Generating Balanced and Realistic BGP Traffic for Machine Learning-Based Anomaly Detection

Shadi Motaali, Jorge E. López de Vergara, Luis de Pedro and Iván González
Escuela Politécnica Superior, Universidad Autónoma de Madrid, Spain
{shadi.motaali, jorge.lopez_vergara, luis.depedro, ivan.gonzalez}@uam.es

***Abstract*—Machine learning (ML)–based BGP anomaly detection faces key challenges, including severe dataset imbalance and lack of ground-truth labeling. Public repositories such as RIPE RIS and RouteViews provide raw BGP data, where anomalies represent less than 5% of traffic—insufficient for training robust ML models. This paper presents a methodology to generate balanced and realistic BGP traffic using Scapy-based packet construction with full RFC compliance and process-ID–based labeling. The framework produces protocol-compliant packets, simulates both normal and anomalous scenarios, and exports datasets in PCAP and ML-ready CSV formats with controllable class distribution. Validation through GNS3 simulations, Cisco traces, and PCAP-to-Scapy reconstruction confirms byte-level correctness. A proof-of-concept dataset containing 45 928 labeled `UPDATE` messages achieved 96.88% classification accuracy using Random Forest, demonstrating that the generated traffic captures realistic behavioral patterns suitable for ML-based anomaly detection and model evaluation.**

***Index Terms*—BGP, Synthetic Traffic, Anomaly Detection, Scapy, Network Security, Dataset Generation, Traffic Generation**

I. Introduction

Border Gateway Protocol (BGP) serves as the fundamental routing protocol of the Internet, enabling autonomous systems (AS) to exchange routing information and maintain global connectivity. However, BGP, in its original design, lacks intrinsic route validation or cryptographic protection mechanisms, relying instead on trust between peers. Although later extensions such as TCP-AO [1], RPKI [2], and BGPsec [3] enhance security, their deployment is still far from complete. This partial adoption leaves the global routing system vulnerable to attacks—including prefix hijacking and route leaks—that can lead to traffic interception, Denial-of-Service events, and large-scale outages.

To solve this problem, machine learning (ML) offers promise for detecting BGP anomalies by learning patterns from historical traffic data. However, ML effectiveness critically depends on training data quality and balance [4]. Existing public datasets suffer from severe class imbalance (anomalies $< 5\%$ of traffic), missing ground-truth labels, and timestamp-based labeling ambiguity when correlating traffic with documented incidents.

This work is partially funded by a grant from the Dept. of Electronics and Communication Technologies at Universidad Autónoma de Madrid, as well as by the R&D activity program with reference TEC-2024/COM-504 and acronym RAMONES-CM, granted by the Comunidad de Madrid through the Directorate General for Research and Technological Innovation via Order 5696/2024.

Generating synthetic BGP traffic that addresses these dataset limitations while maintaining realism presents several technical challenges:

- **Protocol Complexity**: BGP involves complex state machines, session establishment procedures, and diverse message types (OPEN, UPDATE, KEEPALIVE, NOTIFICATION) that must follow strict RFC specifications.
- **Attribute Handling**: BGP UPDATE messages contain multiple path attributes (AS_PATH, NEXT_HOP, LOCAL_PREF, COMMUNITIES) that must be correctly formatted and consistent.
- **Dual-Stack Support**: Modern deployments require proper handling of both IPv4 and IPv6 through Multiprotocol BGP (MP-BGP).
- **Attack Realism**: Synthetic anomalies must accurately represent real attack scenarios while maintaining protocol validity.
- **Scalability**: Network simulation platforms (e.g., GNS3 [5]) require substantial resources.

This paper addresses these challenges by presenting a systematic methodology for generating balanced and realistic BGP traffic through Scapy packet manipulation and comprehensive RFC compliance, including the fix of a Scapy BGP module issue that produced non-compliant KEEPALIVE messages. The remainder of this paper is organized as follows: Section II reviews related work on BGP datasets and traffic generation, Section III presents the proposed generation methodology and describes the implementation details, Section IV mentions validation process, and preliminary results. Finally, Section V concludes the paper and outlines directions for future work. The complete implementation is available at our GitHub repository.[1]

II. Related Work

Public BGP datasets from RIPE RIS [6], RouteViews [7], and CAIDA [8] operate global networks of route collectors gathering routing tables and updates from multiple vantage points. While comprehensive, these datasets pose challenges for ML training: raw messages lack ground-truth labels, privacy concerns limit analysis, and manual timestamp-based

[1]https://github.com/shadimotaali/BGP_Traffic_Generation_by_Scapy

labeling is unreliable. Fonseca et al. [9] attempted a feature engineering approach for detection, but remained fundamentally limited due to dataset quality.

