be customized to be operated with $factor$, we also deliberately list the variables in the $factor$ that are still unknown and marked as "NaN". For example, */data/evi/AzBzC/2zNaNz1* is the name for the $evidence$ $A = 2, C = 1$, which is compatible with $factor$ $\phi(A, B, C)$.

3) Function: Finally, prefixes */func/<name of functions>/* are reserved for NFN functions. These functions are identified by their names and parameters (where a parameter can be a string, integer or a data name). For instance, the functions of the VE algorithms are:

- */func/reduce/(/data/fac/.../data/evi/...)*
- */func/product/(/data/fac/.../data/fac/...)*
- */func/marginalize/(/data/fac/...,variable)*
- */func/normalize/(/data/fac/...)*

B. Transformation of functions into λ-calculus

As a λ-calculus expression resolution engine is integrated into each NFN node, to adapt the inference algorithm in NFN, we transform the functions of the VE algorithm into λ-calculus, hosted in the NFN processing server. To conduct the inference, various functions are needed but, only the four main ones of the VE algorithm are named in the NFN forwarder. Other functions, called helper functions, listed in table II, are also transformed into λ-calculus and added in the NFN compute server but they are not named using the previously defined naming scheme.

These functions demand significant resources to be executed. In the following sections, we will explain in detail how we transform these functions into λ-calculus, and then how we leverage NFN.

1) Factor reduction: Let $\phi(X)$ be a $factor$, and an $evidence$ $E = e$. We define the reduction of the $factor$ ϕ to the context $E = e$, denoted $\phi[E = e]$, to be a $factor$ over scope $X - E$, such that: $\phi'[E = e](X - E) = \phi(X - E, E = e)$[25]

Now, we only need to consider how we introduce the $evidence$. The *factor reduction* function is built on the idea of erasing invalid entries that conflict with the introduced $evidence$. For example, given the evidence $X_e = x_e$, and a $factor$ $\phi(X_1 \ldots, X_n)$, the new $factor$ $\phi(X_1, \ldots X_{e-1}, X_e = x_e, X_{e+1}, \ldots, X_n)$ is constructed by removing entries which include $X_e \neq x_e$.

The value of the $evidence$ in our approach is designed to mark that the entry should be dropped in the $factor$. To identify the entry which should be dropped, we use value 0, and for the entry that should be retained after the operation, value 1. As shown, the third part of the $factor$ is a serialization of values of a $factor$ in the right order. However, the value $evidence$ is just a fusion of $evidence$ from different variables. Therefore, a function is constructed that allows merging the $evidence$ of variables to an intermediate data that is compatible with the dimension and the right order of the $factor$ to reduce. Function *productEviVal* allows merging of $evidence$ from all the random variables in the $factor$ and providing new data that is compatible with the $factor$. The idea of the algorithm is that we realize the product between values in the $evidence$ of two variables (line 2). In particular, if the value in one of the variables is already marked to be dropped (0) it also means that the entry consisting of this value with all values of other variables will be marked 0, and will also be dropped.

```
1 Function productEviVal(evi1,evi2)
2 |   return reduce(lambda x,y:x+y,[[i*j for i in evi1] for j
  |     in evi2])
```

The following algorithm enables the introduction of an $evidence$ into a $factor$. Firstly, function *getVarDim* allows merging names of variables and their dimensions of $factors$ and $evidence$ into one, which is the meta-data of the new

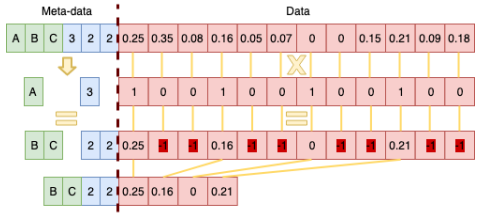

Fig. 2. Example of function factor reduction

$factor$ (line 2). Secondly, values in the $factor$ and $evidence$ are calculated to establish the data of the new $factor$. If the value of the $evidence$ is 0, this means that the entry of the $factor$ will be dropped (line 4). The result will be marked -1, as a value of a valid $factor$ is always positive, and this entry will be dropped later. By contrast, if the value of the $evidence$ is 1, which means that the entry is still valid, the value in the new $factor$ is retained by multiplying it with 1 (line 4). Finally, all of the invalid values in the $factor$, i.e. those which are negative, are removed to finally obtain a valid $factor$ (line 3). Figure 2 illustrates this process.

```
1 Function reduceFactor(fac,evi)
2   return getVarDim(fac,getVarDimEvi(evi)) +
      SEPARATOR_CHAR
3   serialize(filter(lambda x:x >-1,
4     map(lambda x,y: -1 if y == 0 else x*y,
5       deserialize(fac), reduce(lambda x,y:
          productEviVal(x,y),getValEvi(evi)))))
```

