BitCoding: Network Traffic Classification Through Encoded Bit Level Signatures

Neminath Hubballi and Mayank Swarnkar

***Abstract*— With many network protocols using obfuscation techniques to hide their identity, robust methods of traffic classification are required. In traditional deep-packet-inspection (DPI) methods, application specific signatures are generated with byte-level data from payload. Increasingly new data formats are being used to encode the application protocols with bit-level information which render the byte-level signatures ineffective. In this paper, we describe BitCoding a bit-level DPI-based signature generation technique. BitCoding uses only a small number of initial bits from a flow and identify invariant bits as signature. Subsequently, these bit signatures are encoded and transformed into a newly defined state transition machine transition constrained counting automata. While short signatures are efficient for processing, this will increase the chances of collision and cross signature matching with increase in number of signatures (applications). We describe a method for signature similarity detection using a variant of Hamming distance and propose to increase the length of signatures for a subset of protocols to avoid overlaps. We perform extensive experiments with three different data sets consisting of 537 380 flows with a packet count of 3 445 969 and show that, BitCoding has very good detection performance across different types of protocols (text, binary, and proprietary) making it protocol-type agnostic. Further, to understand the portability of signatures generated we perform cross evaluation, i.e., signatures generated from one site are used for testing with data from other sites to conclude that it will lead to a small compromise in detection performance.**

***Index Terms*— Traffic classification, DPI, bit-level signatures.**

I. Introduction

IN ANY network, traffic is heterogeneous in nature carrying packets of different applications and users. Typically each application has a requirement in terms of delay, end-to-end time, jitter, etc for its correct and useful usage. In order to meet these requirements the service providers usually apply Quality of Service (QoS) to different application flows. These QoS actions are in the form of prioritizing traffic of one application over the other. To facilitate these QoS actions, the flows originated from different applications need to be identified correctly. This task of identifying the application to which an aggregate set of packets belong to is known as traffic classification [8], [24]. Traffic classification also has applications in security monitoring where administrator of a network may block certain applications from being used in her network. In addition, traffic classification and characterization is useful for network management activities like traffic engineering, resource provisioning, network fault detection and localization and also in identifying anomalies. It also assumes relevant significance even from a privacy standpoint, e.g. classification of sensitive applications/services may be maliciously exploited from eavesdroppers/unauthorized entities.

Traditionally traffic classification or application identification was done with port numbers [22] as most applications used predefined port numbers. Identifying applications through this method is unreliable these days, mainly because many applications use non standard port numbers, use encryption to hide their identity, tunnel through other protocols. With the emergence of peer-to-peer applications (ex: eMule [6], eDonkey [2], Bit-torrent [5]) and Internet traffic anonymizers like Tor [4], port number based classification is highly inaccurate [16]. To address the issue, techniques which use traffic statistics and machine learning algorithms [11], [21], [32], behavior of applications [15], [17] and deep packet inspection methods [25], [31] are used for traffic classification.

Deep-Packet Inspection (DPI) methods analyze the content carried in the packets to identify applications. These methods perform well in detecting applications on non-encrypted traffic. Many DPI-based detection methods generate application specific signatures using payload content. Traditionally byte level data is used for generating signatures. An important challenge for application identification techniques using DPI is to capture the signatures which are unique and are minimal in length, as signature length governs the computational overhead. Signature generation is mostly a manual effort which can be error prone. Some recent works [25] propose to automate the signature generation. However, there is a surge in data-transfer formats and increasingly network protocols and applications are encoded at the bit-level [29], [30]. This makes the byte-level signatures ineffective in traffic classification. Borrowing motivation from this, we propose a DPI-based bit-level signature generation method for accurate traffic classification. Our method generates bit-level signatures using first n bits of bidirectional flows belonging to application. In particular we make following contributions in this paper.

- We propose $BitCoding$ which is a bit-level signature based application identification method. $BitCoding$ uses data compression for efficient representation of application signatures.
- $BitCoding$ is computationally very inexpensive as it generates signatures from the first n bits of

Manuscript received January 9, 2018; revised June 10, 2018 and August 17, 2018; accepted August 25, 2018; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor M. Schapira. Date of publication September 14, 2018; date of current version October 15, 2018. This work was supported by SERB, Government of India under Grant YSS/2015/000073. *(Corresponding author: Neminath Hubballi.)*

The authors are with the Discipline of Computer Science and Engineering, IIT Indore, Indore 453552, India (e-mail: neminath@iiti.ac.in; phd1401101001@iiti.ac.in).

Digital Object Identifier 10.1109/TNET.2018.2868816

1063-6692 © 2018 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

bidirectional flow. Our experiments show that even with first 40 bits of flow we can reliably identify applications.
- We translate the compressed signatures of application into a state transition machine ($Transition\ Constrained\ Counting\ Automata$) for bit-level matching with traffic flows. For signature matching aggregate set of packets are processed to extract first n bits and are given as input to $TCCA$ which makes appropriate transitions to accept or reject the bit string (hence classify the flow).
- We perform extensive experiments on three different datasets consisting of text based protocols, binary protocols and proprietary protocols, to assess the performance of $BitCoding$ and show that it works for all of them which makes it protocol-type agnostic.
- We perform cross evaluation of $BitCoding$ on datasets from different sites and show that it is able to classify flows with a small compromise in the detection rate.

We organize remainder of this paper as follows. In section II we describe related literature. In section III we describe the working principle of $BitCoding$. Method for handling the overlapping signatures and cross signature matching is described in section IV. The asymptotic complexity of various components of $BitCoding$ are detailed in section V. We describe the experiments in section VI. Experiments done to assess the robustness of signatures for cross application matching is explained in section VII. In section VIII we give details of experiments done to assess the impact of increasing the number of bits in signature and also the number of flows used to generate signatures. In section IX, we give the results of comparison of $BitCoding$ with couple of prior works. We end this paper with concluding remarks in section X.

II. Prior Work

In this section we describe some of the recent works on traffic classification. These methods mainly fall into two broad categories as those using machine learning and based on DPI techniques.

A. Machine Learning Methods Using Feature Statistics

There is significant amount of literature describing network traffic classification using machine learning algorithms. Both supervised and unsupervised machine learning algorithms have been used to learn and predict flow classes (applications) using a set of statistical features extracted from packet header (flow), connection and timing information. The feature set for these machine learning algorithms are mainly statistics like number of packets in either direction, number of bytes, connection duration, size of the packets, etc.

1) Supervised Learning: Moore and Zuew [21] use Naive Bayes algorithm, Auld *et al.* [9] use Bayesian Networks using a set of statistical features to classify traffic. Lim [18] performed a comparative study of various machine learning algorithms for traffic classification to conclude C.4 algorithm has better performance than others. Zhang *et al.* [33] used three features of flows (destination port, destination ip and protocol) as correlation information with Nearest Neighbor Classifier for classification to address small training sample size. This correlation improves the accuracy of NN classifier. Recently supervised machine learning algorithms have also been used to identify whether traffic has originated from anonymizers like Tor, I2P and JonDonym [20].

2) Unsupervised Learning: These methods use feature vectors generated from flows and create clusters using a notion of similarity. The clusters so created (from training set) are labeled and subsequently a test feature vector is assigned a label based on the notion of similarity to the nearest cluster. McGregor *et al.* [19] used Expectation Maximization (EM) algorithm to generate a set of clusters for classification. Erman *et al.* [12] used K-means, DBSCAN and Autoclass clustering algorithms for traffic classification with similar features used in supervised machine learning methods.

B. Using Payload Content

Payload analysis or DPI-based methods use payload content of packets/flows to identify applications. Usually signatures generated from the payload are used to find a match in network traffic as it contains application related information. These signatures are in the form of keywords [25] or in the form of short sequence of bytes known as n-grams [14], [26], [31]. Application detection is based on sequence of keywords identified within the payload and their order of appearance in the payload or by the frequency of n-grams within the payload of a particular type of application. As searching for a signature match is a string comparison problem and is computationally expensive, few prior works also described techniques for efficient and accelerated string matching [10], [23], [27], [28]

Recently Yuan *et al.* [30] proposed to generate application specific signatures using bit-level data. BitMiner [30] generates bit-level signature by learning the association between the bit value and the position of bit within the payload. These bit signatures are expressed using extended regular expressions. Bitslearning [29] is an extension to BitMiner where they use machine learning algorithms to learn the bit values and their positions.

III. BitCoding

In this section we describe our proposed technique $BitCoding$ used for application identification/traffic classification in networks. $BitCoding$ uses signatures generated from bit-level content of payload[1] to identify flows corresponding to different applications. The choice of using bit-level content is motivated by the fact that, there are many protocols which encode data at bit-level. For example, consider the first 32 bits of NTP packet structure shown in Figure 1. The first two bits (0^{th} and 1^{st}) indicate whether the last minute of the day will have a leap second or not. Next 3 bits indicate the version number of NTP and the bits from 5 to 7 indicate different modes of NTP. In this case even the mode of NTP is changed (hence change in bit positions of 5 to 7) we can still use the first 5 bits to guess the protocol or even the first two bits are changing the bits corresponding to version number (which are not likely to change) can be indicators of protocol type. In another example case, DNS protocol sets a flag QR in

[1]Payload is the data in a packet above the transport layer header (TCP/UDP)

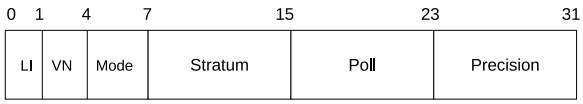

Fig. 1. First 32 bits of NTP packet structure.

header to indicate whether DNS packet is a query, which can be a signature bit to identify DNS flows. Thus the bit-level details can be used for robust signature generation.[2]

$BitCoding$ classifies bidirectional flows in the network traffic. These signatures are efficiently encoded so as to reduce the total signature length and encoded signatures are converted into a state transition machine. $BitCoding$ uses first n bits from bidirectional flow to generate signatures. The first step for signature generation is bidirectional flow reconstruction from network traffic. Subsequent to the reconstructed flow, first n bits of the flow are extracted for signature generation. In the next two subsections we elaborate on signature generation from training data and classifying flows from generated signatures respectively.

