BAB II

TINJAUAN PUSTAKA

2.1 *Distributed Denial of Service (DDoS)*

Distributed Denial of Service (DDoS) adalah serangan siber yang dilakukan dengan cara mengirimkan lalu lintas dalam jumlah yang besar dari banyak perangkat ke satu target untuk mengganggu atau menghentikan layanan. Serangan ini memanfaatkan sumber daya jaringan sehingga sistem tidak mampu untuk merespon permintaan yang sah.

Seiring berkembangnya sistem pertahanan DDoS , muncul varian baru yang sulit untuk dideteksi yaitu, *Low-Rate Distributed Denial of Service* (LRDDoS) karena varian ini menyerupai lalu lintas yang normal. Alih alih menghabiskan *bandwidth* dan sumber daya jaringan, serangan LRDDoS ini menargetkan kelemahan pada lapisan protokol. Penyerang mengirimkan paket serangan berbahaya dengan laju yang rendah, sehingga sistem keamanan berbasis jaringan tidak dapat mengenali ciri- ciri serangan tersebut. Oleh karena itu, LRDDoS menjadi tantangan baru dalam keamanan jaringan dan menarik untuk diteliti lebih dalam, khususnya dalam konteks IoT dan SDN [1][16].

2.2 *K-Nearest Neighbors* (KNN)

K-Nearest Neighbors (KNN) adalah metode dalam *machine learning* yang digunakan untuk klasifikasi terhadap sekumpulan data yang sudah terklasifikasikan sebelunya. Algoritma ini bekerja berdasarkan pada jarak terpendek mulai dari sampel uji ke sampel latih untuk menentukan KNN-nya. Setelah mengumpulkan KNN, kemudian diambil mayoritas dari KNN untuk dijadikan prediksi dari sampel uji. Dekat jauhnya jarak biasanya dihitung berdasarkan jarak *Eucledian* [17]. Berikut langkah – langkah untuk menghitung metode *K-Nearest Neighbors* (KNN):

1) Menentukan parameter K,
2) Menghitung jarak antara *data train* dan *data testing*,

Perhitungan jarak yang paling umum dipakai pada perhitungan pada algoritma KNN adalah menggunakan jarak *Euclidean*. Rumusnya adalah :

$$euc = \sqrt{(\sum_{i=1}^{n}(pi - qi)^2)} \quad (2.1)$$

Pada rumus 2.1 dimana :

pi : sampel data / data training

qi : data uji / data testing

i : variabel data

n : dimensi data

3) Mengurutkan jarak yang terbentuk,
4) Menentukan jarak terdekat sampai urutan K,
5) Memasangkan kelas yang bersesuaian,
6) Mencari jumlah kelas dari tetangga yang terdekat dan menetapkan kelas tersebut sebagai kelas data yang akan dievaluasi.

2.3 *Extreme Gradient Boosting* (XGBoost)

Extreme Gradient Boosting (XGBoost) merupakan sebuah kelompok *Decision Tree* yang didasarkan pada *Gradient Boosting* dan dirancang untuk menjadi sangat *scalable*. Seperti *gradient boosting,* XGBoost membuat ekspansi tujuan secara additif dengan *minimizing loss function* oleh itu XGBoost hanya menangani *Decision Tree* sebagai *Base Classifers*, variasi *loss function* digunakan untuk mengontrol kompleksitas *tree* [18]. Adapun rumus dalam penggunaan algoritma XGBoost sebagai berikut :

$$L_{xgb} = \sum_{i=0}^{n} L\left(y_{i,} F(x_i)\right) + \sum_{m=1}^{M} \cap (h_m) \quad (2.2)$$

Pada rumus (2.2) dimana:

L_{xgb} = fungsi kerugian yang ingin diminimalkan oleh model XGBoost secara keseluruhan.

$\sum_{i=0}^{n} L\left(y_{i,} F(x_i)\right)$ = Menunjukkan penjumlahan dari 0 hingga n, dimana n adalah jumlah data secara keseluruhan. Ini adalah fungsi kerugian antara nilai sebenarnya yi dan prediksi F(xi) pada data ke-iii.

$\sum_{m=1}^{M} \cap (h_m)$ = Menunjukkan penjumlahan dari 1 hingga M, di mana M adalah jumlah total pohon dalam model. Ini adalah fungsi regularisasi untuk pohon ke-m yang membantu mengontrol kompleksitas model.

