Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Three State Evaluation of Flow based Intrusion Detection

Aditya Godase

adityagodase84@gmail.com

COEP Technological University, Pune

Sarthak Godge

COEP Technological University, Pune

Prathamesh Bhawari

COEP Technological University, Pune

Vaidahi Gotmare

COEP Technological University, Pune

Research Article

Keywords: intrusion detection · network security · CNNLSTM · class imbalance · data leakage · CIC-IDS2017 · CSECIC-IDS2018

Posted Date: November 25th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11843470/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

THREE-STATE EVALUATION OF FLOW-BASED INTRUSION DETECTION

Sarthak R. Godge
Department of Computer Science and Engineering
COEP Technological University
Pune 411005, India

Prathamesh A. Bhawari
Department of Computer Science and Engineering
COEP Technological University
Pune 411005, India

Aditya V. Godase
Department of Computer Science and Engineering
COEP Technological University
Pune 411005, India

Vaidahi A. Gotmare
Department of Computer Science and Engineering
COEP Technological University
Pune 411005, India

V. M. Khadse
Project Guide
Department of Computer Science and Engineering
COEP Technological University
Pune 411005, India.

ABSTRACT

Flow-based intrusion detection systems classify whether an attack is happening. We introduce a third class corresponding to pre-intrusion reconnaissance activities based upon attack subtypes that have been observed in the capture. This means that the classification is based upon the contents of the flow itself. We investigate if sequential modelling methods can detect this new class. We achieve a macro-F1 score of 0.8605±0.0019 using a hybrid convolutional recurrent neural network trained upon 20-flow windows with a partition of contiguous blocks of records. The best neural model achieves a score of 0.8668±0.0018. Gradient boosted trees trained upon 6 summary statistics from the same window give a score of 0.8758±0.0002. This means the simpler model has a repeatable advantage over the complex model. It is also 4-30 times more stable to seed changes. However, using 20-flow windows provides an improvement of 3.1 macro-F1 points compared to single record classification. Using class imbalance mitigation techniques gives up to 0.5 macro-F1 improvement compared to no mitigation. However, none improve the threshold free average precision score for the reconnaissance class. Retraining using the traditional shuffled partition improves the macro-F1 score by 3.2 and the average precision for the reconnaissance class by 12.2. This indicates why often perfect scores are achieved on these datasets. In addition, the recall scores for the reconnaissance class break down to show either very high values (≥ 0.997) for port scanning or moderate values (between 0.32 and 0.70) for infiltration. This means the overall detection rate for the early stages of an attack is not informative.

Keywords intrusion detection · network security · CNNLSTM · class imbalance · data leakage · CIC-IDS2017 · CSECIC-IDS2018

1 Introduction

When detecting a denial of service payload flooding a link by firing one detection we have successfully detected but too late. Any intrusion will be a campaign with discovery required to find hosts that are accessible, map services

and compromise. While these actions are individually inconspicuous they are typically considered benign. The only opportunity to cheaply detect during this is during this stage.

To detect whether flows are normal, probing or attacking we must classify short windows of data. While it is trivial to include a new state, making that state something that can be learned is difficult. Previous work often includes a pre-attack state by simply labelling any flows prior to labelled attacks. The label here is not related to the actual data we have but what comes after. This means that the classifier can only succeed by rote learning of when attacks occur in the dataset. Our work ties this state to sub-types of behaviour that are naturally probing.

A second issue is how data is split. As flows are generated very quickly in a single attack burst, any randomised splitting of the data will cause some flows from the same burst to be in each set. Part of our test score will therefore be measuring recall. This is commonly acknowledged but rarely quantified. We split the data into blocks of consecutive records while still keeping the randomised protocol.

Our contributions are: (i) a three state problem where the middle state is based on observed sub-types of probing. (ii) A block based split and measurement of the inflation caused by the randomised approach using the same data. (iii) All design decisions have controls to determine whether the window is useful or the ordering is useful. Only the window was found to be useful. (iv) The probing state is broken down into sub-types showing that the overall state is almost entirely driven by port scanning.

