Computer Communications 184 (2022) 56–63

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A resource allocation deep active learning based on load balancer for network intrusion detection in SDN sensors

Usman Ahmed [a], Jerry Chun-Wei Lin [a,*], Gautam Srivastava [b,c]

[a] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, 5063, Bergen, Norway
[b] Department of Mathematics and Computer Science, Brandon University, 270 18th Street, Brandon, Canada, R7A 6A9
[c] Research Center for Interneural Computing, China Medical University, Taichung 40402, Taiwan

ARTICLE INFO

Keywords:
Software-defined networking (SDN)
Network performance
Intelligent load balancing
Autonomous
Security

ABSTRACT

Dynamic traffic in a software-defined network (SDN) causes explosive data to flow from one system to another. The explosive data affects the functionality of system parameters, network-level configuration, routing parameters, network characteristics, and system load factors. Adapting to the traffic flow is a key research area in SDN in today's big data world. Load balance vehicular sensor accessibility reduces delays, lowers energy consumption, and decreases the execution time. This paper combines the entropy-based active learning model to identify intrusion patterns efficiently, which is a packet-level intrusion detection model. The developed afterload balancing model can track the attack on the network. We then proposed a load balancing algorithm that optimizes the vehicular sensor usability by using sensor computing capability and source needs. We make use of a convergence-based mechanism to achieve high resource utilization. We then perform experiments on the state-of-the-art intrusion detection dataset. Our experimental results show that the load balancing mechanism can achieve 2× in performance improvements compared to traditional approaches. Thus, we can see that the designed model can help improve the decision boundary by increasing the training instance through pooling strategy and entropy uncertainty measure.

1. Introduction

The application of the Internet of things (IoT) and distributed computing has enabled a massive growth of heterogeneous applications. The future of IoT will connect many heterogeneous devices with the ability to communicate with the network directly [1]. Billions of objects (i.e., sensors network) will be connected to the Internet in the next generation of networks. This will result in extensive amounts of data that give rise to data delivery issues. Objects may include home application, traffic flow analysis, irrigation systems—these objects are usually equipped with several sensors or nodes. The role of these sensors/nodes is to gather and analyze real-time environments.

Connected sensors with autonomous vehicles will also grow and become the future of intelligent transportation systems. Travel comfort, road security and safety will depend upon high data rates and reliable connectivity among autonomous vehicles. Such a transformation will increase the need for a safe and convenient network environment from transportation and transport infrastructure. These sensors are designed to acquire real-time data, and efficient processing is required for better performance. The gathered data is then being used by cloud computing-based applications [2]. The applications can store, process, and update the data in real-time. The applications have centralized data centers that are distributed geographically. Fog computing-based application services are handled at the network's edge. The application can deliver high efficiency and throughput in a distributed environment. The integration of IoT with cloud and fog computing has the potential to make the IoT more efficient to operate at the distributed level and provide increased uptime and cost-efficiency. These factors help improve the functionality of computing, storage and networking resources [1,3]. Thus, enabling fast interaction of data resulting in low latency. Both fog and cloud computing will be connected to billions of smart things and IoT gateways. IoT gateways consist of several small gateways through wireless sensor networks.

Software define networks (SDN) are expected to be a key component of the next-generation (5th generation wireless network) that integrates IoT with human-based services. SDN help to organize heterogeneous networks by following tasks.

1. **Transport**: a massive amount of data generated by terminals, sensors, and nodes.
2. **Allocation**: of computing resource and storage sources in distributed data centers.
3. **Processing**: as well as collection of data efficiently.

* Corresponding author.
E-mail addresses: usman.ahmed@hvl.no (U. Ahmed), jerrylin@ieee.org (J.C.-W. Lin), srivastavag@brandonu.ca (G. Srivastava).

https://doi.org/10.1016/j.comcom.2021.12.009
Received 24 August 2021; Received in revised form 12 October 2021; Accepted 12 December 2021
Available online 21 December 2021

SDN can be reconfigurable wireless networks that gather the required data efficiently [4]. SDN is designed to enable the authentication of users and sensors to define accessibility rules. Current issues with IoT networks are associated with data collection and security and privacy of the collection's pertaining data. Efficient data collection and analysis remain an open research question; however, with the control over the network by SDN use. This makes the network faster, as well as easier to manage. SDN can redirect traffic (number of packets) when needed [5]. This redirection helps IoT applications deal with data latency tasks. Due to network reconfiguration, networks are required to be more responsive, efficient, and secure. Artificial Intelligence (AI) enables SDN to provide massive advantages regarding security. SDN can answer both networking and security issues often encountered in IoT networks. SDN is a networking framework used to overcome IP networks' issues, which are hard to manage due to reconfiguration. The reconfigurable wireless networks (RWNs) are composed of nodes that can perform the task defined by the software. The task includes the reconfiguration of nodes, control of hardware as well as protocol associated with them. The reconfiguration is achieved by including and excluding the node task and node reconfiguration, which also includes updating routing protocols.

1.1. Motivation

Different scenarios based on network configurations have been proposed in many works [6]. However, many networks are not compatible with one another due to the nature of application and type of data and protocols used. The vehicular network and its application have different requirements from the networking of sensors [7]. For example, an emergency application requires low network latency. Sometimes the application may demand dynamic bandwidth for traffic flows between users. Other IoT ecosystems issues include efficient data transportation from data sources (i.e., sensors) to data processing and storage centers.