2) Factor product: The next function that is transformed into a λ-calculus function is the $factor$ product. The function is defined as follows: let X,Y and Z be three disjoint sets of variables, and let $\phi_1(X,Y)$ and $\phi_2(Y,Z)$ be two $factors$. We define the $factor$ product $\phi_1 \cdot \phi_2$ to be a $factor$ ψ: $Val(X,Y,Z) \rightarrow \mathbb{R}$ as follows: $\psi(X,Y,Z) = \phi_1(X,Y) \cdot \phi_2(Y,Z)$ [25].

The goal of this operation is to multiply the values of two $factors$ together. The joint distribution of the resulting $factor$ is the merging of two scopes. However, several merging entries are not valid. For example $\phi(X = 1, Y = 1) \cdot \phi(Y = 2, Z = 1)$ is an invalid entry as $Y = 1$ and $Y = 2$ are in conflict and should be dropped. From this, our proposed approach consists in using the function *verifyValidIndex* to mark 1 if the entry is valid and 0 if it is not (line 6). Finally, we multiply the values of this function with the values of a $factor$ if it is positive and mark -1 if it is zero (line 4). Following the same methodology with the function *factor reduction*, entries with values -1 are considered as invalid entries and are dropped in the final $factor$ (line 3).

```
1 Function productFactor(f1,f2)
2   return getVarDimFac(mergeVarDim(f1,f2)) +
      SEPARATOR_CHAR
3   serialize(list(filter(lambda x:x >-1,
4     map(lambda x,y: -1 if y == 0 else x*y,
5       productValFac(f1,f2),
6       verifyValidIndex(f1,f2)))))
```

3) Factor marginalization: To achieve the sum-product operation, a *factor marginalization* is needed. This function is defined as follows: let X be a set of variables, and $Y \notin X$ a variable. Let $\phi(X,Y)$ be a $factor$. We define the $factor$ marginalization of Y in ϕ, denoted $\sum_Y$, to be a $factor$ ψ over X such that: $\psi(X) = \sum_Y \phi(X,Y)$ [25].

The idea of this algorithm is that we have a joint distribution and we want to get a new distribution that eliminates a random variable. Then, we compute the sums in the margin over the distribution of the variable being eliminated. For example, in the case of the $factor$ illustrated in Table I, we want to eliminate variable C. We will sum-up values of $\phi(A = 1, B = 1, C = 1)$ and $\phi(A = 1, B = 1, C = 2)$ to get values for the new $factor$ $\phi'(A = 1, B = 1)$. To perform this, we construct the function *listIdxFac*, which lists all entries of variables in the $factor$ but not the variable to eliminate (line 5). For example, let f be the $factor$ in table I:

```
listIdxFac(f,"C")=["A1B1","A1B2","A1B1",
"A1B2","A2B1","A2B2","A2B1","A2B2",
"A3B1","A3B2","A3B1","A3B2"]
```

Thanks to the *listIdxFac* function, we list all unique entries of random variables of the new $factor$ (line 6):

```
listIdxFac(getVarDimFac(removeVar(f,"C"))
,"C")=["A1B1","A1B2","A2B1","A2B2",
"A3B1","A3B2"]
```

Then, for each value in the list of unique entries, we locate the positions of duplicated entries and mark them as 1 (line 5). This step allows us to know the positions of values that belong to the same entry in the new $factor$, to sum up these values. Finally, the value of ϕ in the new $factor$ is calculated from the sum of duplicated entries where their positions are marked as 1.

```
1 Function marginalizeFactor(f,v)
2   return getVarDimFac(removeVar(f,v)) +
      SEPARATOR_CHAR
3   serialize(
4     map(lambda z:reduce(lambda x,y:x+y,
5       map(lambda x,y:x*y, getValFac(f),map(lambda
          x:1 if z in x else 0,listIdxFac(f,v))))
6     ,listIdxFac(getVarDimFac(removeVar(f,v)),v)))
```