2.4 *Ensemble Stacking*

Stacking adalah sebuah prosedur umum dimana sebuah *learner* akan di *training* untuk menggabungkan beberapa *individual learner* yang disebut *first-level learners*, sedangkan yang menggabungkan disebut *second-level learner* atau *metalearner*. *Stacking* atau *Stacked Generalization* adalah teknik lain untuk menggabungkan *multi learner*. Tidak seperti *bagging* dan *boosting*, *stacking* digunakan untuk menggabungkan *classifier* yang berbeda. *Stacking* terdiri dari dua level yaitu *base learner* sebagai level-0 dan *stacking* model *learner* sebagai level-1 atau *meta learner*. *Base learner* (level-0) menggunakan model berbeda untuk belajar dari suatu dataset. *Output* dari masing – masing model dikumpulkan untuk membuat dataset baru. dalam dataset baru, setiap *instance* berhubungan dengan nilai sesungguhnya yang seharusnya diprediksi. kemudian dataset tersebut digunakan oleh *stacking model learner* (level-1) untuk memberikan hasil akhir [19].

2.5 *Software Defined Network* (SDN)

Software Defined Network (SDN) atau jaringan terdeteksi perangkat lunak merupakan pendekatan *modern* dalam pengelolaan jaringan komputer yang dirancang untuk mengatasi keterbatasan jaringan tradisional. Jaringan konvensional umumnya menggabungkan fungsi kendali dan pengiriman data dalam satu

perangkat keras, seperti *router* atau *switch*, yang menyebabkan kesulitan dalam konfigurasi, kurangnya fleksibilitas, serta ketergantungan pada vendor tertentu. SDN memecahkan masalah ini dengan memisahkan *control plane* (bidang kendali) dan *data plane* (bidang data), dimana bidang kendali dikelola oleh *controller* secara terpusat, sedangkan *data plane* tetap dijalankan oleh perangkat fisik yang meneruskan paket sesuai instruksi.

Pemanfaatan arsitektur ini memungkinkan pengelolaan jaringan dilakukan secara dinamis, fleksibel, dan efisien tanpa harus merubah konfigurasi langsung pada perangkat keras. Administrator jaringan dapat memantau dan mengatur lalu lintas dengan mudah dari satu titik pusat. Selain itu SDN dapat meningkatkan efisiensi operasional, mempercepat proses adaptasi terhadap perubahan kebutuhan jaringan, memungkinkan otomasi layanan, serta membuka jalan bagi pengembangan teknologi jarigan yang lebih cerdas dan terintegrasi [20].

2.6 Ryu Controller

Ryu merupakan *controller* SDN *open source* berbasis Python yang dikembangkan NTT jepang, dirancang secara modular dan berbasis komponen untuk mendukung berbagai protokol seperti *OpenFlow* dan *Netconf* yang berfungsi sebagai otak jaringan SDN dengan mengelola aliran data melalui arsitektur berlapis (fisik, jaringan, aplikasi), menyediakan antarmuka API *northnound* dan *southbound*, serta memungkinkan pengendalian lalu lintas jaringan secara terpusat dan fleksibel melalui integrasi dengan emulator seperti Mininet [21].

2.7 *OpenFlow*

OpenFlow adalah protokol *sounthbound* paling umum dalam arsitektur SDN yang menghubungkan *SDN-Controller* dengan *SDN-Switch* melalui saluran komunikasi berbasis TCP yang dapat diamankan dengan TLS, *controller* mengatur dan memantau *switch* secara langsung menggunakan *flow* Tabel yang berisi aturan, aksi, dan statistik, serta menjalankan proses penemuan topologi jaringan secara berkala melalui *OpenFlow Discovery Protocol* (OFDP) yang memanfaatkan pesan

Link Layer Discovery Protocol (LLDP) yang berguna untuk mendeteksi hubuungan antar perangkat jaringan [22].

2.8 Mininet-IoT

Mininet adalah emulator jaringan yang berbasis Linux yang digunakan untuk menciptakan lingkungan virtual jaringan SDN secara ringan dan efisien. Simulasi topologi memungkinkan lengkap dengan *host*, *switch*, dan *controller* tanpa memerlukan perangkat fisik [23].

2.9 TcpReplay

Tcpreplay merupakan utilitas *open-source* yang digunakan untuk menyunting dan memutar ulang *(replay)* paket-paket jaringan yang telah direkam sebelumnya dalam format *.pcap. Dalam penelitian ini digunakan untuk mereproduksi lalu lintas jaringan nyata dari dataset ke dalam lingkungan uji SDN secara terjadwal, sehingga memungkinkan pengujian sistem klasifikasi lalu lintas berbasis *deep-learning* secara realistis dalam kondisi online. *Tcpreplay* dapat mengirimkan file dengan format *.pcap. *tcpreplay* juga mampu mengatur kecepatan dalam pengiriman paket yang biasa disebut *packet per second* (pps) [24].