A. Signature Generation and Encoding

$BitCoding$ generates application specific signatures with training data. This phase has four stages as bidirectional flow reconstruction, bit-signature generation, run length encoding and state transition machine construction. These four stages are shown in Figure 2 and elaborated subsequently.

1) Constructing Bidirectional Network Flows: A network flow consists of a series of packets exchanged between two hosts. These two hosts are identified by two unique IP addresses. A network flow is uniquely identified by the 5-tuple

$SrcIP, DstIP, SrcPort, DstPort, Protocol$ where

$SrcIP$	Source IP address of host
$DstIP$	Destination IP address of host
$SrcPort$	Source Port number
$DstPort$	Destination Port numbers and
$Protocol$	Layer 4 Protocol (TCP/UDP)

All the packets sharing the same 5-tuple will be part of a flow. All the packet's (within the flow) payload data is taken and concatenated which is then used as input for the subsequent signature generation phase. As layer-4 protocol is also part of 5-tuple, flows are usually differentiated as TCP flow and UDP flow.

TCP Flow: A TCP connection is established with a 3-way handshake mechanism which generates a bidirectional flow of packets between two hosts (A and B) as below. All the packets in between this 3-way handshake and till the connection closure with a 4-way handshake are part of this flow.

1) $(A) \dashrightarrow [SYN] \dashrightarrow (B)$
2) $(A) \dashleftarrow [SYN/ACK] \dashleftarrow (B)$
3) $(A) \dashrightarrow [ACK] \dashrightarrow (B)$

UDP Flow: UDP is a connectionless protocol and hence identifying a flow and differentiating flows between same hosts is little different than TCP flow. In this case, timing information is used for differentiating the flows. All packets having an inter-packet timing of not more than δ and sharing the same 5-tuple i.e., source IP address, source port, destination IP address and destination port are considered as part of one flow. The threshold δ is a user defined threshold and is fixed suitably.

[2]Bit-level signatures guarantee a lower bound performance of byte-level signatures for the same content length

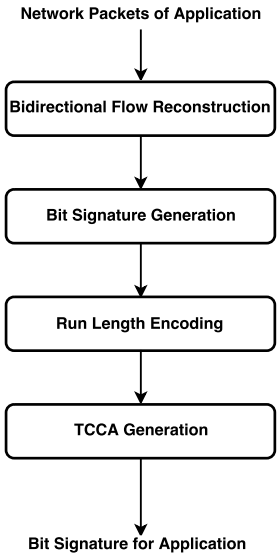

Fig. 2. Bit-level signature generation.

TABLE I
FLOW EXTRACTS AND SIGNATURE GENERATION

Flow 1	f_1^1	f_2^1	....	f_n^1
Flow 2	f_1^2	f_2^2	....	f_n^2
Flow 3	f_1^3	f_2^3	....	f_n^3
.	.	.	.	.
.	.	.	.	.
Flow K	f_1^K	f_2^K	....	f_n^K
Signature	s_1	s_2	....	s_n

2) Bit-Signature Generation: $BitCoding$ generates application specific bit-signatures using a set of invariant bits of the payload selected in the previous stage. Assuming there are K ($K \in I$) such bidirectional flows of an application in training set, it collects the first n bits from each of the K bidirectional flows of application $\mathcal{A}$ and generates n bit-signature A_{Sig} for that application. The first n bits of the i^{th} flow ($1 \leq i \leq K$) are of the form $f_1^i, f_2^i, \cdots, f_n^i$. All K flow extracts are used for generating signatures as follows. The j^{th} bit $[1, n]$ position of every flow extract are used to decide the j^{th} signature bit of A_{Sig}. The j^{th} signature bit is set to 0 if the j^{th} bit of every flow ($1 \leq j \leq K$) has a value of 0 and the j^{th} signature bit is set to 1 if the j^{th} bit of every flow ($1 \leq j \leq K$) has a value of 1. If some of these bit positions have 0's and 1's the j^{th} signature bit is set to *. The signature creation process is shown in Table I where each s_i is a signature bit.

A sample application bit-signature generation using bits is shown in Figure 3. In this example there are 3 flows each with 20 bits and are used for signature generation. The first bit of all

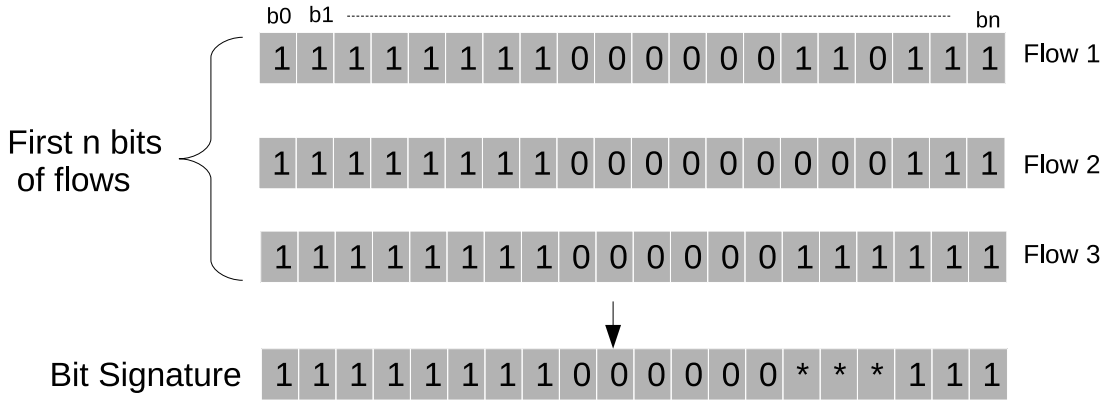

Fig. 3. Bit-signature generation.

the three flows are having a value of 1, thus this bit is included in the signature as 1. Also the next 7 bits of each of the flows are also having 1's, hence the next 7 signature bits are also labeled with 1's. Similarly the bit positions from 9-14 of all the 3 flows are having 0's and hence the signature bits 9-14 also will have 0's in these positions. The 15^{th} bit of flow 1 is having a bit value of 1, flow 2 is having a value of 0 and flow 3 is having a value of 1. Hence the corresponding bit in signature is set to * (indicating 0/1 and insignificant for detecting this application). Similar labelling is done with remaining bits and the final signature generated is $11111111000000***111$.

3) Run-Length Encoding: We can notice that, signature bits consist of 1's, 0's and *'s and each signature is of n bits. For efficient representation, storage and comparison purposes we perform Run-length Encoding (RLE) of these n bits. RLE is a technique used in loss-less data compression. RLE reduces the size of a repeating string of characters by specifying number of repetitions. In RLE runs of data probably i.e., sequences of the same data value in many consecutive data elements are stored as a single value and count of number of times that data value is repeated is stored. Consider the signature sequence for the above example which has bit values of $11111111000000***111$, and after encoding with RLE it is converted to $8W6Z3*3W$.

4) State Transition Machine Creation: Subsequent to the RLE there is an encoded signature which needs to be compared with network traffic flows (to identify applications). We convert the encoded bit-signature into a state transition machine. We elaborate on how to generate a state transition machine corresponding to each application below.

In order to represent and compare bits of signature, we design a new state transition machine $Transition\ Constrained\ Counting\ Automata$ ($TCCA$). This $TCCA$ is the final bit-signature for an application. Input to this state transition machine creation is compressed RLE bit-signature of an application. The machine $TCCA$ is formally defined as

$\mathcal{M} = (Q, \Sigma, C, \delta, q_0, F)$ where
- Q A finite set of states
- $q_0 \in Q$ Is an initial state
- Σ Is a finite set of input symbols
- $F \subseteq Q$ is a set of final states
- C Is a finite set of Counters with each $c_i \in C$ taking values in $\mathcal{N} \bigcup 0$
- δ Is a set of transitions defined as $\delta : Q \times C \times \Sigma \rightarrow Q \times (C \rightarrow C)$

Each transition $\delta_i \in \delta$ is a six tuple $\langle q_i, q_j, c, \sigma, \phi(c_i), Inc(c_j)\rangle$ with the elements representing
- q_i Current state
- q_j Next state
- c_i Counter value at the state q_i
- $\sigma \in \Sigma$ Input symbol
- $\phi(c_)$ Is a constraint (invariant condition) on counter value c_i at state q_i on this transition
- $Inc(c_j)$ Is a function which initialize the counter in the next state q_j to a new value

A sample $TCCA$ generated with 20 bit-signature (represented as RLE) in the above example is shown in Figure 4. There are 5 states in the $TCCA$ labeled from q_0 to q_4 with q_0 being start state and q_4 being final state. Each state has a counter (C_0 to C_4) which will be initialized to a new value every time a transition visits the state. The machine starts in state q_0 with counter at q_0 being set to 0, read bits from test flow and makes allowed transitions to reach the final state. The transitions of $TCCA$ have an input symbol (bit value) and a constraint on counter value which acts as a guard and the transition is allowed only if the constraint is satisfied (evaluated to be true). For instance in Figure 4 state q_0 has a transition defined to itself (self loop) on input 1. This transition has a constraint on counter value of C_0 being in between 0 to 8. This constraint maps the requirement of reading 8 consecutive 1's in the flow at the beginning, while incrementing the counter value by 1 each time it is traversed. The next transition from q_0 to q_1 is on input 0 and is valid only when the counter value at C_0 is 8 (having read 8 consecutive 1's) and sets the counter C_1 at q_1 to 1 (read a 0 after 8 consecutive 1's). Whenever there is * in signature, it will have two transitions one with input 0 and other with input 1 both increment the counter values at the next states. Similarly all bits and their positions are enforced by $TCCA$.