4) Factor Normalization: *Factor normalization* is the simplest of all the functions described. Here, we divide all the values of the $factor$ (line 5-6) by their sum to normalize the $factor$ (line 3) and finally obtain a conditional probability: $P(X_q|E = e) = \frac{1}{Z}\prod_{i=1}^{n}\phi'_i(X_i)$[25] where $Z = \sum_{i=1}^{n}\phi'_i(X_i)$.

```
1 Function normalizeFactor(fac)
2   return getVarDimFac(fac) +SEPARATOR_CHAR
3     serialize(map(lambda x,y:x/y,
4       deserialize(getValFac(fac)),
5       [reduce(lambda x ,y : x+y ,
6         deserialize(getValFac(fac)))]
7         *len(list(deserialize(getValFac(fac))))))
```

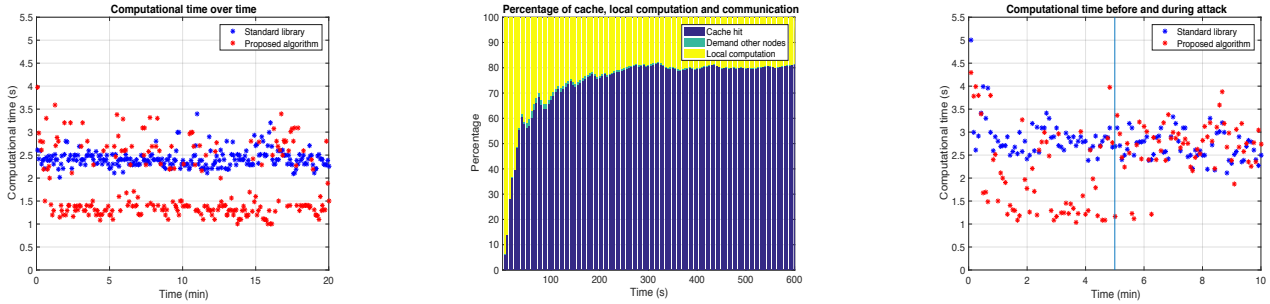

Fig. 3. Performance of the proposed approach over time: (a) Snapshot of computational time over time; (b) Evolution of proportion of requests over time; (c) Snapshot of computational time before and during attack

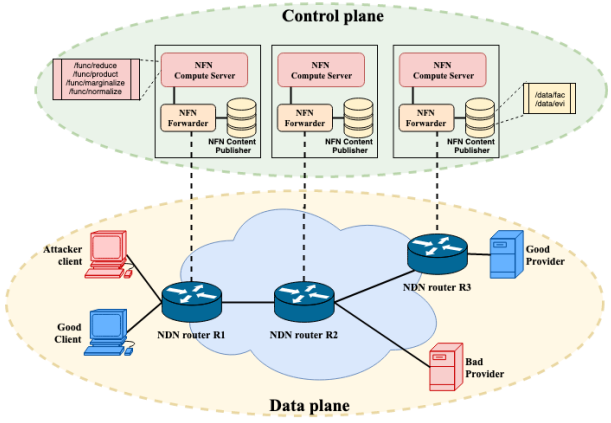

Fig. 4. Experiment topology

V. Numerical results

A. Use case: the Content Poisoning Attack

To evaluate the performance of our approach, the Content Poisoning Attack (CPA) is considered as a use-case. The detection of this attack is achieved through a dedicated detector presented in [28]. The considered topology, containing three NDN routers with both a NFN forwarder and a NFN compute server, is depicted in Figure 4. Each node integrates all functions, data, and exchange results. A legitimate producer of NDN content is connected to R1 while a malicious one is connected to R2. R2 is connected to both a legitimate user and an attacker whose purpose consists in corrupting the data hosted in all network caches to prevent the legitimate user from accessing the desired and correct content. This attack is one of the major threats by the NDN community. A BN is proposed in [29] as an anomaly detection framework. It consists in 19 discrete nodes: an Anomaly node, and 18 metrics. An MMT monitoring probe is coupled with each router to extract and collect data concerning the 18 metrics every 5 seconds. However, to demonstrate and evaluate all functions in the VE algorithm, we consider one of the metrics as unknown; otherwise, the *factor marginalization* function would not be used as there would be no variable to eliminate.