2.10 Python

Python adalah bahasa pemrograman yang populer dan banyak digunakan oleh industri maupun komunitas pengembang untuk membangun berbagai jenis aplikasi mulai dari dekstop, web, hingga *mobile*. Bahasa pemrograman ini pertama kali dikenalkan oleh Guido van Rossum. Seseorang pengembang yang berasal dari Belanda. Salah satu keunggulan bahasa pemrograman python yaitu tersedianya beragam *library* yang mempermudah dalam proses pengembangan. Dalam penelitian ini, beberapa *library* yang digunakan antara lain : pandas, numpy, sklearn, matplotlib, dan pickle.

2.11 Penelitian Terdahulu

Pada Tabel 2.1 merupakan beberapa penelitian terdahulu yang digunakan penulis sebagai referensi dan acuan dasar dalam melakukan penelitian :

Tabel 2. 1 Penelitian Terdahulu

No	Penulis,Tahun	Judul	Dataset	Metode	Hasil
1	Achmad Irfani Nur Iman, Fauzi Dwi Sumadi, Zamah Sari,2023	Low Rate DDOS Attack Detection Using KNN On SD-IOT [10]	Dataset dibuat sendiri oleh penulis	*K-Nearest Neighbors (KNN)*	Hasil eksperimen untuk 10 pps mendapatkan *accuracy, precision, recall, f1-score* sebesar 50.51%. Hasil eksperimen untuk 20 pps sebesar 92.71%. Hasil eksperimen untuk50 pps sebesar 13.21%. Hasil eksperimen untuk 100 pps sebesar 58.47%. Hasil eksperimen untuk 200 pps sebesar 62.28%.
2	Wahyuli Dwiki Nanda, Fauzi Dwi Setiawan Sumadi,2022	LRDDoS Attack Detection on SD-IoT Using	Mendeley Data dengan judul Low Rate DDoS (MQTT)	*Random Forest* dengan *Logistic*	Hasil untuk 50 Packet pps mendapatkan accuracy, precision, recall,

		Random Forest with Logistic Regression Coefficient [11]		*Regression Coefficient*	f1-score sebesar 92.3%. Hasil untuk 100 pps sebesar 98.2%, 98.2%. Hasil untuk 200 pps sebesar 98.7%.
3	Fauzi Dwi Setiawan Sumadi, Christian Sri Kusuma Aditya,2020	Comparative Analysis of DDoS Detection Techniques Based on Machine Learning in OpenFlow Network [12]	Mendeley Data dengan judul SDN-DDOS (ICMP,TCP, UDP)	SVM (RBF), SVM (LIN), KNN, DTC, RFC, MLP, GNB	Hasil untuk 420.000:18.000 menggunakan SVM (RBF) mendapatkan *accuracy, precision, recall, f1-score* sebesar 100%. Hasil untuk SVM (LIN) sebesar 100%. Hasil untuk KNN sebesar 87%, 93%, 88% dan 86%. Hasil untuk DTC sebesar 100%. Hasil untuk RFC sebesar 84%, 86%, 84% dan 83%. Hasil untuk MLP sebesar 34%, 32%, 36% dan 23%. Hasil untuk GNB

					sebesar 93%, 95%, 94% dan 93%. Hasil untuk 420.000:36.000 menggunakan SVM (RBF) mendapatkan *accuracy, precision, recall, f1-score* sebesar 100%. Hasil untuk SVM (LIN) sebesar 100%. Hasil untuk KNN sebesar 95%, 89%, 92% dan 87%. Hasil untuk DTC sebesar 89%, 78%, 83% dan 80%. Hasil untuk RFC sebesar 80%, 88%, 84% dan 82%. Hasil untuk MLP sebesar 65%, 52%, 66% dan 54%. Hasil untuk GNB sebesar 59%, 68%, 83% dan 70%.
4	Muhammad Abizar,	Low-Rate distributed	Mendeley Data dengan	SVM (LIN),	Hasil pada metode LR, hampir semua

	Muhammad Ferry Sptian Ihzanor Syahputra, Ahmad Rizky Hhabibullah, Christian Sri Kusuma, Fauzi Dwi Setiawan Sumadi,2023	denial of service attacks detection in software defined network-enabled internet of things using machine learning combined with feature importance [13]	judul LRDDoS dataset (CoAP)	SVM (RBF), RF, DT, MLP, GNB, ADB, KNN	algoritma yang diuji seperti SVM (LIN), DT, MLP, GNB, ADB dapat mencapai akurasi, presisi, recall, dan F1 score sebanyak 100% dengan classification sebanyak 4%. Algoritma GNB menjadi hasil yang paling unggul karena membarikan hasil klasifikasi yang paling sempurna. Sedangkan pada algoritma KNN dan SVM (RBF) menunjukkan hasil yang paling buruk dengan akurasi hanya sekitar 51.88%, presisi 25.94%, recall 50%, dan F1-score 34.16%. Pada metode RFC menunjukkan hasil yang kurang stabil