Traffic generation has evolved from replay-based tools [10] that cannot create novel scenarios to sophisticated AI-based methods. Recent generative models include Generative Adversarial Networks (GAN) [11] and Graph Neural Networks (GNN) [12] that model topology and patterns but require substantial real traffic for training and cannot guarantee RFC compliance. Recent works [13], [14] explored using LLMs to generate Scapy code for multi-protocol conversations (ICMP, ARP, DNS, HTTP), either through prompt engineering or by training a model specifically for this task. While promising as an intermediate code generation layer, these methods have limited BGP protocol coverage and do not address dataset balance. BGP-specific tools like BOPIS [15] combine BGP, OpenFlow, and Scapy for SDN monitoring but emphasize real-time analysis rather than systematic dataset generation.

Beyond dataset generation, several works address detection and root cause analysis—downstream applications requiring quality training data. For instance, [16] uses routing dependence analysis with wavelet-based detection and PCA for root cause localization, validated on three real-world events. The work in [17] introduced BEAR, an LLM framework for explaining detected anomalies. BEAR addresses data scarcity by generating event specifications via LLM, retrieving real BGP data from public collectors, and injecting synthetic anomalies. While achieving realism through data modification, it remains constrained by available patterns and cannot easily control class distribution.

In contrast, our methodology generates traffic from scratch without requiring real data, providing complete control over balance and labels. Table I (Appendix) compares all approaches.

III. Methodology

A. Overview

Our methodology for generating balanced and realistic BGP traffic consists of five key components:

1) **RFC Compliance Framework**: Generated traffic is constructed directly from RFC-defined BGP message formats, replicating each field and byte structure specified in the standard.
2) **BGP Session Establishment**: Implementing complete TCP handshake and BGP OPEN/KEEPALIVE exchange.
3) **Normal Traffic Generation**: Creating diverse legitimate BGP UPDATE scenarios.
4) **Anomalous Traffic Generation**: Systematically generating various attack patterns.
5) **Dataset Export and Labeling**: Converting packets to machine learning-ready formats with process-ID tracking.

During development, a large language model (Claude Sonnet 3.7 [18]) assisted primarily with initial code scaffolding and Scapy boilerplate generation, such as repetitive attribute handling and basic packet construction templates. Nevertheless, the overall architecture, traffic generation logic, attack scenario design, and validation pipeline were developed manually. All LLM-generated code segments were subsequently reviewed, corrected where necessary, and validated against RFC specifications and Cisco/GNS3 traces. The final implementation (4 075 lines of Python code) runs independently without runtime LLM access. This workflow—using LLMs for accelerating initial development while maintaining rigorous manual validation—may inform other researchers exploring similar tools for protocol-level code generation.

B. BGP Session Establishment

Our implementation follows RFC 4271 [19] compliance throughout the session lifecycle. Sessions are established over TCP port 179 with standard three-way handshakes. All TCP-level requirements are met, including window sizing (RFC 879 [20]), sequence number tracking, and a 60-byte minimum packet size with padding.

Following TCP establishment, peers exchange BGP OPEN messages containing: version 4, AS numbers (2-byte and 4-byte formats per RFC 4893 [21]), 180-second hold time, and IPv4-format BGP identifier. Optional parameters advertise capabilities: Multiprotocol support for IPv4/IPv6 unicast (RFC 4760 [22]) with AFI/SAFI values, route refresh (RFC 2918 [23]), and 4-byte AS support.

After a successful OPEN exchange, peers confirm establishment via KEEPALIVE messages. We discovered Scapy [24] KEEPALIVE implementation violates RFC 4271, requiring 19-octet messages. Multiple inheritance (`class BGPKeepAlive(BGP, BGPHeader)`) combined with layer binding causes duplicate headers and 38-byte packets. We corrected this using `BGPKeepAlive()` directly, producing compliant 19-byte packets validated against GNS3 and Cisco implementations.

C. Traffic Generation Strategy

1) Traffic Generation Parameters and Timing: To balance realism with reproducibility, we employed a hybrid approach combining static topology structure with randomized parameters. **Static components** include AS relationship structure (peering topology), AS tier assignments (Tier 1/2/3 hierarchy), and basic prefix allocation schemes. **Randomized components** include AS numbers within tier-appropriate ranges (Tier 1: 1 000–5 000, Tier 2: 10 000–20 000, Tier 3: 30 000–50 000, Private: 50 000–65 000), IP addressing with random private subnet assignments (Tier 1: 100.64.0.0/10, Tier 2: 172.16.0.0/12, Tier 3: 192.168.0.0/16, Peering: 10.0.0.0/8, Public prefixes: 203.0.0.0/8, 198.51.100.0/24), BGP parameters (randomized router IDs, hold times 60–180s, LOCAL_PREF values, community strings), and session timing with realistic random delays. However, three specific IP addresses and AS numbers are kept static for reproducible attack scenario generation and validation.