For this reason, the purpose of this paper is to address the scalability of sensors nodes as well as the operational cost of network services for deploying the infrastructure [8]. The proposed model balances the load among heterogeneous nodes. The task distribution among different nodes is done in a load-balancing manner. In this way, a massive amount of data will be transported from nodes to processing centers smoothly. This enables the SDN to instantly deliver the analysis tasks and improving resource utilization, throughputs, and tasking executions [9]. The reconfigurable network's security is also being investigated by deploying an active learning-based algorithm that expands its knowledge over time. The learning algorithm detects nodes' activity and performs the task to defend against different types of attacks. Deep learning-based active learning methods optimize and efficiently expand their knowledge by learning from the instances associated with it. In this way, the user-centric learning-based mechanism can deliver data from resource to source and autonomously learn to adapt to reconfigurable environments while maximizing the available resources' utility.

1.2. Contribution

In a reconfigurable SDN, a vast number of the papers discuss the node routing problem. The various authors used heuristic-based solutions by using the network structure. However, heterogeneous applications require low data latency and an increase in resource utilization. Therefore, this paper proposes the load balance sensor task execution in a vehicular network connected to an SDN. Our model uses the current node states and balance the sensors load to improve resource utilization, throughput, and execution time. Our goal is to optimize the resource utilization of sensor data defined by the SDN in the cluster of interconnected sensors. The acceleration is enabled by gathering data from sensors in a distributed manner; they minimize the requested data from the heterogeneous application and increasing resource utilization. To summarize, the main contributions of this paper are:

1. Development of an SDN load balancing algorithm to increase data delivery and maximize resource utilization.
2. Development of packet-level intrusion detection using deep active learning.
3. An empirical analysis of the proposed algorithm and its comparison with the state-of-the-art algorithms.

We organize the paper as follows. In Section 2, we represent a different load balancing algorithm. Section 3 presents the system architecture and performance evaluation for the proposed load balancing heuristic. Section 4 presents the experimental setup and results. Section 5 discussion of future direction and conclude the research work.

2. Related work

We presented a few types of research concerning the scheduling and load balancing problem. The effect of load balancing methods helps device suitability. The load balancer was able to distribute the device's loads. The selected algorithms have a specific effect: load balancing and optimization for specific application needs. Like another load balancing, SDN also needs to find the best load balancing method. Various methods are proposed that achieve load balancing among the different device, edge nodes, controller and server [2].

The controller-based load balancer can increase the control plane that results in the SDN controller's decentralization. In [4], the authors proposed the RLMD method to balance traffic load in the multi-controller SDN network. The authors proposed the multi-partition-based algorithm to balance communication of controller, switches and node [10]. The model is then compared with the typical methods and manages to better resource allocations in dynamic traffic load. The algorithm can work under multiple controllers.

Lin et al. [11] addressed the traffic congestion in the mobile vehicular network and analyzed its impact. The work was further extended by Yi et al. for the route identification in the traffic jams scenarios [12]. Yi et al. used the graph-based approach to reach charging stations. The optimization algorithm's weakness is that it used all nodes with an unlimited supply of energy. Hence, this unrealistic hypothesis was removed by Yi et al. [13] by using the minimum cost flow algorithms for multi-users. Another vehicular sensor optimization model was proposed that finds the energy consumption and charging solution concerning the road paths. The model can be incorporated with the current transportation system [14]. Another model was proposed for the vehicular network for sensors, and energy-efficient processing [15].

The scheduling method is also analyzed based on the storage strategy for the data flow [16]. The analysis is done based on the data flow scheduling and hierarchical storage algorithms to detect the edge data flow [16]. The edge computing method combined with scheduling measures is used to define the uplink data flow. Another method proposed is the software defines resource allocation based on the security and location of the services [17]. The method utilized the software resource demands to schedule the tasks among different sensors [18]. The learning-based method is also proposed that uses the heterogeneous arrangement and recovery model [18]. Then, optimize the scheme based on the heterogeneous prior functions via supervised learning. The method used prior learning, and signal recoveries are also completed [18].

A migration-based ElasticCon method is proposed in [7]. A pool is balance incrementally, and the performance of the networks is improved in dynamic traffics. However, the model has a high overhead [19]. The proposed model search for the optimized switch to migrate the load balancing task. The resultant mapping helps to reduce controller load and avoids degrading the performance of the network [20]. The authors are also done a load balancing task on multi-controller architecture in the network. The oscillation problem is resolved with the proposed network—the controller partition control traffic among the different available controllers.

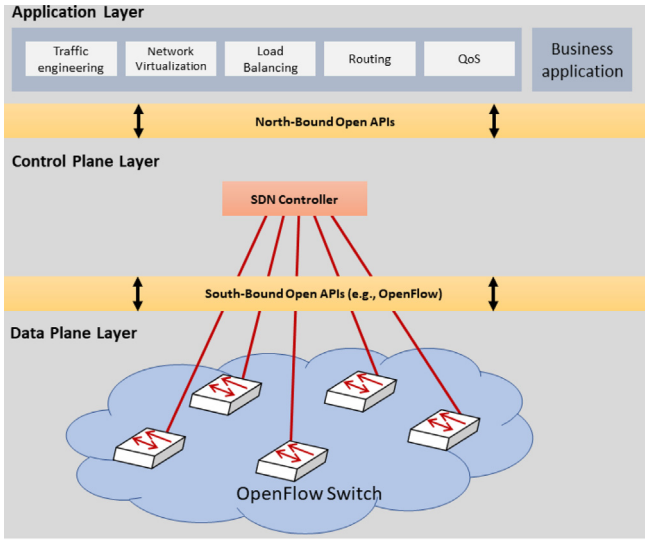

Fig. 1. Overview of the software define networks.