B. Flow Classification

The process of signature matching with $TCCA$ is shown in Figure 5. Similar to signature generation process, in this phase too (for flow classification) bidirectional flows

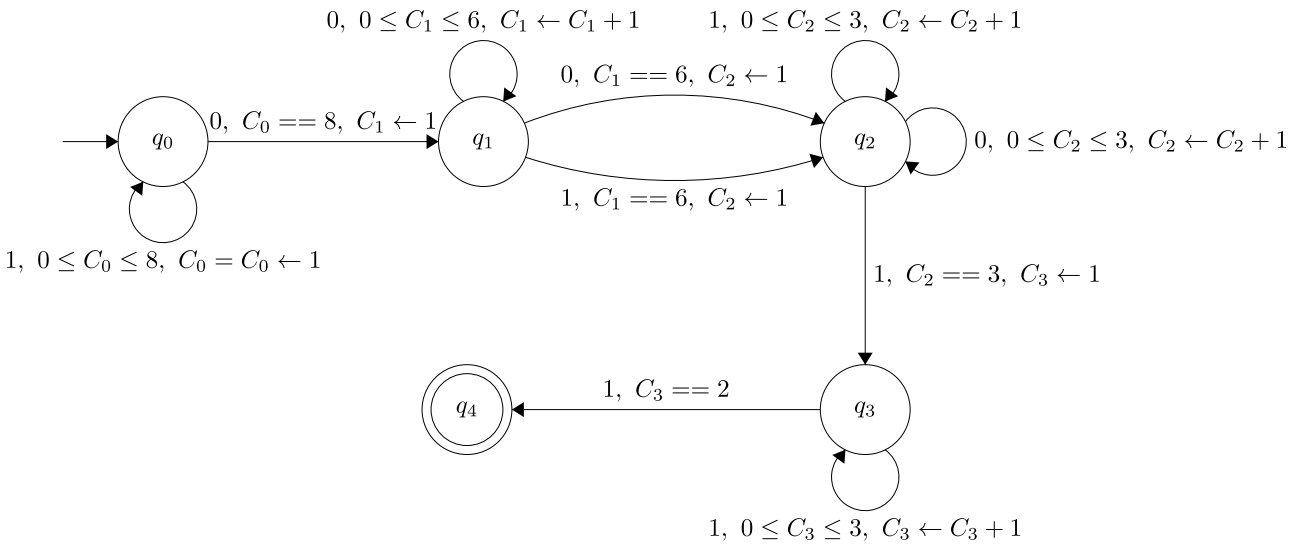

Fig. 4. $TCCA$ generated for the example signature 8W6Z3*3W.

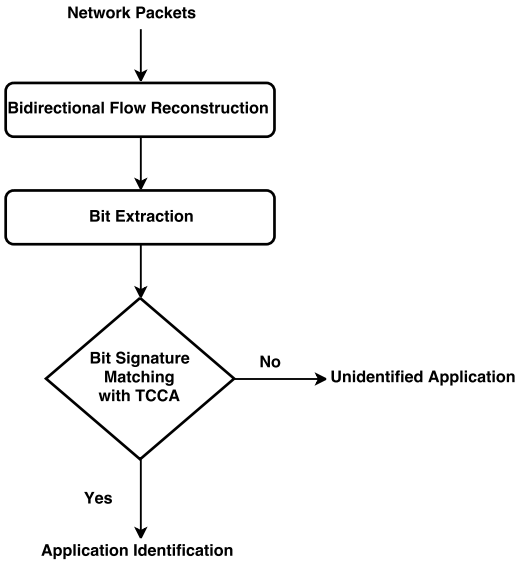

Fig. 5. Flow classification.

are reconstructed. First n bits of the flow are extracted for comparison with bit-signatures encoded and represented as $TCCA$. As mentioned previously every application has a $TCCA$ as its signature. Each of these $TCCA$'s will be in their respective start states and the n bits from a test flow F_{test} will be given as input (a bit at a time from first to last bit) to all the application signatures/$TCCA$s. Every $TCCA$ makes allowed transitions and the flow is labeled with application A if the corresponding $TCCA$ of application A reaches the final state. If none of the $TCCA$s reach the final state, it is labeled as not classified.

Consider two example bit sequences as 11111111000000101111 and 001111111000000101111. The first sequence 11111111000000101111 which is given as input to the $TCCA$ of Figure 4, it is easy to see that it reaches the final state as it contains eight 1's in the beginning and six 0's next, subsequent three bits are accepted invariably and then the last three digits are 1's. However second sequence is not accepted by $TCCA$ as it starts with a 0 and there is no transition with input 0 when the counter C_0 is 0 at state q_0.

IV. Addressing Signature Overlap

It has to be noted that, every application has one signature. While short signatures are computationally inexpensive (for both storage and comparison) and as number of applications increases there might be chances of signature overlap (two different applications having same signatures). This will lead to misclassification of flows of one application as other one.[3] This can be addressed by increasing the signature length of applications (see experiments section VIII-A). These two are contradictory goals (short signatures and no overlap) to be achieved together. To avoid such overlaps we compute the similarity between signatures of different applications using a form of $Hamming\ Distance$ (HD). Hamming distance is measured between two strings of equal length and its value is the number of bit positions at which the corresponding bits are differing. It is worth noting here that bit-signatures also contain *'s along with 0 and 1. We do a minor change in the standard calculation of HD between a pair of signatures, which is ignoring * bits from the comparison. We name this changed distance measurement as $Relaxed\ Hamming\ Distance\ (RHD)$. An example RHD calculation is shown in Figure 6. In this figure there are two bit-signatures and the corresponding RHD is 3.

We measure the RHD between every pair of applications and find that the reason for cross signature matching is due to low RHD between them. We label those signatures which exhibit maximum similarity (or minimum distance) with other signatures as $weak\ signatures$. In order to improve the classification performance we identify a set of application signatures which show maximum similarity and increase the signature lengths only for those protocols.

[3]Signature overlap is a common problem in firewalls and intrusion detection systems

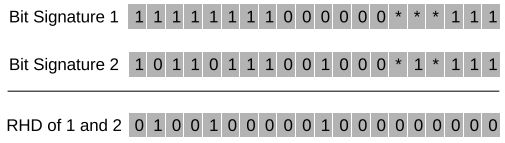

Fig. 6. RHD calculation between bit signatures.

V. Complexity Analysis

In this section we describe the asymptotic complexity of each component of $BitCoding$ model. Table II shows the complexity of each module of our proposed model. Flow reconstructor has to inspect each packet header and add it into appropriate flow. If there are l flows and p packets, then it has a complexity of O($l \times p$). Bit-Extractor has to read the first n bits which is a constant time operation, hence its complexity is O(1). Bit-signature generation has to operate on all flows of an application. If there are K flows of an application then complexity of signature generation is $O(K)$ as number of bits is constant n for every application which is $O(1)$ operation. Run-Length-Encoding (RLE) has to read n bits of signature which is $O(1)$ operation for each application and for X different applications it is $O(X)$. $TCCA$ construction is just encoding of RLE signature into a state transition machine which can be done in constant time $O(1)$ and as there are X applications total complexity is $O(X)$. Flow Classifier is a state transition machine and it has to match every bit with every possible instance of state transition machine. This in the worst case has a complexity of O($m \times n$) where m is number of machine instances and n is number of bits in a flow.

VI. Experiments and Results

In this section we describe the experiments done to evaluate the $BitCoding$ performance in detecting applications. In the next two subsections we elaborate on the datasets we used in our experiments and the results obtained with $BitCoding$ respectively.

A. Dataset Description

For our experiments with $BitCoding$, we used three different datasets. The first dataset we generated in Networks and Security Lab of IIT Indore by collecting network traffic. Second and third datasets are publicly available. One of this public dataset is from 'Digital Corpora' [1] and the other one is provided by FOI Information warfare lab [7] (FOI is the Swedish Defence Research Agency). In total we experiment with 20 different application layer protocols consisting of text and binary protocols. Some of these protocols are described through standards (RFC) and others are proprietary protocols. Table III provides the list of protocols used in the experiments along with their types (text/binary, open/proprietary). For the subsequent discussion we name the first dataset as Private Dataset, second dataset as Public-1 Dataset and third dataset as Public-2 Dataset.

$BitCoding$ is a supervised traffic classification technique where flows of applications are used to generate signatures in the first phase and subsequently used for detecting applications. Thus we divided each dataset into nearly equal parts (approximately 50% for every protocol) and used the first part for generating signatures and the other part for testing the detection performance. The statistics of flows in each case (training and testing) for every protocol and for each dataset are shown in the tables IV, V and VI.