For temporal realism, BGP `UPDATE` packets are generated over a configurable interval (e.g., 20 min), where inter-arrival times are sampled from a Pareto distribution to reproduce the bursty, heavy-tailed timing observed in real routing traffic [25], [26]:

$$f(x) = \frac{\alpha x_m^\alpha}{x^{\alpha+1}}, \quad x \geq x_m, \tag{1}$$

where x_m is the minimum delay and α controls tail heaviness; $\alpha = 2.5$ models normal behavior and $\alpha = 1.8$ attack conditions. The resulting traffic shows median and 95th-percentile delays of 0.015 s and 0.040 s (see Appendix II). This configuration yields temporally realistic synthetic traces for downstream feature extraction and anomaly detection.

2) Normal Traffic Generation: To create diverse and realistic normal BGP traffic, we implemented eight distinct legitimate routing scenarios representing common operational events in production networks: (1) ORIGIN attribute changes (IGP/INCOMPLETE transitions), (2) AS_PATH modifications through legitimate path prepending for traffic engineering, (3) NEXT_HOP changes during route updates, (4) LOCAL_PREF modifications for policy-based routing, (5) COMMUNITIES attribute updates for route tagging and filtering, (6) Duplicate announcements caused by prefix repetition, (7) AS_PATH length variations due to topology changes, and (8) ROUTE AGGREGATION/SUMMARIZATION for prefix consolidation.

3) BGP UPDATE Messages with Path Attributes: Our implementation generates complete BGP UPDATE messages, contains mandatory path attributes (ORIGIN, AS_PATH, NEXT_HOP) and optional transitive/non-transitive attributes. Our implementation generates complete UPDATE messages based on attribute patterns observed in public BGP repositories (RIPE RIS, RouteViews), as these datasets represent the realistic attribute combinations studied in anomaly detection research.

For IPv4 announcements, we generate UPDATE messages with: ORIGIN (IGP/EGP/INCOMPLETE), AS_PATH with realistic AS sequences, NEXT_HOP addresses, MULTI_EXIT_DISC (MED), LOCAL_PREF, ATOMIC_AGGREGATE flags, AGGREGATOR information, and COMMUNITIES (standard and extended). For IPv6, we use MP_REACH_NLRI (Multiprotocol Reachable NLRI) with AFI=2 (IPv6), SAFI=1 (Unicast), global and link-local next-hop addresses, and properly encoded IPv6 prefixes.

Route withdrawals require special handling. For IPv4 withdrawals, we found an implementation issue with Scapy layer stacking. Our solution creates the BGP UPDATE instance first without layering it on BGPHeader, sets the withdrawn-routes, path-attr, and NLRI fields directly on that instance, then layers it with BGPHeader and processes normally. For IPv6, we use MP_UNREACH_NLRI to represent route removals. Normal traffic also includes legitimate path attribute updates: LOCAL_PREF changes, AS_PATH length variations, COMMUNITIES modifications, and route re-announcements with different attributes.

4) Anomalous Traffic Generation: Our methodology generates three primary categories of BGP anomalies selected for operational relevance and direct mapping to attributes commonly used in detection research: (1) **Prefix Hijacking** — unauthorized announcement of legitimate prefixes including both exact prefix hijacks and more-specific sub-prefix hijacks that exploit longest-prefix matching; (2) **Path Manipulation** — malicious AS_PATH modifications including path shortening to attract traffic, invalid AS sequences, and prepending anomalies that deviate from normal traffic engineering patterns; and (3) **UPDATE Flooding (DoS)** — excessive UPDATE message generation designed to overwhelm BGP peers and consume processing resources.

To enable training of balanced ML models, we employ process-ID–based labeling, where each packet is tagged during generation with a unique identifier. Attack packets are tracked in a dedicated set, allowing precise labeling as "normal", "prefix_hijacking", "path_manipulation", and "dos_attack" during dataset export. This approach provides perfect ground-truth labels without the ambiguity of timestamp-based labeling used in real-world datasets. The generation process allows controllable class distribution, enabling the creation of perfectly balanced datasets (50-50 normal/abnormal) or custom ratios based on specific research requirements, addressing the severe imbalance problem in public BGP datasets, where anomalous traffic represents less than 5% of total traffic.

Although these attack patterns represent real-world scenarios, we emphasize that the generated attack traffic is intended exclusively for research and machine learning model training, not for deployment or live network testing.

5) BGP NOTIFICATION Messages: NOTIFICATION messages are legitimate protocol-defined messages for error handling and session termination per RFC 4271 [19]. Unlike attack scenarios, these represent proper BGP behavior when errors occur. We generate five types: (1) Message Header Error (Code 1) with Bad Message Length subcode, (2) OPEN Message Error (Code 2) with Bad Peer AS subcode, (3) UPDATE Message Error (Code 3) with Malformed Attribute List subcode, (4) Hold Timer Expired (Code 4), and (5) Finite State Machine Error (Code 5). Each NOTIFICATION message is accompanied by the corresponding TCP ACK responses to maintain an accurate session state throughout the simulation. Moreover, NOTIFICATION messages are labeled as "normal" rather than anomalous traffic, since they represent legitimate protocol behavior under error conditions.