2 Related Work

The highest performing results in the CIC-IDS2017 utilise traditional classifiers to classify the flow level features extracted from CICFlowMeter [1]. These classifiers include Support Vector Machines, Nearest Neighbours, Random Forests and Gradient Boosted Ensemble. These methods achieve above 99% binary accuracy. Another set of methods transforms the output vector into a matrix for use with Convolutional Networks. Both are single instance methods meaning that if an attack spanned multiple consecutive flows it would not be captured in the input. Convolutional Recurrent Networks have been used to solve this issue by using a convolution to capture local interactions between features, and a recurrent network that captures how the representation changes with each instance [5]. Improvements of up to 8 F1 points have been observed when minority classes were classified compared to single paradigm methods.

Fewer papers have focused on auditing the data itself. Various attempts have been made to re-label the data revealing duplicate instances, incorrect labels and issues during feature extraction. These labelled versions of the data have been released [3, 4]. This is relevant to the claims of near perfect accuracy as a large amount of performance was due to duplicates in the original releases. We have also defined the intermediate class based on its content, measured the leakage, and tested the sequential hypothesis using a control group that was given the same window but in an aggregated form.

3 Method

3.1 Assigning Labels to States

We have a sequence of flow records $f_1, f_2...$ where each record is a vector containing a number of statistical features. We define a window $W_i = (f_i...f_{i+T-1})$ where we assign a state $y_i \in \{0, 1, 2\}$. If the records contain benign labels then y = 0. If the labels are INFILTRATION or PORTSCAN, then y = 1. These labels denote enumeration or foothold attacks that can be identified from the records. Finally, if the label is any other attack type such as denial of service, brute-force authentication, web attack, botnet command and control, or Heartbleed then y = 2. We take the label from the last record, $y_i = \ell(f_{i+T-1})$. Using the worst label present in the window will label a window beginning with an attack record followed by 19 benign records as an attack. This means we are rewarding our detector for detecting an attack in a steady stream.

3.2 Data and Normalisation

The CIC-IDS2017 dataset contains five days of data and is the only dataset with a volume of PORTSCAN records. It also contains few INFILTRATION records. The CSE-CIC-IDS2018 dataset [2] contains ten days' worth of data and has more testbed resources, so it has a volume of INFILTRATION records but no PORTSCAN category.Both datasets contain labels assigned to bidirectional flows by CICFlowMeter. This produces around 80 features including session length, moments of packet sizes in each direction, time between packets, flags and ratios of throughputs. An alias table is used to convert the 130 features present in the two different datasets into a canonical form. This results in 78 features and any record from either year will be represented the same way.

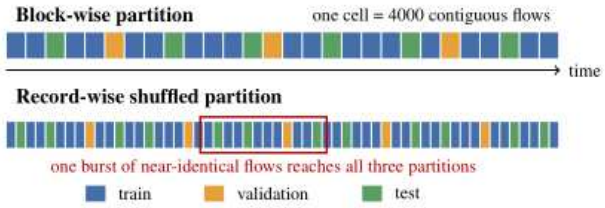

Figure 1: The two partitioning protocols. The shuffled one is retained as a controlled condition so that its effect can be measured.

3.3 Preprocessing and Blocking

Throughput features are calculated by dividing by the session length. If the session ended quickly then the value becomes infinite. We map theses values to the missing indicator before any numerical processing. We sort the data into chronological order using a stable sort. Any exact duplicates (all features + label the same) are removed within each file. Some of the 2018 days had over 40% of rows duplicated. Duplicating is the easiest way to improve scores through shuffling as the same records will exist on both sides.

The deduplicated stream is split into blocks of size B = 4000. Each block takes the worst label within the block. The blocks are split into training, validation and testing sets using a stratified split of 70/10/20. This means that the blocks are grouped together based on their label. As we are grouping the data into blocks any burst of similar records will remain together (Fig. 1).