TABLE II

Module-Wise Complexity of $BitCoding$

Module Name	Complexity	Explanation
Flow-Reconstructor	O($l \times p$)	l= number of flows, p = number of packets in the trace.
Bit-Extractor	O(1)	Constant number of bits to be read from the payload training flow
Bit-Signature	O(K)	For K flows of an application
RLE	O(X)	Need to read n bits for an application and there are X applications
$TCCA$ Generator	O(X)	Convert RLE into a graph for X applications
Flow-Classifier	O($m \times n$)	m= number of machines and n = number of bits in the payload of test flow

TABLE III

Application Protocols Used in the Experiments

Abbreviation	Protocol	Type	Proprietariness
BACnet	Building Automation and Control network	Binary	ASHRAE
BitTorrent	Bit torrent protocol	Text	No
BJNP	Used to communicate with printer	Binary	Canon
Bootp	Bootstrap protocol	Binary	No
CUPS	Common Unix Printing System	Text	Apple Inc.
DNS	Domain Name System	Binary	No
Dropbox	Dropbox LAN Sync protocol	Text	Dropbox
GsmIp	GSM over Internet protocol	Text	No
HTTP	Hyper Text Transfer Protocol	Text	No
Kerberos	Kerberos protocol	Binary	No
MWBP	Microsoft Windows Browsing Protocol	Text	Microsoft
NBNS	NetBIOS Name Service	Binary	No
NBSS	NetBIOS Session Service	Binary	No
NTP	Network Time Protocol	Binary	No
POP	Post Office Protocol	Text	No
QUIC	Quick UDP Internet Connections	Binary	No
RPC	Remote Procedure Call	Binary	No
SIP	Session Initiation Protocol	Text	No
SMTP	Simple Mail Transfer Protocol	Text	No
SSH	Secure Shell	Binary	No

TABLE IV

Private Dataset Statistics

Protocol	TCP/UDP	Flows for Training	Size of Training (MB)	Flows for Testing	Size of Testing (MB)
BitTorrent	TCP	01578	245.8	01582	150.4
DNS	UDP	65158	005.7	65528	005.7
Dropbox	UDP	02256	098.2	02276	153.4
HTTP	TCP	97668	220.4	97756	328.3
SIP	UDP	01218	194.1	01280	191.4
SMTP	TCP	01194	010.1	01216	022.9
SSH	TCP	02208	006.2	02212	006.2
Total	-	**171280**	**780.5**	**171850**	**858.3**

B. Evaluation

We implemented $BitCoding$ as a standalone Java program with JNetPacp programming library [3]. JNetPacp library also has a module for bidirectional flow reconstruction. We used this method for reconstructing flows from packets in our experiments.

We used training portion of each protocol taken from each of the three datasets and generated signatures for every protocol. These signatures will be used for future detection/classification of traffic flows. We use *Recall* as a measure of classification performance for $BitCoding$ (In

TABLE V
PUBLIC-1 DATASET STATISTICS

Protocol	TCP/UDP	Flows for Training	Size of Training (MB)	Flows for Testing	Size of Testing (MB)
BACnet	UDP	00018	000.097	00022	000.074
BJNP	UDP	00068	000.026	00076	000.031
Bootp	UDP	00162	004.400	00172	004.500
CUPS	UDP	00090	000.107	00094	000.218
DNS	UDP	50938	012.900	51700	011.100
Dropbox	UDP	00050	000.109	00052	000.319
HTTP	TCP	35928	151.100	35936	133.600
MWBP	UDP	00016	000.565	00014	000.574
NBNS	UDP	01964	007.800	01964	007.500
NTP	UDP	00402	000.652	00402	000.141
QUIC	UDP	00186	000.110	00254	000.115
SMTP	TCP	01040	010.100	01042	009.900
Total	-	**90862**	**187.996**	**91728**	**168.072**

TABLE VI
PUBLIC-2 DATASET STATISTICS

Protocol	TCP/UDP	Flows for Training	Size of Training (MB)	Flows for Testing	Size of Testing (MB)
Bootp	UDP	0182	00.080	0182	0.096
DNS	UDP	1926	00.865	1916	1.200
GsmIp	TCP	0018	00.007	0018	0.015
HTTP	TCP	0514	04.800	0506	9.000
Kerberos	UDP	1338	01.600	1344	1.900
NBNS	UDP	0578	00.853	0580	0.680
NBSS	TCP	0754	02.700	0746	3.900
NTP	UDP	0400	00.145	0404	0.648
POP	TCP	0112	00.035	0114	0.036
RPC	TCP	0014	00.020	0014	0.141
Total	-	**5836**	**11.105**	**5824**	**17.616**

this paper we use $Recall$ as a representative evaluation metric as the main focus is to study the robustness of signatures generated and understanding the signature overlap between different protocols). *Recall* is the ratio of flows correctly labeled as a particular application to the total number of flows belonging to that application. Equation 1 defines this.

$$Recall = \frac{TP}{TP + FN} \tag{1}$$

We perform three experiments with the above three datasets to assess the detection performance of $BitCoding$. In our experiments we select first 40 ($n = 40$) bits of the flow for generating signatures. Another point worth noting with $BitCoding$ is, if there are few flows which are corrupted or incomplete, these may influence the signature generation due to mismatch in bit positions. One way to address this issue is to filter such flows from training set or use a threshold on percentage of times a particular bit is either 1 or 0. In our experiments we used second approach by setting threshold to 99%.[4] The three experiments and results obtained are furnished below.

1) Homogeneous Experiments: In the homogeneous experiments we used the training portion of each of the three datasets (Private, Public-1 and Public-2) to generate signatures for each of the protocols. The other portion of the same dataset is used as the testing dataset. The $Recall$ obtained for each protocol is shown in Table VII. From these results we can notice that, $BitCoding$ has an average rate of $Recall$ greater than 99%. This experiment suggests that $BitCoding$ performs very well if the testing dataset is from the same site as that of training dataset.

2) Heterogeneous Experiments: In heterogeneous set of experiments we used the training portion of each dataset to generate signatures for application protocols in that dataset. The testing portion of the other two datasets are used to calculate the $Recall$. The idea is to assess robustness of signatures generated in detecting applications when presented with dataset collected from other sites.[5] Table VIII, Table IX and Table X show the $Recall$ for the cases where the signatures are generated with only the training portion of the Private, Public-1 and Public-2 Datasets and tested with testing portion of the other two datasets respectively. It is worth noting here that, in this set of experiments testing is done only for the overlapping set of protocols (training and testing). This experiment gives an idea of how robust the generated signatures are "in not including site specific information in the signatures". For example, some protocols include the name of hosts, user names, etc in the communication. If the training data is collected from a particular site there is a chance that these keywords also be part of invariant bits of signature which should be avoided. We can notice from the three tables that in all the cases of cross evaluation $Recall$ is over 90% (with many having 95 to 100%) which indicates that the generated signatures are indeed robust (with little compromise in performance) in detecting applications when presented with data from other sites.

3) Grand Experiments: In grand experiments we merged the training portion of all the protocols of all the three datasets and generated a grand training dataset. This grand training dataset is used to generate the signatures for each protocol. For testing purpose we used the three dataset's respective testing portions and also a combined dataset (of all the three testing portions) to evaluate the performance. The signatures generated after the run length encoding (RLE) in this case are shown in Table XI. These signatures are subsequently converted into $TCCA$. One such $TCCA$ for DNS protocol is shown in Figure 7. The time taken to generate the signatures and converting them to $TCCA$ in each case are also shown in Table XII.[6] The time taken to generate the signatures depends on the number of flows in the training dataset as from every flow bits are extracted and processed to derive the signature. We can notice that $BitCoding$ is very fast in processing flows and generating signatures. Even for HTTP with 134110 number of flows in training set, it is taking less than a minute to generate the signature. Table XIII shows the $Recall$ rate for four different experiments. We can notice that the results are similar to the previous cases (with many having $Recall$ close to 100%). We can also notice that, $Recall$ has improved for many protocols compared to heterogeneous experiments, indicating signatures are more robust.

[4]All results shown in this paper are with this threshold, although there are only few protocols which had incomplete/corrupted flows.