IV. Experimental Evaluation

A. Validation Methodology

We validated our generated traffic through four complementary approaches. **(1) GNS3 Cross-Comparison**: We configured identical BGP sessions in GNS3 with Cisco IOS routers using the same AS numbers and IP addresses, then compared packet structures, field values, and message sequences via Wireshark. **(2) Cisco Trace Comparison**: We analyzed our packets against Cisco BGP traces [27], comparing packet length distributions, header formats, attribute ordering, and TCP characteristics. **(3) RFC Compliance Verification**: We systematically verified RFC 4271 [19] compliance, including message headers (Marker, Length, Type), FSM state transitions, attribute formats, and error handling. **(4) Bidirectional PCAP Validation**: We used a PCAP-to-Scapy conversion tool [28] to reverse-engineer our PCAPs.

Figure 1 (see Appendix) illustrates this bidirectional validation process. Using the PCAP-to-Scapy conversion tool with the Scapy BGP contrib module, we successfully reverse-engineered our generated PCAPs, reconstructing complete BGP session establishment (OPEN, KEEPALIVE) and UPDATE messages with all path attributes, including ORIGIN, AS_PATH, NEXT_HOP, MED, LOCAL_PREF, and COMMUNITIES (Panel B). Wireshark dissection (Panel C) confirms RFC 4271 compliance. The perfect correspondence across all BGP attributes and NLRI prefixes validates that our generated traffic is both RFC-compliant and correctly parseable by programmatic and network analysis tools. This bidirectional validation—generating Scapy code to produce PCAPs, then converting PCAPs back to Scapy code—confirms the structural integrity of our generated traffic.

B. Dataset Export and Characteristics

Our system exports generated traffic in two formats for different analysis purposes. All packets are written to PCAP files using the Scapy `wrpcap()` function, enabling analysis with Wireshark, packet-level inspection, debugging, and integration with existing network analysis workflows.

Additionally, we developed a custom parser to extract BGP-specific features and export them to CSV format, directly usable for ML feature engineering and model training. The CSV export includes 13 features per UPDATE message: message type, timestamp, subtype (announcement/withdrawal), peer IP address and AS number, announced/withdrawn prefix, AS_PATH (as string), ORIGIN type, NEXT_HOP address, MED value, LOCAL_PREF value, COMMUNITIES (as string), AGGREGATOR flag and information, and critically, label ("normal", "prefix_hijacking", "path_manipulation", and "dos_attack") based on process-ID tracking.

The generated dataset demonstrates balanced class distribution with 54.21% normal updates, 12.33% prefix hijacking instances, 12.60% path manipulation instances, and 20.86% DoS attack updates. Complete dataset characteristics are in Table III (Appendix).

Our implementation generates 20 minutes of mixed BGP traffic in approximately 24 minutes on commodity hardware (MacBook Pro, M4-Pro, 24 GB RAM, Scapy 2.6.1) spending about 8 GB in the process. This implementation enables a more convenient generation of time-bounded, realistic datasets without the large memory overhead commonly required by simulation-based approaches [5] which would need many more resources to have a similar topology. Detailed generation performance metrics are reported in Table IV (Appendix).

C. Machine Learning Classification Validation

To validate the utility of the generated dataset for machine learning (ML)-based anomaly detection, we trained a Random Forest classifier [29] (scikit-learn, $n\_estimators = 100$) on the 45 928 BGP `UPDATE` messages. From the exported CSV containing 13 packet-level fields per UPDATE file, 25 temporal and statistical features were computed over sliding time windows following [4]. These features capture `UPDATE` message rates, `AS_PATH` characteristics, prefix announcement and withdrawal patterns, as well as path attribute dynamics.

The dataset was aggregated into 158 time windows and subjected to binary classification (normal vs. anomalous traffic) using an 80–20 train–test split with stratified sampling to preserve class distribution (54.21% normal, 45.79% anomalous windows). The Random Forest classifier achieved an accuracy of 96.88%, with precision, recall, and F1-score of 0.98, 0.95, and 0.96, respectively. The confusion matrix indicated only one false positive among 32 test windows. These results serve as an internal consistency check confirming discriminative patterns in the synthetic data, although not as standalone proof of full realism.

While class balance is not strictly mandatory for anomaly detection, training on severely imbalanced data typically degrades sensitivity to minority classes. Our methodology provides *controllable* class distribution at the packet level, unlike oversampling techniques (e.g., SMOTE [30]) that operate on extracted feature metadata. This enables researchers to generate balanced training sets while reserving imbalanced test sets for reflecting operational conditions.

V. Conclusion and Future Work

The presented framework adheres to RFC-defined formats, supports diverse attack simulations, and provides precise ground-truth labeling with PCAP and ML-ready exports. Validation on 45 928 labeled `UPDATE` messages using a Random Forest classifier achieved 96.88% accuracy, confirming that the generated traffic realistically represents BGP behavior and is suitable for training and evaluating ML-based detection models.