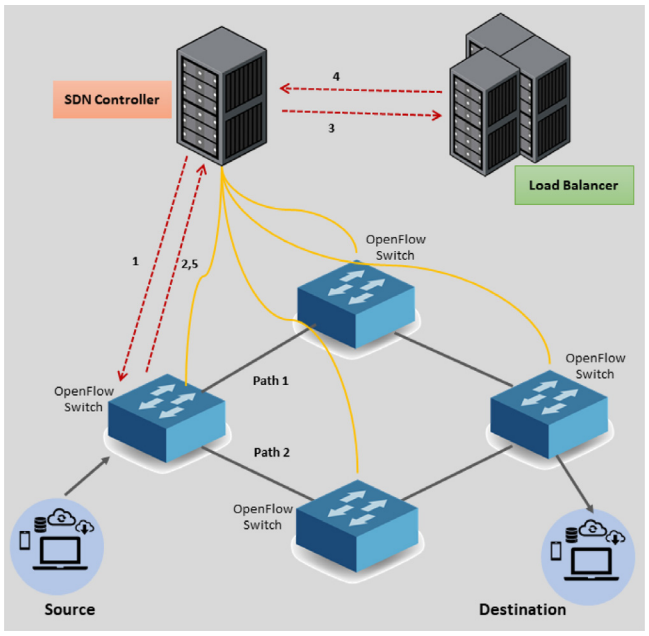

Fig. 2. Proposed SDN load-balancing algorithm framework.

The authors also proposed the server-based load balancing [21]. Then model distributes the clients among multiple servers. The least server is used as a migrating server—the author's performance with round-robin and random-based model. The author's model can achieve good network performance. The GA-based model proposed by [22]. The model selects multiple clients with multiple requests to balance a load. GA's application uses a roulette wheel as the selection method, single-point crossover, and single point mutation. The coefficient-based server model is used as fitness values. The model was compared with random and round-robin methods.

The model is optimized to balance the cloud consumer application [23]. The particle swarm optimization (PSO), the average application response time, maximized throughput and improve resource utilization. The online traffic analysis maps load dynamically. The pool-based servers balancing achieve high performance. Researcher in [9,24–28] implemented similar allocating requests to the server. The SDN based load balancer mainly focus to balance resource and application task efficiently. The SDN ability is to map the distribution dynamically. The congestion-aware multi-meter load balancing also has been achieved during the dynamic increase of the network users.

3. Methodology

In this section, we describe the proposed load-balancing model. The common framework is depicted clearly in Fig. 1. The architecture is composed of three different planes, i.e., **Application** plane, **Control** plane and **Data** plane, respectively. The Application plane provides the user interface for the application-oriented task. In the Control plane, all the decisions are made intelligently and controlled by the SDN. The Control plane implements the security policies, i.e., packet forwarding and controlled rules [2]. The northbound Application Programming Interface's (API) role is to provide the communication interface between the application and control plane. The communication APIs are mostly open-source since we can make easy modifications to fulfilled the application needs. The last plane is a Data plane, also called the infrastructure plane, and it is comprised of devices, i.e., routers, switches and bridges, etc. The OpenFlow (southbound API) is mostly adopted for the communication of control and data plane. The OpenFlow protocol makes communication between the SDN controller and device very easy.

Fig. 2 shows the SDN structure that provides application-oriented flexibility such as load balancing. The SDN structure helps in the customization of applications, which leads to fulfilling the demand for any heterogeneous services. Because of the increased volume of data, intelligent load balancing is one of the main demands of heterogeneous applications. The balancing helps to optimize the network in terms of resource utilization, throughputs while also minimizing congestion. The load balancing method also helps in the application's predictive analyses and provides useful insight into the network and its application. We mention the framework in Fig. 2, describing the load balancing flow. The SDN controller can access all of the network information intelligently at a lower level and communicate with the upper network layers, as can be seen in Fig. 1. The information includes the infrastructure layout, applications, devices connected with them. The SDN controller takes advantage of the global view of the network to optimize load balancing tasks. The network topology and connected devices information help in the application's execution-oriented task. The architecture shows the SDN approach in Fig. 2.

We attempt to clearly show the proposed algorithm flow in Fig. 3. The algorithm takes the input of sensors, heterogeneous application data, as well as the topology of the network at execution time. Then, the application requested is mapped according to the minimum completion of each sensor's task. The balancer has a two-parts, resource selector and a load balancer. The SDN controller provides the load of the sensors for the requested data. We select the maximum loaded sensor. Then, data gathering required by the minimum time to complete is selected as a migrating task. Next, it is moved to the sensors that have loaded less than *(total load/number of the sensor)* and have the minimum time requirement for completing critical application data. If the SDN controller does not find any machines, then the end minimum completion time sensor is selected, and the loop continues until it migrates the data. After the migration, then the required data request is removed from the selected sensor. In this way, we remove selected sensor data from the load balancing task. This process continues until we distribute data among all the machines after that resource utilization ratio is calculated and compared. We propose a convergence-based mechanism. If the resource utilization improves, then the new resource utilization value is set to old resource utilization and sensor data mapping is saved, and we set the convergence point to zero. If there is no improvement in resource utilization, the convergence value is incremented. We set the value of the convergence to half of the requested data from the source