[5]All three datasets are collected from different sites

[6]We show the signatures and TCCA for this case as this experiment is comprehensive

TABLE VII

RECALL FOR HOMOGENEOUS EXPERIMENTS. (a) PRIVATE DATASET. (b) PUBLIC-1 DATASET. (c) PUBLIC-2 DATASET

(a)

Protocol	Recall (in%)
BitTorrent	100.00
DNS	099.69
Dropbox	100.00
HTTP	099.23
SIP	097.97
SMTP	100.00
SSH	100.00

(b)

Protocol	Recall (in%)
BACnet	095.45
BJNP	100.00
Bootp	100.00
CUPS	093.61
DNS	099.93
Dropbox	100.00
HTTP	097.54
MWBP	100.00
QUIC	100.00
NBNS	099.59
NTP	100.00
SMTP	100.00

(c)

Protocol	Recall (in%)
Bootp	100.00
DNS	099.79
GsmIp	100.00
HTTP	100.00
Kerberos	100.00
NBNS	098.97
NBSS	098.91
NTP	100.00
POP	100.00
RPC	100.00

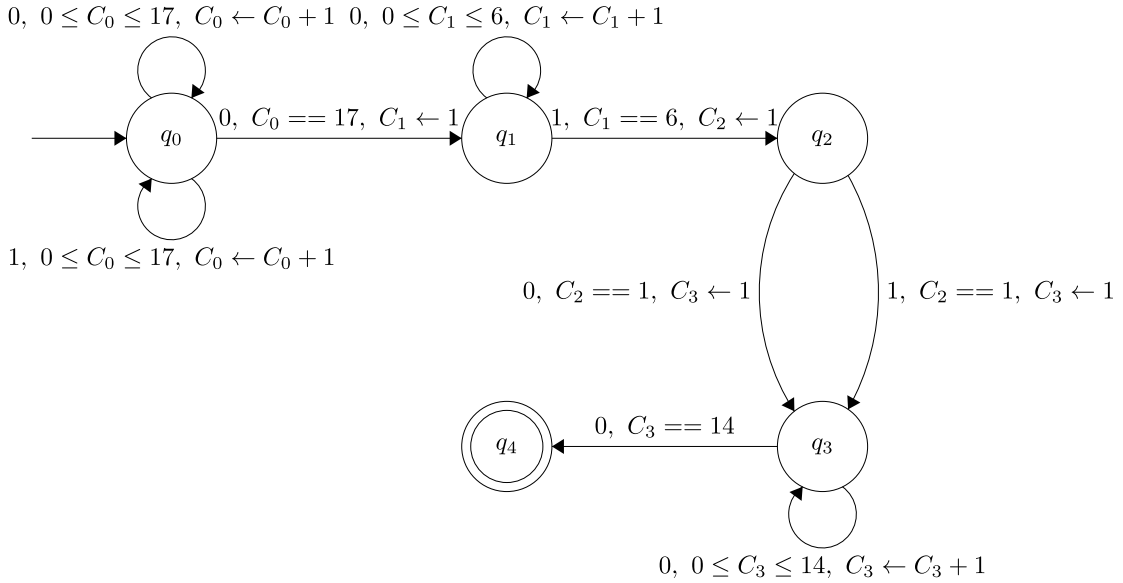

Fig. 7. $TCCA$ generated for DNS with 40 bit length compressed signature "17*6Z1W1*15Z".

TABLE VIII

RECALL FOR TRAINING WITH PRIVATE DATASET. (a) PUBLIC-1 DATASET. (b) PUBLIC-2 DATASET

(a)

Protocol	Recall (in%)
DNS	100.00
Dropbox	100.00
HTTP	099.52
SMTP	100.00

(b)

Protocol	Recall (in%)
DNS	095.19
HTTP	093.28

TABLE IX

RECALL FOR TRAINING WITH PUBLIC-1 DATASET. (a) PRIVATE DATASET. (b) PUBLIC-2 DATASET

(a)

Protocol	Recall (in%)
DNS	096.39
Dropbox	100.00
HTTP	095.96
SMTP	100.00

(b)

Protocol	Recall (in%)
Bootp	100.00
DNS	094.99
HTTP	091.69
NBNS	099.68
NTP	100.00

TABLE X

RECALL FOR TRAINING WITH PUBLIC-2 DATASET. (a) PRIVATE DATASET. (b) PUBLIC-1 DATASET

(a)

Protocol	Recall (in%)
DNS	096.39
HTTP	093.89

(b)

Protocol	Recall (in%)
Bootp	090.69
DNS	099.93
HTTP	099.93
NBNS	099.59
NTP	099.51

VII. MEASURING ROBUSTNESS OF SIGNATURES WITH CROSS APPLICATION FLOWS

As mentioned previously in section IV signatures of one application might match with the flows of other applications. The chances of this cross signature matching increases as number of protocols increase particularly with short signature lengths. To understand the robustness and uniqueness of signatures generated by $BitCoding$, we performed an experiment with nXn signature matching. In this experiment we evaluate the flows of one application protocol with signatures of all other protocols. For this experiment we used the grand experiments dataset (both training and testing). The results of these experiments are summarized in the Table XIV. The rows in the table show the number of flows of type indicated in the column are matched against the signature of protocol in the row. For example the third column BitTorrent has 1582 number of flows in testing dataset and 1582 of them are matched with BitTorrent signature. Similarly the second row first column has a value of 0 indicating none of BJNP flows are

TABLE XI

SIGNATURES GENERATED FOR DIFFERENT PROTOCOLS WITH $n = 40$ BITS IN GRAND EXPERIMENTS

Protocol	Signature
BACnet	1W6Z1W4Z4*11Z1W1*1Z1*1Z1*1Z1*1Z2*1Z1*
BJNP	1Z1W1Z1W5Z1W2Z3W2Z1W2Z1W1Z1W2Z1W1Z1W11Z1W
BitTorrent	3Z1W2Z2W1Z1W4Z1W2Z2W1Z1W2Z1W1Z3W1Z1W3Z1W1Z1W1Z1W2Z
Bootp	6Z2*7Z1W5Z2W8Z9*
CUPS	2Z2W2*1Z1*2Z2W1*4Z1*1W5*1Z1*1W5*1Z1*1W1*1Z3*
DNS	17*6Z1W1*15Z
Dropbox	1Z4W1Z2W2Z1W3Z1W2Z2W1Z1W4Z2W1Z4W1Z3W2Z2W
GsmIp	1Z1W1Z2W1Z1W2Z1W4Z1W2Z1W1Z2W4Z1W3Z1W9Z1W
HTTP	1Z1W1Z5*1Z1W2Z1*1W1*1W1Z1W1Z5*1Z6*2Z2*1Z4*
Kerberos	1Z8*4Z2*5Z20*
MWBP	3Z1W3Z1W4Z1*1Z1W1Z16*2W6Z
NBNS	17*1Z1*1Z2*1Z1W3Z1*12Z
NBSS	1*6Z1*16Z16*
NTP	7*1W5Z3*3Z13*8Z
POP	2Z1W1Z1W1Z2W1Z1W2Z4W1Z1W2Z1W1Z2W2Z1W6Z1W2Z1*1W2Z
QUIC	1Z15*6Z1W1Z16*
RPC	1W24Z1W4*2Z2W6*
SIP	1Z1W1Z5*1Z1W1Z5*1Z1W1Z1*1Z3*1Z2*1Z1W2*1W1Z7*
SMTP	2Z2W2Z1W3Z2W2Z1W3Z2W6Z1W6Z1*1W1Z3*1Z
SSH	1Z1W1Z1W2Z2W1Z1W1Z1W2Z2W1Z1W2Z1W5Z1W1Z2W1Z1W2Z2W2Z1W1Z

TABLE XII

SIGNATURE GENERATION TIME FOR GRAND EXPERIMENTS

Protocol	Training time (in seconds)
BACnet	00.445
BJNP	00.414
BitTorrent	04.018
Bootp	00.704
CUPS	00.500
DNS	34.500
Dropbox	03.367
GsmIp	00.288
HTTP	46.416
Kerberos	01.226
MWBP	00.373
NBNS	01.820
NBSS	00.871
NTP	00.815
POP	00.368
QUIC	00.493
RPC	00.237
SIP	03.629
SMTP	01.587
SSH	01.502
Total time	**103.573**

TABLE XIII

RECALL FOR TRAINING WITH GRAND DATASET

Protocol	Grand Dataset	Private Dataset	Public-1 Dataset	Public-2 Dataset
BACnet	095.45	-	095.45	-
BJNP	100.00	-	100.00	-
BitTorrent	100.00	100.00	-	-
Bootp	100.00	-	100	100
CUPS	093.61	-	093.61	-
DNS	099.75	099.69	100	097.91
Dropbox	100.00	100.00	100.00	-
GsmIp	100.00	-	-	100.00
HTTP	098.18	098.44	097.42	091.69
Kerberos	100.00	-	-	100.00
MWBP	100.00	-	100.00	-
NBNS	099.44	-	099.69	098.96
NBSS	098.90	-	-	98.90
NTP	100.00	-	100.00	100.00
POP	100.00	-	-	100.00
QUIC	100.00	-	100.00	-
RPC	100.00	-	-	100.00
SIP	097.96	097.96	-	-
SMTP	100.00	100.00	100.00	-
SSH	100.00	100.00	-	-

matched against signatures of BACnet. Similar interpretation is done for other cases too. We can notice that there are indeed some cases where signatures of one protocol match with flows of other protocol (shown in red color). For example out of 119144 flows of DNS 116646 flows are matching with NBNS protocol.

As mentioned in section IV we identify *weak signatures* by computing the $Relaxed\ Hamming\ Distance\ (RHD)$ between the signatures generated (although we considered few other distance measures for experiments, we do not show them here for the sake of brevity). We computed the RHD values between every pair of application protocol signatures generated in the above case. The RHD values obtained are shown in Table XV. We can notice that, most of the cross signature matching is happening with those applications whose signatures have a zero RHD values (Ex. DNS and Kerberos). However, there are few cross signature matching even with non-zero distance values too (28 HTTP flows matching with CUPS). We manually screened these flows and noticed that, these are either incomplete or corrupted just like the ones found in training set.

VIII. HOW THE NUMBER OF BITS AND NUMBER OF FLOWS AFFECT SIGNATURE QUALITY

In this section we study the impact of two important parameters on the quality of signatures generated. First study is on increasing the number of bits used for signature generation and second is the number of flows used for signature generation. These two are elaborated in the next two subsections.