While our proof of concept demonstrates the feasibility of synthetic BGP traffic generation, several limitations guide future work. The current dataset models 11 autonomous systems with independent per-session traffic generation, whereas the global Internet includes over 70 000 ASNs with complex inter-peer dependencies. Future extensions will scale the topology and model route propagation effects, where UPDATE messages at one peer trigger subsequent announcements at downstream peers, better capturing convergence dynamics. We will incorporate Routing Information Base (RIB) records for richer temporal and topological context calibrated against RIPE RIS collector data, and evaluate whether models trained on synthetic data generalize to real BGP traces. Additional learning models (XGBoost, SVM, LSTM, Transformer) will be assessed, comparing our packet-level generation against metadata-level oversampling techniques (e.g., SMOTE). The attack taxonomy will expand to include route leaks [31]–[33], BGP session hijacking, and coordinated multi-AS attacks.

References

[1] J. Touch, A. Mankin, and R. Bonica, "The TCP Authentication Option," Internet Engineering Task Force, RFC 5925, Jun. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5925.txt

[2] H. Schulmann, N. Vogel, and M. Waidner, "Rpki: Not perfect but good enough," 2024. [Online]. Available: https://arxiv.org/abs/2409.14518

[3] M. Lepinski and K. Sriram, "BGPsec Protocol Specification," Internet Engineering Task Force, RFC 8205, Sep. 2017. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc8205

[4] S. Motaali, J. E. López de Vergara, and L. de Pedro, "Hybrid feature selection and explainable machine learning for bgp anomaly detection," in *Proc. 4th International Conference on Computing, IoT and Data Analytics*, Madrid, Spain, 2025.

[5] GNS3 Technologies, "GNS3: Graphical network simulator," https://www.gns3.com/.

[6] RIPE Network Coordination Centre, "RIPE routing information service (RIS)," https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris.

[7] U. of Oregon, "Routeviews project," http://www.routeviews.org/.

[8] CAIDA, "The center for applied internet data analysis (caida)," https://www.caida.org/.

[9] P. Fonseca, E. S. Mota, R. Bennesby, and A. Passito, "BGP dataset generation and feature extraction for anomaly detection," in *2019 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2019.

[10] Yazan Siam, "Tcpreplay," https://github.com/appneta/tcpreplay.

[11] Y. Yin, Z. Lin, M. Jin, G. Fanti, and V. Sekar, "Practical gan-based synthetic ip header trace generation using netshare," *SIGCOMM*, Aug. 2022.

[12] T. J. Anande and M. S. Leeson, "Generative adversarial networks (gans): A survey on network traffic generation," *International Journal of Machine Learning and Computing*, vol. 12, no. 6, Nov. 2022.

[13] J. A. Delgado-Soto, J. E. López de Vergara, I. González, D. Perdices, and L. de Pedro, "GPT on the wire: towards realistic network traffic conversations generated with large language models," *Computer Networks*, vol. 265, no. 111308, Jun. 2025.

[14] I. González, J. E. López de Vergara, J. A. Delgado-Soto, D. Perdices, and L. de Pedro, "Training LLMs to speak network," in *Proceedings of the 4th International Conference on Computing, IoT and Data Analytics, ICCIDA 2025*, Madrid, Spain, Jul. 2025.

[15] A. SECK, S. E.ZABOLO, C. S.E.BASSENE, and N. C. SENE, "Bopis-software: A python based software application for sdn east-west inter autonomous system communication and programmable network monitoring tool," *WINCOM*, 2023.

[16] S. Zhao, X. Huang, and P. Zhang, "Locating the root cause of large-scale BGP anomaly with routing dependence," in *2024 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE, 2024.

[17] H. Li, M. Fedeli, V. Kolar, and D. Klabjan, "BEAR: BGP event analysis and reporting," *arXiv preprint arXiv:2506.04514*, June 2025. [Online]. Available: https://arxiv.org/abs/2506.04514

[18] Anthropic, "Claude AI," https://claude.ai.

[19] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," Internet Engineering Task Force, RFC 4271, Jan. 2006. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4271.txt

[20] J. Postel, "The TCP Maximum Segment Size and Related Topics," Internet Engineering Task Force, RFC 879, Nov. 1983. [Online]. Available: https://www.rfc-editor.org/rfc/rfc879.txt

[21] Q. Vohra and E. Chen, "BGP Support for Four-octet AS Number Space," Internet Engineering Task Force, RFC 4893, May 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4893.txt

[22] T. Bates, R. Chandra, D. Katz, and Y. Rekhter, "Multiprotocol Extensions for BGP-4," Internet Engineering Task Force, RFC 4760, Jan. 2007. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4760.txt

[23] E. Chen, "Route Refresh Capability for BGP-4," Internet Engineering Task Force, RFC 2918, Sep. 2000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2918.txt

[24] Scapy Project, "BGP Contrib Module – Scapy Documentation," https://scapy.readthedocs.io/en/stable/api/scapy.contrib.bgp.html.