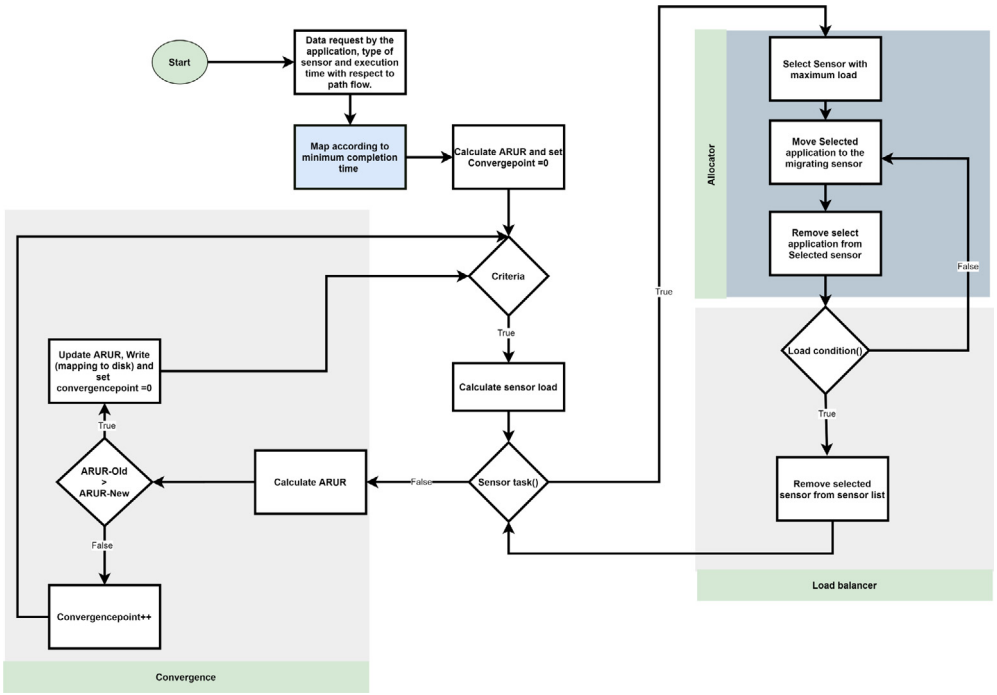

Fig. 3. Flowchart of the proposed algorithm.

Table 1
Preliminary notations used in the proposed system.

Notations	Description
E	Application execution time on each sensor.
$M_i task$	Application assigned to each sensor.
$E_{apps[M_i]}$	Set of application execution time on the ith sensor.
$Total - E$	Set of sensors total execution time
$Load calculation$	Set of sensors load
$Total_{Load}$	Sensors total load
$Application_{map}$	Set of sensors job mapping
$convergencepoint$	Criteria to stop: Convergence value is repeated to the $\frac{numberof application}{2}$
$ARUR_Old$	$\frac{mean(ready_{time} - Sensors)}{makespan}$ before Updation
$ARUR_New$	$\frac{mean(ready_{time} - Sensors)}{makespan}$ after Updation
$selected_{application}$	A application (having minimum execution time for the $selected_{sensor}$) that migration needed
$selected_{sensor}$	A sensor which is overloaded
$mig - sensor$	A sensor which have load less than $\frac{Total_{ET}}{numberof sensors}$ and minimum completion time for the application
$M_i job$	ith sensor jobs

application during the empirical analysis. The convergence value shows we achieve maximum resource utilization. The proposed algorithm has modules, i.e., the resource allocator *(lines 4 to 19, Algorithm 1)* and load balancer *(lines 6 to 13, Algorithm 2)*. The symbol table mentioned in Table 1 explains the algorithm variables.

In Algorithm 1, the input is the execution time of all applications requested by sensors and output is load-balanced mapping of application on all sensors. The minimum execution time based mapped is performed initially. Then, the convergence point is set to zero *(Line 2, Algorithm 1)*. Afterwards, the average resource utilization of the minimum execution time mapping is calculated. Then the convergence loop runs on the sensors regarding load *(Line 3, Algorithm 1)*. The resource allocator produces the load balance application mapping *(Lines 4 to 19, Algorithm 1)*. The execution time of all applications is the sum of each sensor. The sensor's execution is the sum to get the total execution time. A load for each sensor is calculated by taking the fraction of sensors from execution time to total execution time *(Lines 4 to 19, Algorithm 1)*. We select the maximum load of sensors. Then selected sensor minimum execution time application is migrated to other sensors *(Lines 6 to 13, Algorithm 2)*. The loop continues with the remaining application until the sensor's load is equal to the total execution time divided by the number of the sensor. After migrating the applications, the loop selects another sensor and starts migrating the new selected sensors application *(Lines 6 to 13, Algorithm 2)*. In this way, we attain new mapping for the network. Then, the resource utilization ratio is calculated by using formula i.e., $\frac{mean(ready_{time} - Sensors)}{makespan}$ *(Lines 20 to 27, Algorithm 1)*. If the values improved, then the convergence point is set to zero *(Line 8 of Algorithm 1)*, and mapping is saved. If there is no improvement, then the convergence is incremented.

4. Evaluation

We compare the proposed model with MCT, Min–Min and Max–min resource allocation algorithms, current well-known state-of-the-art

Table 2
Dataset samples.