A. Effect of Increasing the Number of Bits

As we noticed in the last section (with nXn evaluation experiment and RHD calculation) there are indeed few cases where flows of one type of protocol are matched with signatures of other type of protocol leading to incorrect classification or at-best a guess of application (matching with more than application signature). In this section we provide results of the experiments done to find the effect of increasing the signature bits. We performed experiments by increasing the number of bits from 40 to 80 and 120 and evaluated the number of cross signature matches for those cases which showed cross matching in previous experiment (nXn evaluation). Table XVI shows the new signatures generated with 80 bits of payload extract. Table XVII shows the number of signatures matched for the protocol type which had issue with 40 bits signature. We can notice that even with 80 bits too the cross signature matching continued, however the number of cross matching cases decreased compared to the previous case. Similarly Table XVIII shows the signatures generated with 120 bits of payload extract (for only those cases which had cross matching even with 80 bits signature) and Table XIX shows the cross signature matching performance. We can notice that with signature length of 120 bits the cross signature matching issue is completely addressed. We computed the RHD values between all pairs of protocols and the distances are shown in Table XX. We can notice that there are no zero RHD values in this case which also justify the no cross signature matching. With this experiment we can conclude that for different protocols we need signatures of different lengths. For correct detection of application, this length needs to be empirically found out. One way to address this is to generate a signature for an application and test it for overlap with RHD values and increase the length if there is a overlap.

TABLE XIV

MATRIX FOR nXn EVALUATION SCENARIO WITH $n = 40$ BITS

	BACnet (22)	BJNP (76)	BitTorrent (1582)	Bootp (354)	CUPS (94)	DNS (119144)	Dropbox (2328)	GsmIp (18)	HTTP (134198)	Kerberos (1344)	MWBP (14)	NBNS (2544)	NBSS (728)	NTP (806)	POP (114)	QUIC (254)	RPC (14)	SIP (1280)	SMTP (2258)	SSH (2212)	Not Classified
BACnet	21	0	0	0	0	16	0	0	0	0	0	14	0	0	0	0	0	0	0	0	1
BJNP	0	76	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
BitTorrent	0	0	1582	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Bootp	0	0	0	354	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
CUPS	0	0	0	0	88	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	6
DNS	0	0	0	0	0	118848	0	0	0	0	0	0	0	0	0	0	0	0	0	0	296
Dropbox	0	0	0	0	0	0	2328	0	0	0	0	0	0	0	0	0	0	0	0	0	0
GsmIp	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0	0	0
HTTP	0	0	0	0	0	0	0	0	131756	0	0	0	0	0	0	0	0	0	0	0	2442
Kerberos	0	0	0	0	0	1710	0	0	0	1344	0	36	0	408	0	0	0	0	0	0	0
MWBP	0	0	0	0	0	4	0	0	0	0	14	0	0	0	0	0	0	0	0	0	0
NBNS	0	0	0	0	0	116646	0	0	0	0	0	2530	0	0	0	0	0	0	0	0	14
NBSS	0	0	0	0	0	0	0	0	0	0	0	0	720	0	0	0	0	0	0	0	8
NTP	0	0	0	0	0	1792	0	0	0	0	0	0	42	806	0	0	0	0	0	0	0
POP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	114	0	0	0	0	0	0
QUIC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	254	0	0	0	0	0
RPC	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	0	0	0	0
SIP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1254	0	0	26
SMTP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2258	0	0
SSH	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2212	0

TABLE XV

MATRIX FOR nXn RELAXED HAMMING DISTANCE WITH $n = 40$ BITS

	BACnet	BJNP	BitTorrent	Bootp	CUPS	DNS	Dropbox	GsmIp	HTTP	Kerberos	MWBP	NBNS	NBSS	NTP	POP	QUIC	RPC	SIP	SMTP	SSH
BACnet	-	9	13	4	7	2	18	13	4	1	3	1	0	0	12	2	3	6	11	14
BJNP	9	-	14	12	10	7	20	8	1	3	7	5	9	5	14	2	12	4	18	17
BitTorrent	13	14	-	14	7	10	14	12	5	3	5	7	8	6	12	5	11	6	14	13
Bootp	4	12	14	-	7	3	17	13	4	1	3	2	3	0	11	1	5	4	11	12
CUPS	7	10	7	7	-	3	4	10	8	3	6	2	5	4	10	1	7	5	2	8
DNS	2	7	10	3	3	-	15	7	1	0	2	0	1	0	6	2	4	3	5	10
Dropbox	18	20	14	17	4	15	-	14	8	3	10	13	10	8	20	4	15	5	14	11
GsmIp	13	8	12	13	10	7	14	-	2	3	8	6	9	4	16	4	12	3	14	13
HTTP	4	1	5	4	8	1	8	2	-	3	5	1	5	2	2	1	7	1	9	4
Kerberos	1	3	1	1	3	0	3	3	3	-	0	0	0	0	4	0	1	2	4	4
MWBP	3	7	5	3	6	2	10	8	5	0	-	2	2	2	9	0	4	3	7	10
NBNS	1	5	7	2	2	0	13	6	1	0	2	-	1	0	5	2	3	3	4	9
NBSS	0	9	8	3	5	1	10	9	5	0	2	1	-	0	12	1	0	3	8	9
NTP	0	5	6	0	4	0	8	4	2	0	2	0	0	-	5	0	3	2	5	6
POP	12	14	12	11	10	6	20	16	2	4	9	5	12	5	-	3	16	5	16	17
QUIC	2	2	5	1	1	2	4	4	1	0	0	2	1	0	3	-	2	1	3	3
RPC	3	12	11	5	7	4	15	12	7	1	4	3	0	3	16	2	-	6	5	11
SIP	6	4	6	4	5	3	5	3	1	2	3	3	3	2	5	1	6	-	8	1
SMTP	11	18	14	11	2	5	14	14	9	4	7	4	8	5	16	3	11	8	-	14
SSH	14	17	13	12	8	10	11	13	4	4	10	9	9	6	17	3	15	1	14	-

TABLE XVI

SIGNATURES GENERATED FOR DIFFERENT PROTOCOLS WITH $n = 80$ BITS

Protocol	Signature
BACnet	1W6Z1W4Z4*1IZ1W1*1ZI*1ZI*1ZI*1Z2*1ZI*2Z2*1ZI*2Z2*3W1*3W1*1W1*4W1*1Z3*1ZI*1Z2W6*
CUPS	2Z2W2*1ZI*2Z2W1*4Z1*1W5*1ZI*1W5*1ZI*1W1*1Z3*1ZI*1W1*1Z3*1ZI*1W2Z1*1ZI*1ZI*1W2*1Z2*1ZI*1W1*5ZI*1W2*2ZI*
HTTP	1ZIW1Z5*1ZIW2ZI*1W1*1W1ZIW1Z5*1Z6*2Z2*1Z4*1Z7*1Z7*1Z7*1Z7*
DNS	17*6Z1W1*22Z1W14Z1*17Z
Kerberos	1Z8*4Z2*5Z31*1Z28*
MWBP	3Z1W3Z1W4Z1*1ZIW1ZI6*2W6ZIW1ZIW1ZIW5ZI*1ZI*1Z3*1ZI*1Z4*8ZIW3ZIW1ZIW1Z
NBNS	17*1ZI*1Z2*1ZIW3ZI*19ZI*15ZI*16Z
NTP	7*1W5Z3*3Z13*15ZI*2ZI4*15ZI*

As *BitCoding* uses RLE to compress the signature bits for efficient representation, we computed the compression ratio achieved by RLE for signatures of different lengths. The compression ratio is the ratio of length of signature in the RLE encoded signature to the length of signature before RLE. This is given by Equation 2. Figure 8 shows the compression ratio achieved by RLE for 5 different protocols.[7] We can notice that increasing the length of signature bits achieve better compression ratio. Similar observations are made for other protocols too.

$$Compression\ Ratio = \frac{Length\ After\ RLE}{Length\ Before\ RLE} \quad (2)$$

[7]We show for only 5 protocols here for clarity

TABLE XVII

MATRIX FOR nXn EVALUATION SCENARIO WITH $n = 80$ BITS

	BACnet	CUPS	DNS	HTTP	Kerberos	MWBP	NBNS	NTP	Not Classified
BACnet	20	0	0	0	0	0	0	0	0
CUPS	0	94	0	14	0	0	0	0	0
DNS	0	0	117400	0	0	0	0	0	1744
HTTP	0	0	0	131756	0	0	0	0	2442
Kerberos	0	0	1710	0	1338	0	36	382	6
MWBP	0	0	0	0	0	14	0	0	0
NBNS	0	0	116646	0	0	0	2530	0	14
NTP	0	0	1792	0	0	0	42	796	10

B. Effect of Number of Flows on Signature Quality

As we generate signatures from the payloads of training application flows, the number of flows present in the training

TABLE XVIII

SIGNATURES GENERATED FOR DIFFERENT PROTOCOLS WITH $n = 120$ Bits

Protocol	Signature
CUPS	2Z2W2*1Z1*2Z2W1*4Z1*1W5*1Z1*1W5*1Z1*1W1*1Z3*1Z1*1W1*1Z3*1Z1*1W2Z1*1Z1*1Z1*1W2*1Z2*1Z1*1W1*5Z1*1W2*2Z1*1Z1*1W2*1Z2*1Z1*1W5*1Z1*1W5*2Z1W5*2Z1W2*3W
DNS	17*6Z1W1*22Z1W14Z1*36Z5*1Z7*1Z7*
Kerberos	1Z8*4Z2*5Z31*1Z39*1Z5*1Z1*2Z4*1Z3*1Z2*5Z3*
NBNS	17*1Z1*1Z2*1Z1W3Z1*19Z1*15Z1*31Z1*2Z1W6Z1W3Z3*1Z1W1Z5*
NTP	7*1W5Z3*3Z13*15Z1*2Z14*15Z18*1Z10*1Z3*3Z1*2Z2*