Three feature selection methods are applied to the data. These methods are all trained on the training data. Variance thresholds are applied to remove any nearly constant columns. Correlation filtering is applied to remove one feature from any pair with a correlation above 0.98. Mutual information is applied to rank the remaining features based on their relationship with the label. Only the top k = 35 are kept. Many of the features have a large range of values and are skewed to the right with valid extreme values. We therefore apply a transformation $\phi(x) = \text{sign}(x)\log(1 + |x|)$ We serialise the fitted transformation so that we can apply the same preprocessing during inference as during training. Finally we extract windows of size T = 20 with a stride of 2. This means that we do not have any windows overlapping block boundaries and there are no test records in training windows.

3.4 Architecture

Model A uses a convolutional layer followed by a recurrent aggregator. Two 1-D convolutions of width 3 are used. The first extracts 64 patterns from the time series of the 35 descriptors which span 3 consecutive data points. The second combines these into 128 patterns. Padding is used to preserve the number of steps, and the convolutions do not have a bias since the normalisation allows for a learnable offset. Then a pooling layer of width 2 is applied to halve the size, then two recurrent layers. The second of these reduces the size to 1 window of size 64. A gating mechanism means the signal from early in the window is passed through to the decision, with any noise in between ignored.

Model B acts as a baseline model. Three convolutions are stacked together with widths 3,5 and 7. Their outputs are combined. A bi-directional gated recurrent unit is used with 96 units instead of the stacked recurrent layers. An attention mechanism is used in the window summary, by taking the element-wise mean weighted by attention for each step, and concatenating this with the element-wise max. This weighting means that each position within the window is weighted rather than the final position being favoured. No pooling has been applied so the recurrent layers see all 20 steps in the input.

3.5 Imbalanced classes

In both releases there is a large imbalance, with benign traffic being around an order of magnitude more than the other classes. All models were trained using the same parameters. The control model was trained with no modifications, just cross entropy. Weights used square root balanced class weights with majority under-sampling. Focal loss [7] was also used [7]. Sampler used stratified sampling to create mini-batches with roughly equal numbers of all three states. Smote [6] was also used, with interpolation of the minority windows. However, the partition was fixed, and this was only done during training blocks.

4 Experimental Setup

This experiment has a control for every decision. Model D is a boosted ensemble [8] trained at the record level. This provides an upper-bound performance when no temporal knowledge is available. Model C uses the same learning algorithm but is trained using summary statistics from the window. This means the information seen by Model C is the same as Model A but summed rather than sequenced. Any difference in performance will be due to the sequence structure. Model E is a blend of the two class distributions of Model A and Model C, while Model E' is a blend of Model B and Model C. Each blend weight is determined based on the validation performance.

The early stopping, class probability scaling and blending weights are all determined on the validation data. The test data is then scored once. Macro-F1 score is used to rank the models. This gives each state equal weight, meaning that the benign traffic cannot affect the score. This is complimented with threshold independent average precision for the intermediate state. Accuracy is given for comparison purposes. There are two controlled experiments. Every model is retrained and evaluated using the window split with shuffled partitions. The primary models are trained using one capture year and tested using the other. Every plot below shows the mean result over three seeds. This means the standard deviation shows the variance in training.

The configuration used is given in Table I. This is the maximum training time available. The dataset was split into 5,544,378 training windows, 790,172 validation windows and 1,588,153 test windows. Metrics were calculated using a maximum of 106 test windows. Training was carried out on one NVIDIA T4 accelerator.

Table 1: Experimental Configuration

Parameter	**Configuration**
Window length T / stride	20 / 2
Descriptors k	35 of 78 harmonised
Block size B	4000 records
Partition ratio	70 / 10 / 20 of blocks
Optimiser [9]	Adam, $\eta = 10^{-3}$
Batch size	1024 windows
Loss	Cross-entropy; focal ($\gamma = 2$) when balanced
Maximum epochs	50
Early-stopping patience	8
Training windows per neural model	3×10^6
Training windows for tree models	2×10^6
Evaluation windows (cap)	10^6
Seeds	3