[25] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic (extended version)," *IEEE/ACM Transactions on Networking*, 1994.

[26] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Transactions on Networking*, 1995.

[27] Cisco Meraki, *BGP - Messages Wireshark*, Cisco Systems, Inc., online documentation; accessed 26-October-2024. [Online]. Available: https://documentation.meraki.com/MX/Other_Topics/BGP_-_Messages_Wireshark

[28] Scapy Project, "PCAP to SCAPY," https://github.com/toonst/pcap2scapy.

[29] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[30] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[31] K. Sriram, D. Montgomery, D. McPherson, E. Osterweil, and B. Dickson, "Problem Definition and Classification of BGP Route Leaks," Internet Engineering Task Force, RFC 7908, Jun. 2016. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7908

[32] G. Huston, "Leaking routes," 2012.

[33] M. S. Siddiqui, D. Montero, R. Serral-Gracià, X. Masip-Bruin, and M. Yannuzzi, "Route leak identification: A step toward making inter-domain routing more reliable," in *Proc. IEEE International Conference on Communications (ICC)*, 2014, pp. 3436–3441.

Appendix

TABLE I
COMPARISON OF BGP TRAFFIC GENERATION AND ANALYSIS APPROACHES

Approach	**Needs Real Traffic**	**RFC Comp.**	**Balanced**	**Labels**	**BGP Focus**	**Purpose**
Real Data [6]–[8]	Yes	Yes	No	No	Yes	Collection
Replay [10]	Yes	Yes	No	No	Yes	Replay
GAN [11]	Yes	N/A	Yes	Part.	No	Generation
GNN [12]	Yes	N/A	Yes	Part.	No	Generation
LLM [14]	No	Yes	N/A	N/A	No	Generation
BOPIS [15]	Yes	Yes	N/A	N/A	Yes	Monitoring
Localization [16]	Yes	Yes	No	Part.	Yes	Detection
BEAR [17]	Yes	N/A	N/A	Part.	Yes	Explanation
Ours	**No**	**Yes**	**Yes**	**Yes**	**Yes**	**Generation**

TABLE II
INTER-PACKET DELAY DISTRIBUTION COMPARISON

Metric	**Pareto**	**Weibull**	**Gaussian**
Mean delay (s)	0.018953	0.015780	0.017199
Median delay (s)	0.014663	0.011948	0.014740
95th percentile (s)	0.040281	0.041079	0.022789
Max delay (s)	0.087794	0.071551	0.109695
Min delay (s)	0.006249	0.002290	0.004408

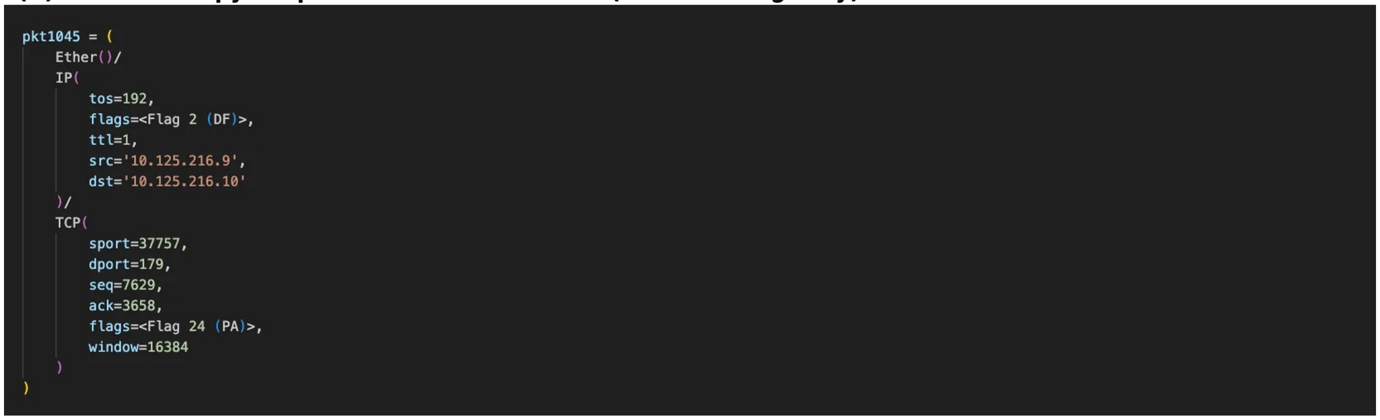
TABLE III
PROOF-OF-CONCEPT DATASET CHARACTERISTICS

Characteristic	**Value**
Update packets	45 928
AS number	11
Announcements	44 975
Withdrawals	953
Normal update	24 898
Prefix hijacking	5 665
Path manipulation	5 785
DoS attacks	9 580

TABLE IV
RUNTIME BREAKDOWN FOR SYNTHETIC BGP DATASET GENERATION.