					terutama pada kecepatan 70 pps menunjukkan classification loss mencapai 51.21%. meskipun demikian algoritma seperti SVM (LIN), DT,GNB,dan ADB tetap mempertahankan performa yang sempurna dengan nilai akurasi, presisi, recall, dan F1-score mencapai 100%. GNB menujukkan performa meningkat lebih tinggi yaitu 51,88% di 20 pps, dan 100 % di 50 pps dan 70 pps. Pada metode RFR hampir semua algoritma berhasil mencapai di nilai 100% pada seluuruh tingkat

					kecepatan paket. Algoritma GNB kembali menjadi yang tercepat dengan waktu yang dibutuhkan hanya 0.013 detik. Hal ini menunjukkan efisiensi yang luar biasa baik dalam performa maupun sumber daya. Pada metode ini, algoritma RF dan DT mencatat hasil yang sedikit lebih rendah yaitu akurasi 90.35%, recall 89.98%, dan F1-Score 90.17%.
5	Muhammad A, Rudianto A, Sakti Pramukantoro E, Kurnianingtyas D	Implementasi Sistem Deteksi Anomali pada Jaringan Komputer dengan Pendekatan XGBoost dan	Mendeley data dengan judul "Towards generating realistic SNMP-MIB dataset for network	XGBoost	Hasil pada metode XGBoost mampu memberikan performa yang sangat tinggi pada tahap evaluasi model, di mana hampir seluruh kelas serangan seperti HTTP

		Data SNMP[14]	anomaly detection".		Flood, ICMP-ECHO Flood, Normal, TCP-SYN Flood, dan UDP Flood memperoleh nilai akurasi, presisi, recall, dan F1-score mendekati 1.00. Hal ini membuktikan bahwa algoritma XGBoost sangat efektif dalam mempelajari pola serangan berdasarkan fitur-fitur SNMP yang digunakan dan mampu melakukan klasifikasi secara akurat pada kondisi pengujian model yang sesuai dengan data latih. Namun, pada tahap pengujian sistem inferensi secara langsung, hasil dari

					algoritma XGBoost menjadi kurang stabil. Pada skenario deteksi statis, sistem hanya mampu memperoleh akurasi sebesar 62.52% dan hasil tersebut dinilai inkonklusif oleh penulis karena adanya kesalahan dalam eksekusi skenario pengujian. Meskipun demikian, sistem inferensi yang diimplementasikan dengan algoritma XGBoost tetap menunjukkan efisiensi sumber daya yang baik. Sistem hanya membutuhkan rata-rata waktu deteksi sekitar 11.5 ms, dengan

					penggunaan CPU sekitar 2.52% dan RAM sekitar 186 MB, sehingga cukup ringan untuk dijalankan secara kontinu. Hasil ini menunjukkan bahwa algoritma XGBoost sangat unggul dalam efisiensi dan performa pada data yang sesuai, tetapi memiliki keterbatasan dalam generalisasi terutama untuk mendeteksi serangan berbasis aplikasi dan serangan low-rate yang tidak terwakili dengan baik dalam pola data pelatihan.

fBerdasarkan Tabel 2.1 di atas, penggunaan algoritma tunggal dalam mendeteksi serangan LRDDoS masih memiliki keterbatasan, terutama dalam hal stabilitas performa pada berbagai variasi *packet per second* (pps). Algoritma seperti

KNN menunjukkan performa yang tidak konsisten, terutama pada kondisi di mana pola trafik serangan menyerupai trafik normal.

Meskipun beberapa algoritma lain mampu mencapai akurasi tinggi, performa tersebut tidak selalu stabil pada seluruh skenario pengujian. Hal ini menunjukkan bahwa tidak ada satu algoritma yang mampu menangani kompleksitas pola LRDDoS secara optimal.

XGBoost mampu menghasilkan akurasi yang tinggi pada data pelatihan dan pengujian, dan membangun model berbasis *gradient boosting* yang dapat menangkap hubungan non-linear antar fitur serta meningkatkan akurasi melalui proses pembelajaran iteratif, tapi berpotensi mengalami bias pada kondisi tertentu.

Oleh karena itu, diperlukan pendekatan yang mampu menggabungkan keunggulan dari beberapa algoritma untuk meningkatkan akurasi dan generalisasi model. Dalam penelitian ini penulis akan menggunakan metode *Ensemble Stacking* dengan *K-Nearest Neighbor* (*KNN)* dan *eXtreme Gradient Boosting* (*XGBoost)* dan menggunakan dataset yang sama seperti pada penelitian [13], yang diperoleh dari situs Mendeley Data dengan judul LRDDoS Dataset (CoAP) [15]. Diharapkan dengan menambah metode *Ensemble Stacking* mampu meningkatkan tingkat akurasi pada metode KNN pada penelitian [13].