5 Results

All numbers are averages +/- standard deviations across 3 different seeds on the held out test blocks. For all models, the operating point with the highest macro-F1 score was found. For ensembles, the class weights were tuned but had no effect on the predictions. Tuning the class weights in Model C increased the macro-F1 to 0.8758 from 0.8734; however, infiltration recall was decreased (see Table III)

5.1 The window helps, but order does not

Answering the title question, using a window of size 20 increases the macro-F1 score to 0.8758 from 0.8449 for single record classification (an increase of 3.1). There is no variation across seeds (maximum difference of 0.0005), so using the context improves results. For Model A, using the same window but in an ordered form gives a macro-F1 score of only 0.8605 +/- 0.0019. This is still lower than the best performing neural network (attention variant B) which achieves 0.8668 +/- 0.0018. This means there is a difference of 0.0090 between the aggregated model and best performing neural network. This difference is approximately 8 times larger than the combined standard error and occurs for all imbalance methods, attention variant and for Model C at the untuned operating point (0.8734).

Ensemble models provide some improvement. Using a blend of models A and C (Model E) provides an improvement of 0.0011 in the macro-F1 score (approximately 2 standard errors). Using a blend of models B and C (Model E') provides a larger increase of 0.0026 with no variation across seeds (0.0001). The average precision of reconnaissance is also improved to 0.768 from 0.760. The neural network representation contains some information not captured by the summary statistics, however not enough to make a competitive model.

Table 2: Test Performance on the Pooled Releases Under Block-Wise Partitioning, Mean ± S.D. Over Three Seeds

Model	Accuracy	Macro-F1	Pre-Att. P	Pre-Att. R	Pre-Att. F1	Pre-Att. AP	Attack F1
D: trees, single record	0.9670 ± 0.0002	0.8449 ± 0.0005	0.539 ± 0.004	0.587 ± 0.001	0.562 ± 0.002	—	0.9926 ± 0.0001
C: trees, window statistics†	0.9770 ± 0.0000	**0.8758**	0.687 ± 0.001	0.602 ± 0.002	0.642 ± 0.001	0.7604 ± 0.0002	0.9993 ± 0.0000
A: CNN–LSTM, no balancing	0.9779 ± 0.0004	0.8605 ± 0.0019	0.804 ± 0.038	0.474 ± 0.020	0.596 ± 0.006	0.7266 ± 0.0077	0.9989 ± 0.0002
A: CNN–LSTM, balanced sampler†	0.9716 ± 0.003	0.8658 ± 0.0008	0.576 ± 0.006	0.662 ± 0.006	0.616 ± 0.003	0.7222 ± 0.0031	0.9985 ± 0.0002
A: CNN–LSTM, SMOTE	0.9749 ± 0.0001	0.8659 ± 0.0012	0.675 ± 0.007	0.568 ± 0.010	0.617 ± 0.004	0.7122 ± 0.0060	0.9954 ± 0.0009
A: CNN–LSTM, class weights†	0.9755 ± 0.0022	0.8639 ± 0.0052	0.709 ± 0.103	0.548 ± 0.085	0.609 ± 0.017	0.7222 ± 0.0175	0.9968 ± 0.0027
B: multi-scale CNN–BiGRU, attention	0.9721 ± 0.0013	0.8668 ± 0.0018	0.584 ± 0.023	0.658 ± 0.021	0.618 ± 0.005	0.7243 ± 0.0042	0.9987 ± 0.0001
E: ensemble of A and C	0.9731 ± 0.0002	0.8769 ± 0.0009	0.586 ± 0.003	0.724 ± 0.003	0.648 ± 0.003	0.7665 ± 0.0039	0.9994 ± 0.0000
E': ensemble of B and C	0.9747 ± 0.0005	**0.8784**	0.617 ± 0.012	0.689 ± 0.015	0.651 ± 0.000	0.7681 ± 0.0006	0.9994 ± 0.0000

† Validation-tuned class scales; all other rows use the argmax decision, which gave the higher macro-F1 for them.