The proposed approach is implemented using PiCN [30], which is the newest version of NFN, written in python. It is compared with a standard Bayesian Network inference algorithm leveraging an open-source library named pgmpy [31] also written in python.

B. Evaluation

We evaluate the performance under different conditions to find out how anomaly detection can be improved by NFN.

1) Performance over time: To assess the performance of our approach, we simulate normal traffic during 10-minute intervals and we measure the processing time evolution, cache hit, local computation and requests to other nodes. As shown in Figure 3.a, the cache is, as expected, almost empty at the beginning. This explains the higher processing time of our approach as compared to the standard one. The effectiveness of our solution is shown when the cache is fed with results from previous executions or from neighbor nodes propagation. In this case, the computational time is lower than that of the standard approach. This is confirmed by the result of Figure 3.b, which shows that the average percentage of local computation and requests to other nodes are the highest at the beginning. Furthermore, we note that the portion of the cache hits initially increases over time, then stabilizes since the cache reaches its limit. The percentage of the cache hits after 10 minutes is 80%. This shows the effectiveness of our approach.

2) Impact of normal and abnormal traffic: Another evolution of the computational time, illustrated in Figure 3.c, is evaluated in the case of an attack, where we simulate normal traffic during 5 minutes, and then the CPA attack during 5 minutes. The vertical blue line marks the time at which the attack starts. As we can see, after the attack, the computational time increases. Since the metrics during an attack are abnormal in comparison to normal traffic, the detection function cannot be found in the cache and it requires additional time to compute operations in the inference algorithm. To conclude, NFN is efficient in normal traffic but, when the attack occurs, the proportion of repeated computational operations decreases, so the computation time with NFN increases.

3) Impact of available CPU resources: The result shown in Figure 5.a proves that the computational time in our approach is lower than that of the standard approach when the computation capacity of hosting node is limited. When the available CPU resource of the router is smaller than 0.4, the proposed

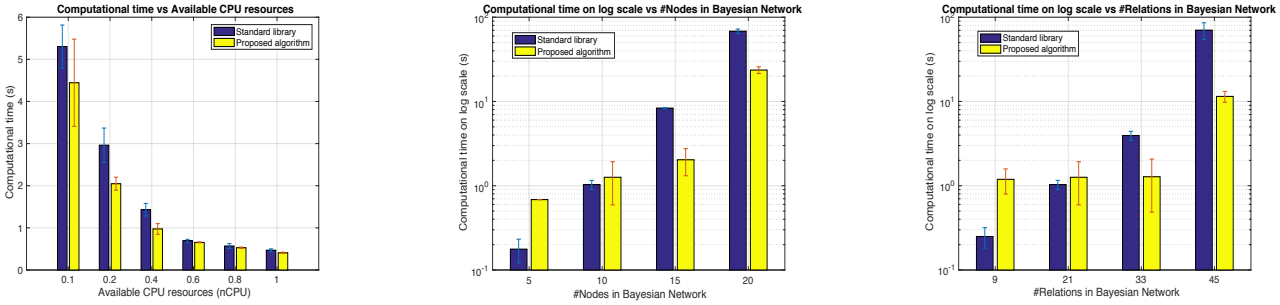

Fig. 5. Impacts on computational time (semi-log scale): (a) Available CPU resources; (b) Number of nodes in BN; (c) Number of relations in BN

approach performs better. The reason is that when the router has sufficient computational resources, it will consume all of the resources to execute the operations of the inference algorithm. Therefore, a standard approach can consume all the resources and perform computational operation rapidly. On the other hand, when it reaches its computation limit (in case of limited resources), the use of the cache helps to better execute the computational operations than with the standard approach. It reduces the computational time from 10% to 30%. Figure 6 also illustrates the usage of CPU during these experiments. We can here conclude from Figure 5.a and 6 that we can benefit from NFN when the computational capacity is limited, which is the case in IoT-type networks.