Application	Sensor 1	Sensor 2	Sensor 3	Sensor 4	Sensor 5	Sensor 6	Suitable
1	0.18	0.60	0.18	0.60	0.62	0.49	Sensor 2
2	0.99	0.15	0.99	0.15	1.39	0.16	Sensor 3
3	2.62	2.14	2.62	2.14	3.99	1.96	Sensor 4
4	1.27	1.62	1.27	1.62	0.35	2.79	Sensor 5
5	4.08	3.83	4.08	3.83	4.70	3.08	Sensor 6
6	0.55	0.34	0.55	0.34	0.76	0.25	Sensor 6
7	0.55	0.34	0.55	0.34	0.76	0.25	Sensor 6
8	0.56	0.31	0.56	0.31	0.82	0.21	Sensor 6
9	0.56	0.31	0.56	0.31	0.88	0.18	Sensor 6
10	0.24	0.17	0.24	0.17	0.30	0.24	Sensor 1

Algorithm 1 Proposed Algorithm

INPUT: Execution time of all sensor per application data.
OUTPUT: Load Balanced mapping of application on the sensor

Select $MCT()$;
$convergencepoint \leftarrow 0$;
$ARUR_{old}()$;
while $convergence < Con_{cric}$ **do**
  for $i \leftarrow 0$ **do** number of sensors
    $E_{apps[M_i]} \leftarrow E_{A1} + \ldots + E_{Jn}$;
  end for
  $Total_E \leftarrow \sum_{i=0}^{n} E$;
  for $i \leftarrow 0$ **do** number of sensors
    $Loadcalculation()$;
    $Total_{Load}()$;
  end for
  $t \leftarrow 0$;
  while $length[sensor]$ **do**
    $selected_{sensor}$;
    $Application_{map}[\,] \leftarrow Load_{balancer}()$;
    Remove S_i;
    $t{+}{+}$;
  end while
  $ARUR_{new}()$;
  if $ARUR_{new} > ARUR_{old}$ **then**
    $convergencepoint \leftarrow 0$;
    $ARUR_{old} \leftarrow ARUR_{new}$;
    $Write_{application-set}$;
  else
    $convergence{+}{+}$;
  end if
end while

Algorithm 2 Load Balancer

INPUT: (1) execution time of all application through on every sensor; (2) selected sensor application and total Load on it; and (3) total load on the selected sensors.
OUTPUT: Application mapping of the selected sensors.

while $Total_{load}()$ **do**
  $Selected_{sensor}$;
  for $i \leftarrow 0$ **do** number of machines
    $new_{sensor} \leftarrow mig - sensor$;
    if $new_{sensor} < \frac{Total_{ET}}{\text{number of sensors}}$ **then**
      $new_{sensor} \leftarrow mapApp_{selected_{job}}$;
      Remove $selected_{application}$ from $Sensors$;
    else
      Remove $new_{application}$ from $selected_{sensors}$;
    end if
  end for
end while
Return $App_{map}[]$.

algorithms [29,30]. The performances are compared on several tests by varying the application and number of sensors. The makespan, throughput and resource utilization metric are used as performance metrics [29,30]. The makespan is the last finish time of any sensors in SDN networks. The better algorithm gives the lowest values of makespan. The average resource utilization ratio is measure in terms of the resource utilization status of the network for the given set of tasks. The excellent algorithm has closer to 1. The throughput is the number of tasks per unit time. The better algorithm has a higher value.

The sample dataset is mentioned in Table 2. It contains 6 sensors that executes the application in seconds. The sensors with the lowest values of execution time are selected as suitable sensors. However, the energy and other factors depend upon the mapping. If the sensor is closer to the switch, it will produce less energy to send the application data resulting in slower execution. If the sensor is away from the switch, then it will take more time. Moreover, if we map all the required application gather data, then the sensor closer to the network will be overload, and the resources will be underutilized. Therefore, the proposed algorithm balances the load in terms of execution time. We consider two datasets, i.e., synthetic 1 and synthetic 2. The first dataset, i.e., synthetic 1, has the six sensors' application execution time. We evaluated it under a different set of the application. In the second dataset, we have 50 sensors and applications to execute different categories, i.e., tiny, small, medium, large and extra-large.

In the comparison of synthetic dataset 1, the proposed algorithm is compared with the different algorithms, as mentioned in Table 3. The results are compared in terms of execution time, throughput and resource utilization ratio. The proposed heuristic method can reduce execution time, increase resource utilization ratio and throughput. The model can achieve 12% decrease executions compared to Max–min and 17% improved ARUR compared to Min–min. The dataset includes the smaller jobs and proposed model is able to map the sensors in load balancer by considering the execution time. The convergence factor helps to improve performance. The model has proven efficient under different conditions, but we find its performance significantly more effective with increasing network sizes (see Table 4).

Synthetic dataset 2 has 10% longer jobs. This is because that the execution time for the proposed model is increased with the increase of the ARUR and throughput, as seen in Table 3. On each iteration, the proposed model can check convergence on each mapping. If the resource utilization improves then convergence is set 0; otherwise, it increments the convergence point. The mapping of the previous ARUR is saved and used if there is no further improvement. The load balancing for each node in the network of the over utilizes node. The mapping is optimal as able to converge after a few iterations.

The computational complexity is also mentioned in Table 5. The complexity of the model increases with the increase of the application data. However, it remains linear when we have an increase in the number of sensors.