Variation in results shows that Model C has much less variation (between +/- 0.0002 across seeds) than the neural networks (between +/- 0.0008 and +/- 0.0052). This means that the simpler model is not only more accurate but also 4-30 times more repeatable. This does not indicate that recurrence can not help with flow data but that, for these datasets and sequence length, the summary statistics contain almost all the information extracted by the recurrent layer. Work has not shown otherwise.

5.2 Class imbalance mitigation

Table 3: Recall on the Intermediate State, by Treatment and Sub-Type

		INFILTRATION Recall	
Model	PORTSCAN R	Untuned	Tuned
C: trees, window stats	0.9997 ± 0.0002	0.647 ± 0.005	0.489 ± 0.002
A: no balancing	0.9990 ± 0.0002	0.324 ± 0.025	0.324 ± 0.025
A: class weights	0.9985 ± 0.0011	0.664 ± 0.057	0.420 ± 0.109
A: balanced sampler	0.9992 ± 0.0001	0.697 ± 0.025	0.565 ± 0.007
A: SMOTE	0.9973 ± 0.0005	0.446 ± 0.013	0.446 ± 0.013
B: attention	0.9994 ± 0.0004	0.560 ± 0.027	0.375 ± 0.095
E: ensemble of A and C	0.9994 ± 0.0003	0.645 ± 0.004	0.645 ± 0.004
E′: ensemble of B and C	0.9997 ± 0.0001	0.600 ± 0.019	0.600 ± 0.019

Table 4: The Same Models and Data Under Two Partitioning Protocols

Model	Metric	Block-wise	Shuffled	Inflation
C: trees	Macro-F1	0.8734 ± 0.0004	0.9055 ± 0.0001	$+0.0321$
C: trees	Pre-Att. AP	0.7604 ± 0.0002	0.8828 ± 0.0004	$+0.1224$
A: weights	Macro-F1	0.8542 ± 0.0029	0.8768 ± 0.0048	$+0.0226$
A: weights	Pre-Att. AP	0.7222 ± 0.0175	0.8044 ± 0.0321	$+0.0822$

Argmax decision in both columns; A is the CNN–LSTM with class weights.

Table III shows the most significant findings. In the first column, all models achieved high port scan recall values ($\geq$ 0.997). However, the infiltration recall varied greatly (between 0.32-0.70). This means that the Pre-Attack F1 score was essentially measuring the average between near perfect detection of one noisy subclass and partial detection of the other. Since port scanning attacks are detected by traditional defences and infiltration attacks are those performed by capable adversaries, this shows that the models are performing worse at detecting infiltration attacks. Any work which provides a single value for detection of different recon classes should be broken down.

As can be seen, combining Table III and Table II, class imbalance can be solved. The models trained using the methods achieve high macro-F1 scores (between 0.8639-0.8659 vs 0.8605) compared to the original data. The balanced sample and SMOTE methods improve performance by 0.005, which is outside of the variance caused by different seeds. Class weights however are within this range. However, there is no improvement in threshold-free average precision (between

0.7122-0.7222 vs 0.7266). This shows that the methods are not improving the model, simply changing the operating point. The original model has a conservative operating point (Pre-Attack Precision 0.804, Recall 0.474) and only detects 0.324±0.025 of infiltration attacks. However, using the balanced sample increases infiltration recall to 0.697±0.025 with a Pre-Attack Precision of 0.464. Since this detector would be used when the analyst handles alerts, this model is preferable despite having a lower overall score. Since the class weights are tuned during validation, all models, including Model C, are skewed towards precision. This increases accuracy but reduces recall.

5.3 Traditional protocol improvements

Table IV compares the performance of the same architectures on the same data using both methods. It shows that shuffling improves performance for the tree model (3.2 increase in macro-F1) and neural model (2.3 increase) (4.4 increase and 2.8 increase with validation tuning). There is a large increase in threshold-free average precision for the rare state (12.2 increase and 8.2 increase). Average precision is a more useful metric as it does not depend on where the decision boundary is. A 12 point improvement isn't due to rounding differences. The neural model also suffers from much less stability (±0.032).

Since only the assignment has changed, this is entirely caused by having nearly identical data from one burst on both sides. This gives an explanation as to why the large values are often reported on the CIC datasets. This also shows that the performance between different protocols isn't comparable.