TABLE XIX

MATRIX FOR nXn EVALUATION SCENARIO WITH $n = 120$ Bits

	CUPS	DNS	Kerberos	NBNS	NTP	Not Classified
CUPS	94	0	0	0	0	0
DNS	0	117374	0	0	0	1770
Kerberos	0	0	1226	0	0	118
NBNS	0	0	0	2530	0	14
NTP	0	0	0	0	788	18

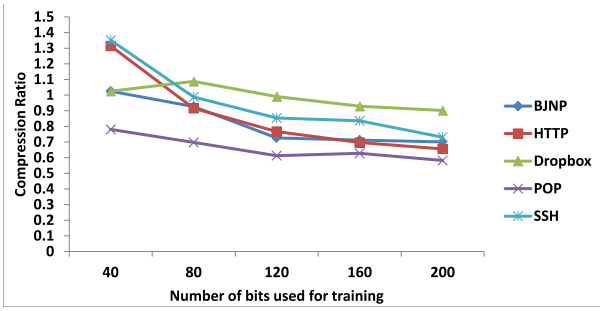

Fig. 8. Compression ratio achieved for different protocols.

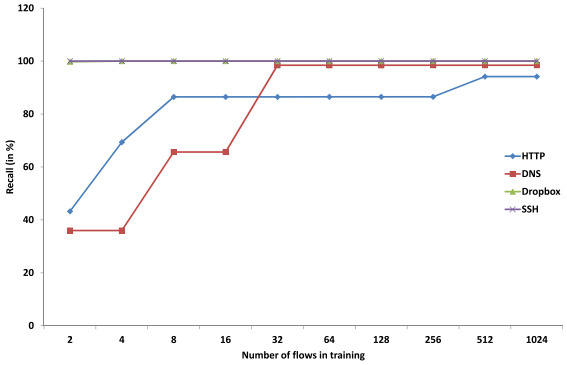

Fig. 9. Effect of number of flows on *Recall*.

dataset is an important parameter for generating quality signatures. It is a natural question to ask how many flows are required to generate good signatures. To understand this, we performed an experiment with different number of flows to generate the signatures and tested the performance using our grand dataset. Figure 9 show the effect on the *Recall* with different number of flows for four protocols. We can notice that for two of the protocols (Dropbox and SSH) the *Recall* rate is 100% from the beginning. However with other two protocols it is gradually increasing and nearing 100% after 256 and 512 number of flows. The same trend is shown with other protocols too. Thus we can infer that, for few protocols small number of flows are sufficient for generating the signatures and for others relatively more (few hundred) are required to generate good signatures.

IX. COMPARISON AND DISCUSSION

In this section we describe the comparison results of *BitCoding* with two other methods found in literature.

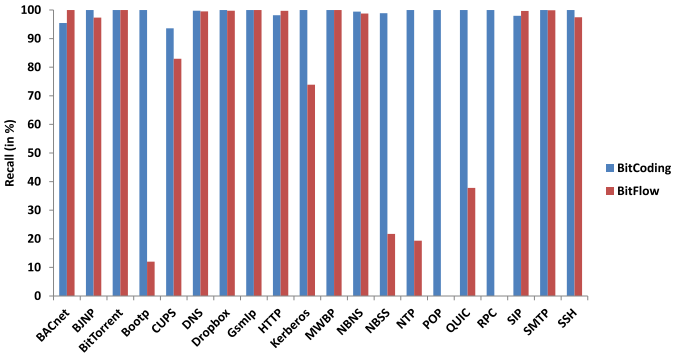

Fig. 10. *Recall* comparison for *BitCoding* and *BitFlow* with 40 bit signature.

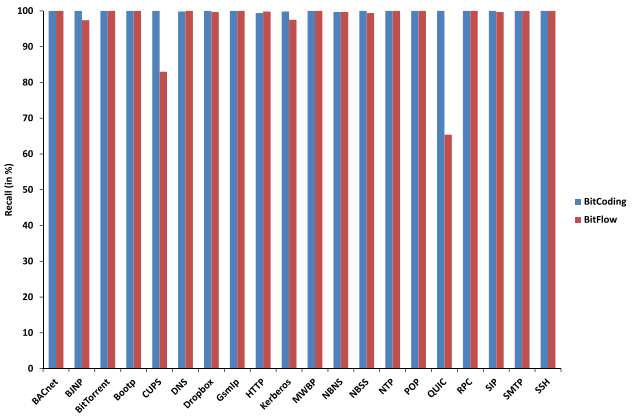

Fig. 11. *Recall* comparison for *BitCoding* and *BitFlow* with 80 bit signature.

A. Comparison With *BitFlow*

We compared the performance of *BitCoding* with a recent method *BitFlow* [29]. *BitFlow* extracts first n bits from each flow of an application in training dataset and use AdaBoost machine learning algorithm for learning the bit patterns. The experiments described in [29], used first 32 bits for TCP flows and first 40 bits for UDP flows. In our experiments we used the three bit lengths (40, 80 and 120 for both the methods) as in our previous experiments and compared the performance. Figures 10, 11, 12 show performance comparison in terms of *Recall* for these experiments. We can notice that, *BitCoding* performs consistently well in comparison to *BitFlow* for the 40 bits experiments with some of the protocol flows are not at all identified by *BitFlow*. For experiments with 80 and 120 bits the performance of *BitFlow* has also improved. One of the reason for under-performance of *BitFlow* for 40 bits experiment is because of uneven number of flow samples in training dataset and other reason being similar bit patterns of protocols which exhibit higher similarity when compared with a similarity measurement (which is common with machine learning algorithms). These issues are bound to happen in any dataset collected from any network. We believe that,

TABLE XX
MATRIX FOR nXn RELAXED HAMMING DISTANCE WITH $n = 120$ Bits

	BACnet	BJNP	BitTorrent	Bootp	CUPS	DNS	Dropbox	GsmIp	HTTP	Kerberos	MWBP	NBNS	NBSS	NTP	POP	QUIC	RPC	SIP	SMTP	SSH
BACnet	-	29	35	7	14	16	34	29	13	4	14	17	13	8	30	2	15	23	19	35
BJNP	29	-	54	12	23	9	55	19	20	3	17	10	17	9	54	2	13	29	29	55
BitTorrent	35	54	-	33	17	39	44	47	19	19	39	42	29	33	28	5	42	27	28	45
Bootp	7	12	33	-	13	4	35	15	9	1	9	3	4	4	30	1	5	17	15	35
CUPS	14	23	17	13	-	13	12	21	11	7	19	14	14	12	23	1	19	12	7	25
DNS	16	9	39	4	13	-	48	11	12	2	13	1	8	6	37	2	7	22	15	29
Dropbox	34	55	44	35	12	48	-	45	22	16	38	49	26	26	50	4	41	22	24	43
GsmIp	29	19	47	15	21	11	45	-	18	7	26	16	19	14	57	4	21	26	26	47
HTTP	13	20	19	9	11	12	22	18	-	5	20	11	10	14	15	4	17	6	15	17
Kerberos	4	3	19	1	7	2	16	7	5	-	6	4	2	1	19	2	1	8	8	20
MWBP	14	17	39	9	19	13	38	26	20	6	-	15	17	9	42	2	11	25	15	41
NBNS	17	10	42	3	14	1	49	16	11	4	15	-	9	5	39	2	9	23	16	43
NBSS	13	17	29	4	14	8	26	19	10	2	17	9	-	5	29	1	4	14	15	30
NTP	8	9	33	4	12	6	26	14	14	1	9	5	5	-	30	2	6	16	16	27
POP	30	54	28	30	23	37	50	57	15	19	42	39	29	30	-	3	47	23	29	47
QUIC	2	2	5	1	1	2	4	4	4	2	2	2	1	2	3	-	2	2	3	3
RPC	15	13	42	5	19	7	41	21	17	1	11	9	4	6	47	2	-	25	18	45
SIP	23	29	27	17	12	22	22	26	6	8	25	23	14	16	23	2	25	-	21	30
SMTP	19	29	28	15	7	15	24	26	15	8	15	16	15	16	29	3	18	21	-	25
SSH	35	55	45	35	25	39	43	47	17	20	41	43	30	27	47	3	45	30	25	-

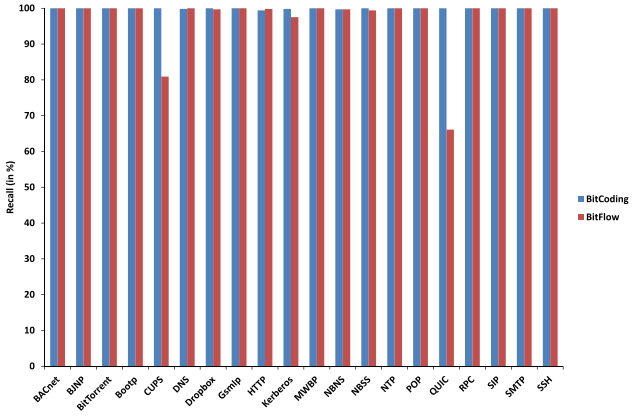

Fig. 12. *Recall* comparison for $BitCoding$ and $BitFlow$ with 120 bit signature.