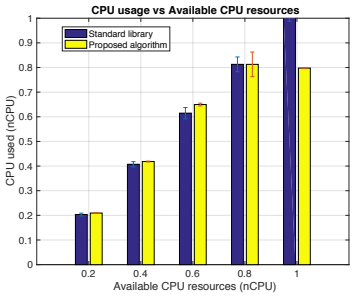

Fig. 6. Impact of available CPU resources on CPU usage

4) Impact of the BN complexity: To evaluate the performance of our approach according to the complexity of BN, the number of nodes in our BN and the number of relations is considered. As shown in Figure 5.b and 5.c, when the BN is simple, thus counting a small number of nodes (less than 10) or relations (less than 21), the computational time in the standard approach is undoubtedly better than in our approach. The reason is that the number of computational operations is ordinary, which means that it does not need the help of a cache or other nodes to perform the computations. However, when the BN becomes complex, the performance of our approach is substantially better than the standard method. In this case, our approach calculates 4 to 8 times faster than the standard one. In fact, in this case, the number of computation operations is huge and exceeds the computational capacity of the node. As a consequence, the use of a cache and the help of other nodes become extremely important. The results show that we benefit from the NFN infrastructure when the BN is complex and needs a significant amount of operations.

5) Impact of latency between nodes: The impact of latency between nodes finally needs to be considered. When the latency increases, the computational time increases as well because the latency presumably has an impact on the communications between the nodes to execute the computational operations. However, this latency is too high (greater than 100ms) and we can see that the computational time does not increase noticeably. By contrast, it is smaller than in the case where the latency is 20ms. At the beginning, the NFN router sends requests to neighbors but it does not receive any result. When the timeout is triggered several times the router does not send requests to the other nodes, and it decides to calculate the operations locally. As a consequence, the following operations are executed faster than in case of 20ms of delay, in which the routers always wait for the response from their neighbors.

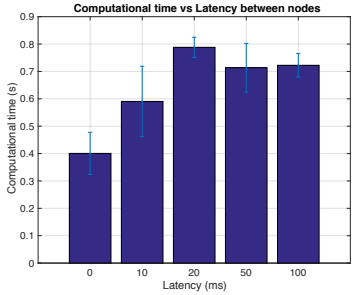

Fig. 7. Impact of delay on computational time

VI. Conclusion and Future Work

Anomaly detection is a critical and complex task which can be supported by NFN. In this paper, we have defined the core elements for the design and implementation of an NFN-supported Bayesian Network inference algorithm. To that aim, we have shown that the VE algorithm in BN can be transformed to λ-calculus functions and then, thanks to NFN, cache and share results between nodes. In the context of CPA detection, we have demonstrated that an NFN-supported BN performs better not only in the case of limited computational resources but also when the BN is complex, thus proving the benefit of this approach for complex anomaly detection function operated in the context of IoT, for instance.

Our future work will focus on further developing the content-oriented control plane by extending the current approach to integrate other methods for anomaly detection. Moreover, other attacks will also be considered to demonstrate the applicability and generality of the proposed approach.

ACKNOWLEDGMENT

This work is partially co-funded by (1) the French National Research Agency (ANR), DOCTOR project, <ANR-14- CE28-0001>, started in 01/12/2014 and supported by the French Systematic cluster and (2) the CRCA and FEDER CyberSec Platform, <201304601>.

REFERENCES

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang, and others, "Named data networking," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, 2014.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[3] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An information centric network for computing the distribution of computations," in *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014, pp. 137–146.

[4] C. Tschudin and M. Sifalakis, "Named functions and cached computations," in *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*. IEEE, 2014, pp. 851–857.

[5] T. N. Nguyen, H.-L. Mai, G. Doyen, R. Cogranne, W. Mallouli, E. Montes de Oca, and O. Festor, "A security monitoring plane for named data networking deployment," *To appear in IEEE Communication Magazine – Feature Topic on Information-Centric Networking Security*, 2018.

[6] S. Rajasegarar, C. Leckie, M. Palaniswami, and J. C. Bezdek, "Distributed anomaly detection in wireless sensor networks," in *2006 10th IEEE Singapore international conference on communication systems*. IEEE, 2006, pp. 1–5.

[7] A. A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for internet of things," *Future Generation Computer Systems*, vol. 82, pp. 761–768, 2018.