4.1. A case study — Intrusion detection using Deep active learning

In this section, we present the experimental planning to detect a DDOS attack on SDN. As the network is a heterogeneous application,

Table 3
Synthetic dataset 1 (number of sensors (6)) results on different algorithms.

Application	Proposed			Max–min			Min–min		
Size	Makespan	ARUR	Throughput	Makespan	ARUR	Throughput	Makespan	ARUR	Throughput
200	24.22	0.9965	8.26	26.64	0.88	7.27	29.01	0.83	7.17
300	36.05	0.9986	8.32	39.66	0.88	7.32	43.59	0.86	6.93
400	45.98	0.9988	8.70	50.58	0.89	7.66	55.09	0.85	7.24
500	56.48	0.9992	8.85	62.13	0.90	7.88	65.18	0.85	7.43
600	67.48	0.9993	8.89	74.23	0.88	7.82	82.32	0.85	7.72
700	80.86	0.9993	8.66	90.57	0.90	7.79	93.54	0.84	7.33
800	99.31	0.9997	8.06	110.24	0.89	7.17	115.85	0.86	6.62
900	105.86	0.9997	8.50	117.51	0.88	7.48	129.36	0.83	7.01
1000	119.97	0.9997	8.34	131.97	0.89	7.42	143.72	0.87	7.22

Table 4
Synthetic dataset 2 (number of sensors (50)) results on different algorithms.

Application	Proposed			Max–min			Min–min		
Size	Makespan	ARUR	Throughput	Makespan	ARUR	Throughput	Makespan	ARUR	Throughput
400	742.46	0.93	0.54	831.55	0.82	0.47	858.87	0.80	0.46
450	780.35	0.96	0.58	866.19	0.85	0.51	927.61	0.82	0.48
500	913.80	0.94	0.55	1023.45	0.85	0.49	1098.02	0.78	0.46
550	963.14	0.97	0.57	1059.45	0.86	0.50	1132.65	0.82	0.48
600	1094.18	0.98	0.55	1225.48	0.86	0.49	1265.74	0.81	0.48
650	1148.05	0.98	0.57	1285.81	0.87	0.51	1379.49	0.85	0.47
700	1269.16	0.98	0.55	1408.77	0.86	0.50	1536.82	0.85	0.48
750	1328.82	0.98	0.56	1461.71	0.87	0.50	1591.93	0.80	0.49
800	1473.67	0.97	0.54	1621.04	0.87	0.49	1716.83	0.82	0.46
850	1516.02	0.98	0.56	1697.94	0.88	0.50	1821.65	0.82	0.47

Table 5
Computational complexity : *M= Number of sensors in the network and N= Number of application in the network.*

Heuristic	Complexity
Min Min, Max Min	$O(MN^2)$
Propose	$O(MN^2)$

Table 6
Dataset used for active learning.

Traffic label	No. of training records	No. of test records
Normal	67,343	9,711
U2R	52	67
R2L	995	2,887
Probe	11,656	2,421
DOS	45,927	7,548

Table 7
Network.

Layer type	Output	Param
Dense 1	1,024	43,008
Dropout	1,024	0
Dense 2	768	787,200
Dropout	768	0
Dense 3	512	3,937,378
Dropout	512	0
Dense 4	256	131,328
Dropout	256	0
Dense 5	256	32,896
Dropout	128	0
Dense 6	1	129
Activation	1	0

there is a chance of a DDOS attack after reconfiguration of the wireless network. The model uses the learning to learn a self-training method to examine and expand knowledge.

The major challenge in machine learning applications is to structure high-quality and meaningful data. This task is costly and laborious. The active learning mechanism helps to learn from the low number of instances and chose the data distribution to the label by the users and made machine learning more applicable. Various real-life applications are thus demanded to mine interesting and meaningful patterns from the data [31].

In this paper, we classify different data packets with different types of attacks, i.e., DoS, U2R, R2L and Probing attacks [32]. In this paper, the feedforward network is used to classifies the attacks into five different categories of attacks, as mentioned in Table 7. The entropy-based method is utilized to select the data distribution from the unlabeled dataset. The model takes advantage of the deep learning mechanism to learn from a limited set of examples for pattern identification. The initial training set contains a small amount of data and an entropy-based model to decide which points to select for inclusion in the training set. The entropy-based model in this paper selects the number of a point depending upon the pool size. The pool point is the separate set that is updated each cycle. It includes the selected point in the training set, and we train an alternative model on the new points mentioned in Fig. 4. The future wireless network method helps in the selection of instances that directly affect the infrastructure. The repetition of the steps helps to increase with training-set and meaningful points over time. The infrastructure-aware instance selection can help to reduce load balancing tasks. The developed method can help reduce data annotation tasks and generalize the machine learning system [31].

The dataset we use to train the model is the KDD dataset [32]. We were initially developed by the MIT lincoln labs to test intrusion detection methods. We use the upgraded KDD dataset — NSL-KDD mentioned in Table 6. The dataset contains different attacks regarding the flow of the traffic. Each record in the dataset is assigned to collect packets information under different network traffic flow. The attacks include **Denial of service attack:** intrusion where flooding happens to disrupt the services hence overloading the host. **User to root attack** is used to access the User pre-exciting access and exploit them. **Remote Local (R2L):** The intrusion attacker tries to access a user account that tries to explore the vulnerability to obtain access to local access. **Probing attack:** the type in which the intruder tries to get the networks access.