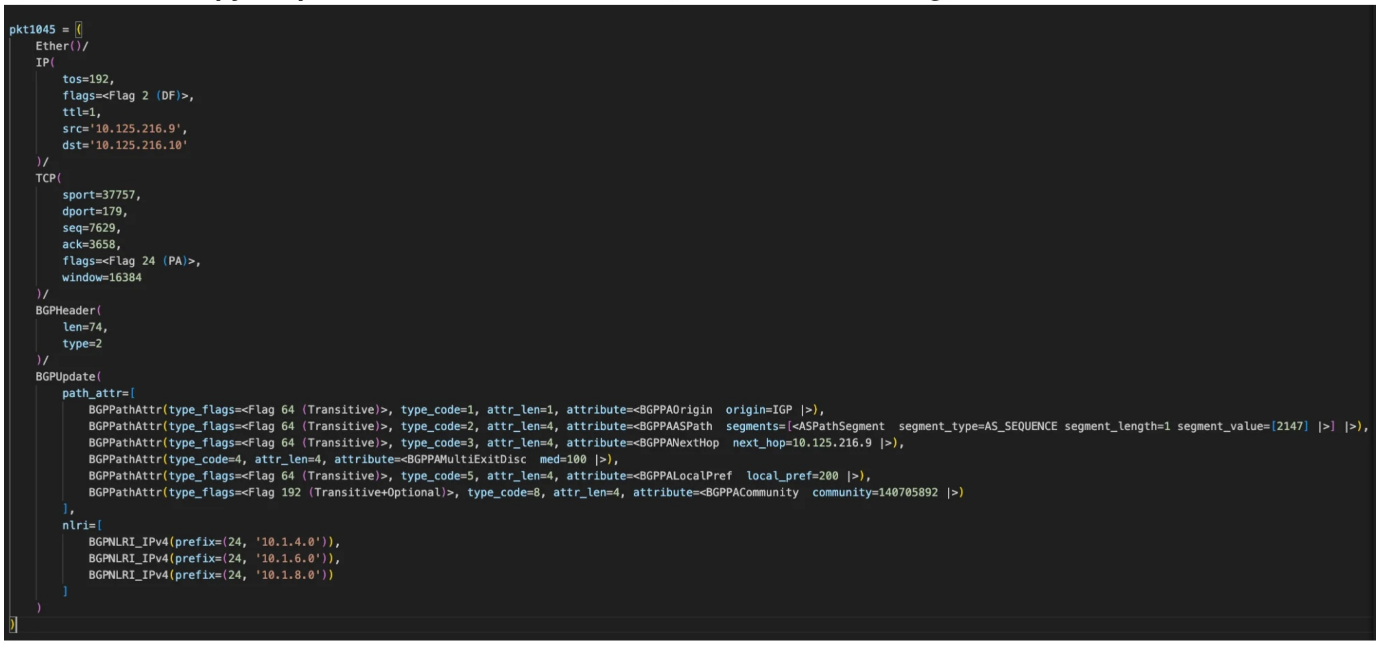
Phase	**Time**
BGP Session Establishment	19 s
BGP Update Generation(Normal/Anomalous)	23.4 min
PCAP to CSV Conversion	21.53 s
Total	**23.81 min**

(A) PCAP-to-Scapy Output Without BGP Module (Basic Parsing Only)



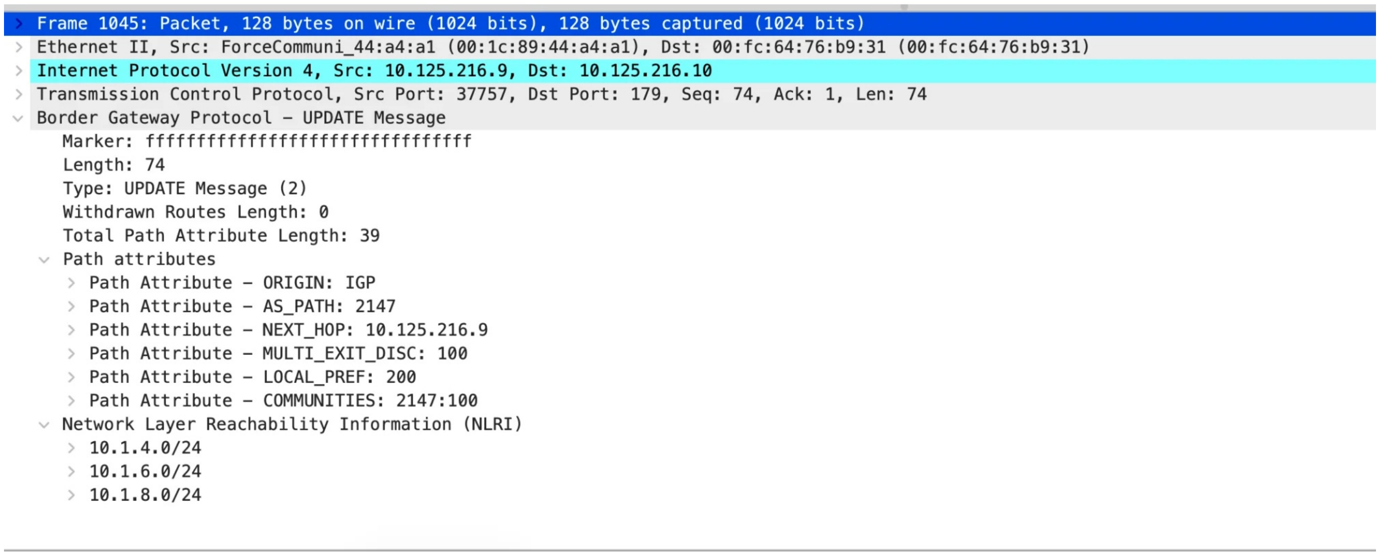
```
pkt1045 = (
    Ether()/
    IP(
        tos=192,
        flags=<Flag 2 (DF)>,
        ttl=1,
        src='10.125.216.9',
        dst='10.125.216.10'
    )/
    TCP(
        sport=37757,
        dport=179,
        seq=7629,
        ack=3658,
        flags=<Flag 24 (PA)>,
        window=16384
    )
)
```