[8] F. Palmieri, U. Fiore, and A. Castiglione, "A distributed approach to network anomaly detection based on independent component analysis," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 5, pp. 1113–1129, 2014.

[9] J.-L. Krivine, "A call-by-name lambda-calculus machine," *Higher-Order and Symbolic Computation*, vol. 20, no. 3, pp. 199–207, 2007.

[10] C. Tschudin and M. Sifalakis, "Named functions for media delivery orchestration," in *2013 20th International Packet Video Workshop*. IEEE, 2013, pp. 1–8.

[11] C. Scherb, U. Schnurrenberger, C. Marxer, and C. Tschudin, "In-network live stream processing with named functions," in *Workshop on Information-Centric Fog Computing, IFIP Networking*, 2017.

[12] C. Marxer and C. Tschudin, "Improved content addressability through relational data modeling and in-network processing elements," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 29–35.

[13] C. Marxer, C. Scherb, and C. F. Tschudin, "Access-controlled in-network processing of named data." in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016, pp. 77–82.

[14] C. Marxer and C. Tschudin, "Schematized access control for data cubes and trees," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 170–175.

[15] M. Król and I. Psaras, "Nfaas: named function as a service," in *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 2017, pp. 134–144.

[16] H. Liu, F. Eldarrat, H. Alqahtani, A. Reznik, X. De Foy, and Y. Zhang, "Mobile edge cloud system: Architectures, challenges, and approaches," *IEEE Systems Journal*, vol. 12, no. 3, pp. 2495–2508, 2017.

[17] D. Grewe, M. Wagner, M. Arumaithurai, I. Psaras, and D. Kutscher, "Information-centric mobile edge computing for connected vehicle environments: Challenges and research directions," in *Proceedings of the Workshop on Mobile Edge Communications*. ACM, 2017, pp. 7–12.

[18] J. Burke, "Browsing an augmented reality with named data networking," in *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, 2017, pp. 1–9.

[19] P. TalebiFard, R. Ravindran, A. Chakraborti, J. Pan, A. Mercian, G. Wang, and V. C. Leung, "An information centric networking approach towards contextualized edge service," in *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 2015, pp. 250–255.

[20] Q. Wang, B. Lee, N. Murray, and Y. Qiao, "Cs-man: Computation service management for iot in-network processing," in *Signals and Systems Conference (ISSC), 2016 27th Irish*. IEEE, 2016, pp. 1–6.

[21] Y. Ye, Y. Qiao, B. Lee, and N. Murray, "Piot: Programmable iot using information centric networking," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 825–829.

[22] Q. Wang, B. Lee, N. Murray, and Y. Qiao, "Iproiot: An in-network processing framework for iot using information centric networking," in *Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on*. IEEE, 2017, pp. 93–98.

[23] H. Zhang, Z. Wang, C. Scherb, C. Marxer, J. Burke, L. Zhang, and C. F. Tschudin, "Sharing mhealth data via named data networking." in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016, pp. 142–147.

[24] L. M. JSM, V. Lokesh, and G. C. Polyzos, "Energy efficient context based forwarding strategy in named data networking of things," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*. ACM, 2016, pp. 249–254.

[25] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[26] L. Zheng, O. Mengshoel, and J. Chong, "Belief propagation by message passing in junction trees: Computing each message faster using gpu parallelization," *arXiv preprint arXiv:1202.3777*, 2012.

[27] Y. Xia and V. K. Prasanna, "Distributed evidence propagation in junction trees on clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 7, pp. 1169–1177, 2011.

[28] "Implementation of content poisoning attack detection and reaction in virtualized NDN networks," in *21st Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN 2018, Paris, France, February 19-22, 2018*, 2018, pp. 1–3. [Online]. Available: https://doi.org/10.1109/ICIN.2018.8401591

[29] H. L. Mai, N. T. Nguyen, G. Doyen, R. Cogranne, M. Wissam, E. Montes de Oca, and O. Festor, "Towards a security monitoring plane for named data networking: Application to content poisoning attack," in *Network Operations and Management Symposium (NOMS), 2018 IEEE*. IEEE, 2018.

[30] "Picn: Python icn and nfn by university of basel," https://github.com/cn-uofbasel/PiCN.

[31] "Python library for probabilistic graphical models," https://github.com/pgmpy/pgmpy.