5.4 Cross year transferability

Table V shows poor performance from all models. All models had poor performance (Macro-F1 between 0.33-0.47) and very low average precision for reconnaissance (≥ 0.13). The single record model performs poorly when trained on 2017, achieving an Attack F1 of 0.048 for 2018. The neural model trained on 2018 labels nearly all attack windows in 2017 as reconnaissance. This gives it a very poor Attack F1 (0.018). It also has a high infiltration recall (0.96) which is due to over-prediction. The window statistics model performs best in both directions but still performs very poorly.

6 Threats to Validity and Conclusioin

Table 5: Cross-Release Transfer, Mean ± S.D. Over Three Seeds

Dir.	Model	Macro-F1	Pre AP	Atk F1
2017 3*→2018	D: record	0.3432 ± 0.0095	—	0.048 ± 0.028
	C: window	0.4545 ± 0.0081	0.042 ± 0.001	0.368 ± 0.022
	A: CNN–LSTM	0.4581 ± 0.0234	0.053 ± 0.009	0.379 ± 0.066
2018 3*→2017	D: record	0.4217 ± 0.0049	—	0.361 ± 0.012
	C: window	0.4659 ± 0.0366	0.082 ± 0.043	0.417 ± 0.056
	A: CNN–LSTM†	0.3400 ± 0.0487	0.133 ± 0.065	0.018 ± 0.020

† Validation-tuned class scales.

There is no buffer, meaning there are no gaps between the training and test sets, so there will be overlapping local context between a training set and the next set. We only remove exact duplicates, so any records with minor differences between them still remain. This means that the values given in Table IV are a lower bound. Both releases contain known labelling and feature extraction issues [3] [4], so this will affect the results of any analysis on the original data. This should be taken into consideration when looking at the infiltration values. The data has been sorted globally, so there could be unrelated conversations within a window. Whilst this mimics how the detection mechanism would receive the data it also reduces the amount that can be learnt about by the recurrent layer. The transfer between the two capture years did not succeed in either direction (see table V).

A reconnaissance state was added to the network based on the attack subtypes observed. The network was tested with no leakage due to shuffling the data at record level. The detector performs well in a strict sense with a high recall for port scanning attacks (0.997+). The infiltration recall is low however (between 0.32 and 0.70). The twenty flow window adds value to the detection, with a gain in macro-F1 of 3.1 points. Overall the results show no benefit to considering the sequence of the window, only the summary. When combined with the summary model there is a small gain (0.26 macro-

F1 points) but this is consistent. The simple model has much less variance (4-30x less) depending on the seed. Applying imbalance solutions gives gains in the macro-F1 of up to 0.5 points but does not increase average precision. This will affect the infiltration recall (between 0.32 and 0.70), so should be decided upon using operational considerations. Using the standard shuffled protocol causes an increase in average precision for reconnaissance of 12 points, this could be responsible for many of the almost perfect results seen in this literature. Future work would include repeating the study on the corrected releases, with a buffer between partitions. The decomposition by attack subtype suggests that the real issue is not finding better architectures for reconnaissance in general but infiltration specifically. None of the models trained performed well at this task.

References

[1] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," in *Proceedings of the International Conference on Information Systems Security and Privacy (ICISSP)*, 2018.

[2] Canadian Institute for Cybersecurity, "CSE-CIC-IDS2018 on AWS." [Online]. Available: https://www.unb.ca/cic/datasets/

[3] G. Engelen et al., "Troubleshooting an intrusion detection dataset: the CICIDS2017 case study," in *IEEE Security and Privacy Workshops (SPW)*, 2021.

[4] L. Liu et al., "Error prevalence in NIDS datasets: CIC-IDS-2017 and CSE-CIC-IDS-2018," in *IEEE Conference on Communications and Network Security (CNS)*, 2022.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, 1997.

[6] N. V. Chawla et al., "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, 2002.

[7] T.-Y. Lin et al., "Focal loss for dense object detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[8] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[9] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.