We first transfer the item's features through the dense 1024 dense units. We then pass the developed model through more 1024 [33,34]. The last layer was `sigmoid`, and we use Adam optimizer for the

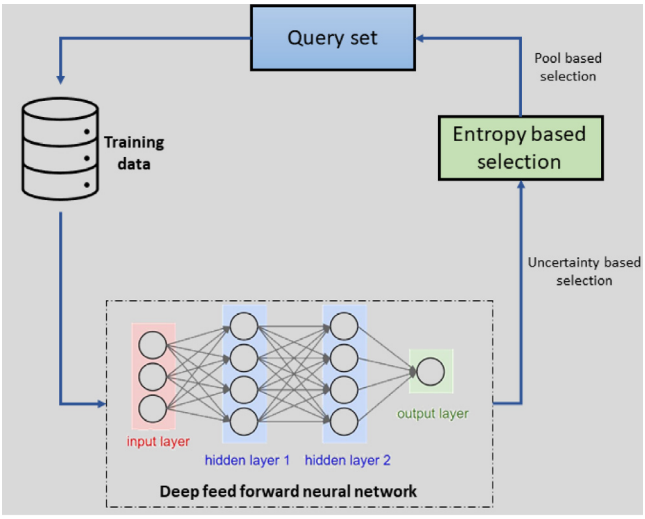

Fig. 4. Entropy based active learning for intrusion detection.

learning rate. We train the network for 100 epochs on each pool. The dataset distribution is the training set (40%), validation set (20%) and unlabeled pooling set (40%).

This paper takes into account self-learning by using the mechanism of deep learning with entropy-based selection. The model is compared with [32]. The model is also compared with traditional models. The model is trained and tested on a public dataset. The deep learning models have also compared with interns of the hidden layer, and it is concluded that the three-layer network is not accurate and efficient. Precision tells about the exactness of the prediction. In this case, the precision value tells about the correct prediction of the fusion suitability classifier. Therefore, the higher value of precision (which is 96% in this case as shown in Table 8) indicates it shows that an active learning classifier can, mostly, correctly predict whether an application possess any attack. If the value of precision is lower, it would be an indication that the classifier is not exact and is unable to identify attacks.

Similarly, recall tells about the completeness of the prediction. In this case, the recall value tells about the correct identification of attacks that belong to a suitable class. Therefore, the higher value of recall (which is 98% in this case as shown in Table 8) indicates that the classifier can, mostly, correctly predict applications that belong to the same class. If the value of precision were lower, it would be an indication that the classifier is not complete and is unable to identify applications that belongs to the same class.

Precision measures the relevant results (exactness of the prediction) rather than irrelevant results and recall measures sensitivity of the most relevant results (completeness of the prediction). According to the recommendation of Geron [35], precision and recall are convenient to combine into a single metric called F1 score. The F1 score is the harmonic mean of precision and recall. The harmonic means gives more weight to lower values as compared to the regular mean. The F1 — the measure is a widely known metric that estimates the entire system performance by combining the precision and recall [36]. The scale of the F1 measure is 1.0: perfect prediction, 0.9: excellent prediction, 0.8: good prediction, 0.7: mediocre prediction, 0.6: poor prediction, 0.5: random prediction and less than 0.5: poor prediction [36]. If the percentage values of precision and recall were lower for the classifier, then applications attacks remain undetected. The fusion suitability classifier will only get a high F1 score if both precision and recall are high. Therefore, in our case, recall is 0.98 and precision is 0.98, but our F1 score is 0.96.

Table 8
Results comparison with deep active learning.

Algorithm	Accuracy	Precision	Recall	fl-score
DNN-1	0.929	0.998	0.915	0.954
DNN-2	0.929	0.998	0.914	0.954
DNN-3	0.930	0.997	0.915	0.955
DNN-4	0.929	0.999	0.913	0.954
DNN-5	0.927	0.998	0.911	0.953
Ada Boost	0.925	0.995	0.911	0.951
Decision tree	0.928	0.999	0.912	0.953
K-nearest neighbor	0.929	0.998	0.913	0.954
Linear regression	0.848	0.989	0.821	0.897
Navic Bayes	0.929	0.988	0.923	0.955
Random forest	0.927	0.999	0.910	0.953
SVM*-Lincar	0.811	0.994	0.770	0.868
SVM*-rbf	0.811	0.992	0.772	0.868
Active learning	0.94	0.96	0.98	0.96

5. Conclusion

This study presented a novel load balancing algorithm to balance the load of SDN among different vehicular sensors by using network and application information. The proposed model can balance the batch of applications among the vehicular sensor connected over the SDN. We tested our approach on different datasets, and the outcome of the proposed model has been compared with well-known heuristic-based models. Moreover, we used an entropy-based active learning approach to classify intrusion attacks. The developed model can achieve high accuracy to identify the patterns in terms of sparse and dense datasets. The entropy-based active learning-based method significantly increases training instances for the deep feedforward model. In the future, the network will be optimized tuned to apply the active learning mechanism. A weighted-based method for each class sub-sample selection can also be considered as a further extension.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was partially supported by the Western Norway University of Applied Siences, Bergen, Norway and was partially funded by the Natural Sciences Research Council of Canada (NSERC) Discovery Grant program (RGPIN-2020-05363) held by Dr. Gautam Srivastava.