(B) PCAP-to-Scapy Output With BGP Contrib Module (Full Attribute Parsing)



```
pkt1045 = (
    Ether()/
    IP(
        tos=192,
        flags=<Flag 2 (DF)>,
        ttl=1,
        src='10.125.216.9',
        dst='10.125.216.10'
    )/
    TCP(
        sport=37757,
        dport=179,
        seq=7629,
        ack=3658,
        flags=<Flag 24 (PA)>,
        window=16384
    )/
    BGPHeader(
        len=74,
        type=2
    )/
    BGPUpdate(
        path_attr=[
            BGPPathAttr(type_flags=<Flag 64 (Transitive)>, type_code=1, attr_len=1, attribute=<BGPPAOrigin  origin=IGP |>),
            BGPPathAttr(type_flags=<Flag 64 (Transitive)>, type_code=2, attr_len=4, attribute=<BGPPAASPath  segments=[<ASPathSegment  segment_type=AS_SEQUENCE segment_length=1 segment_value=[2147] |>] |>),
            BGPPathAttr(type_flags=<Flag 64 (Transitive)>, type_code=3, attr_len=4, attribute=<BGPPANextHop  next_hop=10.125.216.9 |>),
            BGPPathAttr(type_code=4, attr_len=4, attribute=<BGPPAMultiExitDisc  med=100 |>),
            BGPPathAttr(type_flags=<Flag 64 (Transitive)>, type_code=5, attr_len=4, attribute=<BGPPALocalPref  local_pref=200 |>),
            BGPPathAttr(type_flags=<Flag 192 (Transitive+Optional)>, type_code=8, attr_len=4, attribute=<BGPPACommunity  community=140705892 |>)
        ],
        nlri=[
            BGPNLRI_IPv4(prefix=(24, '10.1.4.0')),
            BGPNLRI_IPv4(prefix=(24, '10.1.6.0')),
            BGPNLRI_IPv4(prefix=(24, '10.1.8.0'))
        ]
    )
)
```

(C) Wireshark Dissection Confirming RFC Compliance



```
> Frame 1045: Packet, 128 bytes on wire (1024 bits), 128 bytes captured (1024 bits)
> Ethernet II, Src: ForceCommuni_44:a4:a1 (00:1c:89:44:a4:a1), Dst: 00:fc:64:76:b9:31 (00:fc:64:76:b9:31)
> Internet Protocol Version 4, Src: 10.125.216.9, Dst: 10.125.216.10
> Transmission Control Protocol, Src Port: 37757, Dst Port: 179, Seq: 74, Ack: 1, Len: 74
v Border Gateway Protocol - UPDATE Message
    Marker: ffffffffffffffffffffffffffffffff
    Length: 74
    Type: UPDATE Message (2)
    Withdrawn Routes Length: 0
    Total Path Attribute Length: 39
  v Path attributes
    > Path Attribute - ORIGIN: IGP
    > Path Attribute - AS_PATH: 2147
    > Path Attribute - NEXT_HOP: 10.125.216.9
    > Path Attribute - MULTI_EXIT_DISC: 100
    > Path Attribute - LOCAL_PREF: 200
    > Path Attribute - COMMUNITIES: 2147:100
  v Network Layer Reachability Information (NLRI)
    > 10.1.4.0/24
    > 10.1.6.0/24
    > 10.1.8.0/24
```

Fig. 1. Bidirectional validation of generated BGP traffic using PCAP-to-Scapy conversion and Wireshark dissection confirming RFC compliance.