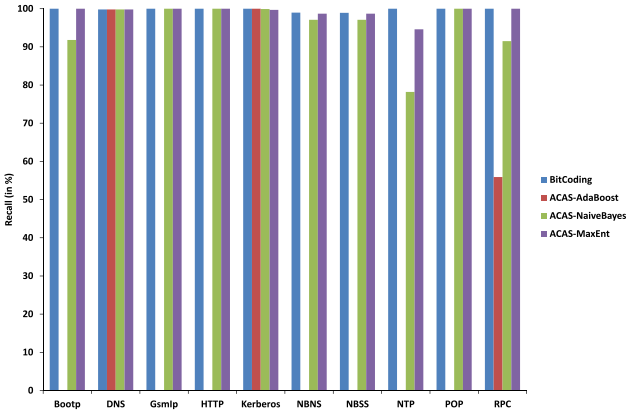

Fig. 13. *Recall* comparison for $BitCoding$ and $ACAS$.

this issue will further aggravate with increase in number of application protocols. In contrast, $BitCoding$ can generate signatures with limited number of flows. Further, $BitCoding$ can work with variable number of bits for each protocol which is not possible for machine learning algorithms. When $BitCoding$ is having variable length signatures (with few applications showing overlap with shorter signature lengths), initial matching can be done with shorter version and later with increased signature length.

B. Comparison With ACAS

We compared $BitCoding$ with another classical method $ACAS$ [13] which works with byte level content. $ACAS$ generates feature vector from first 'n' bytes of the payload. It generates a binary feature vector of size $n * 256$ where ASCI position value (1-256) of a byte at i^{th} position sets the binary value at $256 * i + c[i]$ where $c[i]$ is the position value of byte at i^{th} position in payload. Feature vectors so generated are classified using machine learning algorithms (AdaBoost, NaiveBayes, MaxEntropy). The authors of the paper experimented and reported results for first 64 bytes of payload. We repeated this experiment with public-2 Dataset and compared the performance with 40 bit signatures of $BitCoding$. Figure 13 shows the $Recall$ rate for the two approaches. We can notice that $BitCoding$ performs better for many protocols but using orders of magnitude less time and data.

C. Discussion

The experiments with $BitCoding$ show impressive performance in classifying different types of applications. To the best of our knowledge currently there are no commercial/open source tools available for performing DPI and classification at the bit-level. The extensive experimental results reported in this paper can be a motivation for the development of such tools. We are currently working towards implementing $BitCoding$ on hardware using re-configurable Field Programmable Gate Array (FPGA) boards. Another interesting study would be of performance comparison of DPI at byte-level and bit-level to understand which is computationally more intensive. Also there are many optimizations done to regular expression or string matching techniques for byte-level DPI. One optimization for bit-level signature in the form of $TCCA$ is described in this paper. It would be an useful study to explore other possible optimizations.

X. CONCLUSION

In this paper we presented $BitCoding$ a bit-level signature generation method from application payload for traffic classification. $BitCoding$ uses first n bits of data extracted from

the payloads of bidirectional flow. Subsequently it encodes the signatures with run length encoding and transform into a state transition machine for efficient representation and comparison. With extensive experimentation we showed that, bit-level signatures generated by $BitCoding$ are robust in detecting applications and can be ported from site to site with little compromise in detection performance. Further increasing the signature length increases the detection capability and avoids cross signature matching and it will also help achieve better compression and hence shorter signature representation.

REFERENCES

[1] *Digital Corpora*. Accessed: Jan. 4, 2018. [Online]. Available: http://digitalcorpora.org/

[2] *eDonkey*. Accessed: Jan. 4, 2018. [Online]. Available: http://en.wikipedia.org/wiki/edonkey2000

[3] *JnetPacap*. Accessed: Jan. 6, 2018. [Online]. Available: http://jnetpcap.sourceforge.net/

[4] *Tor Project*. Accessed: Jan. 6, 2018. [Online]. Available: https://www.torproject.org/

[5] *BitTorrent*. Accessed: Jan. 6, 2018. [Online]. Available: http://www.bittorrent.com

[6] *eMule*. Accessed: Jan. 6, 2018. [Online]. Available: http://www.emule-project.net/

[7] *Netresec*. Accessed: Jan. 4, 2018. [Online]. Available: http://www.netresec.com/?page=pcapfiles

[8] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescapé, "Multi-classification approaches for classifying mobile app traffic," *J. Netw. Comput. Appl.*, vol. 103, no. 1, pp. 131–145, Feb. 2018.

[9] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for Internet traffic classification," *IEEE Trans. Neural Netw.*, vol. 18, no. 1, pp. 223–239, Jan. 2007.

[10] A. Bremler-Barr, D. Hay, and Y. Koral, "CompactDFA: Scalable pattern matching using longest prefix match solutions," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 415–428, Apr. 2014.

[11] A. Dainotti, W. de Donato, A. Pescapé, and P. S. Rossi, "Classification of network traffic via packet-level hidden Markov models," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Nov./Dec. 2008, pp. 1–5.

[12] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proc. SIGCOMM Workshop Mining Netw.*, 2006, pp. 281–286.

[13] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: Automated construction of application signatures," in *Proc. ACM SIGCOMM Workshop Mining Netw. Data*, 2005, pp. 197–202.

[14] N. Hubballi, S. Biswas, and S. Nandi, "Sequencegram: N-Gram modeling of system calls for program based anomaly detection," in *Proc. 3rd Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2011, pp. 1–10.

[15] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese, "Network monitoring using traffic dispersion graphs (TDGs)," in *Proc. 7th ACM SIGCOMM Conf. Internet Meas. (ACM)*, 2007, pp. 315–320.

[16] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, "Transport layer identification of P2P traffic," in *Proc. 4th ACM SIGCOMM Conf. Internet Meas. (IMC)*, 2004, pp. 121–134.

[17] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2005, pp. 229–240.

[18] Y. Lim *et al.*, "Internet traffic classification demystified: On the sources of the discriminative power," in *Proc. 6th Int. Conf. (Co-NEXT)*, 2010, Art. no. 9.

[19] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," in *Proc. Int. Workshop Passive Act. Netw. Meas.*, 2004, pp. 205–214.

[20] A. Pescapè, A. Montieri, G. Aceto, and D. Ciuonzo, "Anonymity services Tor, I2P, JonDonym: Classifying in the dark (Web)," *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2018.2804394.

[21] A. W. Moore and D. Zuev, "Internet traffic classification using Bayesian analysis techniques," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst. (SIGMETRICS)*, 2005, pp. 50–60.

[22] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Proc. 6th Int. Conf. Passive Act. Netw. Meas.*, 2005, pp. 41–54.

[23] J. Patel, A. X. Liu, and E. Torng, "Bypassing space explosion in high-speed regular expression matching," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 1701–1714, Dec. 2014.

[24] V. F. Taylor, R. Spolaor, M. Conti, and I. Martinovic, "Robust smartphone app identification via encrypted network traffic analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 1, pp. 63–78, Jan. 2018.

[25] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci, "Towards self adaptive network traffic classification," *Comput. Commun.*, vol. 56, no. 2, pp. 35–46, Feb. 2015.

[26] Y. Wang *et al.*, "A semantics aware approach to automated reverse engineering unknown protocols," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct./Nov. 2012, pp. 1–10.

[27] C. Xu, S. Chen, J. Su, S. M. Yiu, and L. C. K. Hui, "A survey on regular expression matching for deep packet inspection: Applications, algorithms, and hardware platforms," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2991–3029, 4th Quart., 2016.

[28] X. Yu, W. Feng, D. Yao, and M. Becchi, "O^{3}FA: A scalable finite automata-based pattern-matching engine for out-of-order deep packet inspection," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Mar. 2016, pp. 1–11.

[29] Z. Yuan, J. Xu, Y. Xue, and M. van der Schaar, "Bits learning: User-adjustable privacy versus accuracy in Internet traffic classification," *IEEE Commun. Lett.*, vol. 20, no. 4, pp. 704–707, Apr. 2016.

[30] Z. Yuan, Y. Xue, and M. van der Schaar, "BitMiner: Bits mining in Internet traffic classification," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 93–94.

[31] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A semantics-aware approach to the automated network protocol identification," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 583–595, Feb. 2016.

[32] J. Zhang, X. Chen, Y. Xiang, W. Zhou, and J. Wu, "Robust network traffic classification," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 1257–1270, Aug. 2015.

[33] J. Zhang, Y. Xiang, Y. Wang, W. Zhou, Y. Xiang, and Y. Guan, "Network traffic classification using correlation information," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 1, pp. 104–117, Jan. 2013.

Neminath Hubballi received the Ph.D. degree from the Department of Computer Science and Engineering, IIT Guwahati, India. He is currently an Assistant Professor with the Discipline of Computer Science, IIT Indore, India. Prior to the current role, he was with corporate research and development centers of Samsung, Infosys Lab, and Hewlett-Packard. He has several publications in the areas of security. His areas of interests include networks and system security. He has served as a TPC member and the chair of conferences. He is a regular reviewer of many security journals and conferences.

Mayank Swarnkar received the M.Tech. degree in Information Technology from the Indian Institute of Information Technology Allahabad, India, in 2013, with a focus on wireless communications and computing. He is currently pursuing the Ph.D. degree with the Discipline of Computer Science and Engineering, IIT Indore, India. His research interests are in networks and system security.