References

[1] A.A. Neghabi, N.J. Navimipour, M. Hosseinzadeh, A. Rezaee, Nature-inspired meta-heuristic algorithms for solving the load balancing problem in the software-defined network, Int. J. Commun. Syst. 32 (4) (2019) e3875.
[2] T. Semong, T. Maupong, S. Anokye, K. Kehulakae, S. Dimakatso, G. Boipelo, S. Sarefo, Intelligent load balancing techniques in software defined networks: A survey, Electronics 9 (7) (2020) 1091.
[3] G. Srivastava, N. Deepa, B. Prabadevi, P.K. Reddy M, An ensemble model for intrusion detection in the Internet of Softwarized Things, in: Adjunct proceedings of the 2021 international conference on distributed computing and networking, 2021, pp. 25–30.
[4] T. Hu, P. Yi, J. Zhang, J. Lan, Reliable and load balance-aware multi-controller deployment in SDN, China Commun. 15 (11) (2018) 184–198.
[5] H. Sufiev, Y. Haddad, L. Barenboim, J. Soler, Dynamic SDN controller load balancing, Future Internet 11 (3) (2019) 75.
[6] Y. Zhou, M. Zhu, L. Xiao, L. Ruan, W. Duan, D. Li, R. Liu, M. Zhu, A load balancing strategy of SDN controller based on distributed decision, in: IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2014.
[7] Y. Hu, W. Wang, X. Gong, X. Que, S. Cheng, BalanceFlow: Controller load balancing for OpenFlow networks, in: IEEE International Conference on Cloud Computing and Intelligence Systems, 2012.

[8] J. Yu, Y. Wang, K. Pei, S. Zhang, J. Li, A load balancing mechanism for multiple SDN controllers based on load informing strategy, in: The Asia-Pacific Network Operations and Management Symposium, 2016.
[9] A.K. Arahunashi, G.G. Vaidya, N. S, K.V. Reddy, Implementation of server load balancing techniques using software-defined networking, in: The International Conference on Computational Systems and Information Technology for Sustainable Solutions, 2018.
[10] H. Sufiev, Y. Haddad, L. Barenboim, J. Soler, Dynamic SDN controller load balancing, Future Internet 11 (3) (2019) 75.
[11] G. Lin, P. Yi, L. Si, T. Zhu, X. Jiang, G. Li, M.M. Begovic, Robustness analysis on electric vehicle energy distribution networks, in: 2013 IEEE Power & Energy Society General Meeting, IEEE, 2013, pp. 1–5.
[12] P. Yi, T. Zhu, G. Lin, Q. Zhang, Routing renewable energy using electric vehicles in mobile electrical grid, in: 2013 IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems, IEEE, 2013, pp. 19–27.
[13] P. Yi, Y. Tang, Y. Hong, Y. Shen, T. Zhu, Q. Zhang, M.M. Begovic, Renewable energy transmission through multiple routes in a mobile electrical grid, in: ISGT 2014, IEEE, 2014, pp. 1–5.
[14] A.Y. Lam, Y.-W. Leung, X. Chu, Electric vehicle charging station placement: Formulation, complexity and solutions, IEEE Trans. Smart Grid 5 (6) (2014) 2846–2856.
[15] Y. Wang, Z. Su, N. Zhang, BSIS: Blockchain-based secure incentive scheme for energy delivery in vehicular energy network, IEEE Trans. Ind. Inf. 15 (6) (2019) 3620–3631.
[16] P. Liu, S. Lyu, S. Ma, W. Wang, Optimization algorithm of wireless surveillance data transmission task based on edge computing, Comput. Commun. 178 (2021) 14–25.
[17] S. Wu, H. Cao, H. Zhao, Y. Hu, L. Yang, H. Yin, H. Zhu, A softwarized resource allocation framework for security and location guaranteed services in B5G networks, Comput. Commun. 178 (2021) 26–36.
[18] L. Qin, Y. Cao, X. Shao, Y. Luo, X. Rao, Y. Yi, G. Lei, A deep heterogeneous optimization framework for Bayesian compressive sensing, Comput. Commun. 178 (2021) 74–82.
[19] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R.R. Kompella, ElastiCon; an elastic distributed SDN controller, in: ACM/IEEE Symposium on Architectures for Networking and Communications Systems, 2014, pp. 17–27.
[20] C. Liang, R. Kawashima, H. Matsuo, Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers, in: 2014 Second International Symposium on Computing and Networking, 2014.
[21] Y. Zhou, Y. Wang, J. Yu, J. Ba, S. Zhang, Load balancing for multiple controllers in SDN based on switches group, in: The Asia-Pacific Network Operations and Management Symposium, 2017.
[22] F. Chahlaoui, M.R. El-Fenni, H. Dahmouni, Performance analysis of load balancing mechanisms in SDN networks, in: The International Conference on Networking, Information Systems & Security, 2019.
[23] L.D. Chou, Y.T. Yang, Y.M. Hong, J.K. Hu, B. Jean, A genetic-based load balancing algorithm in openflow network, in: Advanced Technologies, Embedded and Multimedia for Human-centric Computing, Dordrecht, 2014, pp. 411–417.
[24] K. Govindarajan, V.S. Kumar, An intelligent load balancer for software defined networking (SDN) based cloud infrastructure, in: The International Conference on Electrical, Computer and Communication Technologies, 2017.
[25] S. Kaur, J. Singh, Implementation of server load balancing in software defined networking, in: Advances in Intelligent Systems and Computing, 2016, pp. 147–157.
[26] H. Kavana, V. Kavya, B. Madhura, N. Kamat, Load balancing using SDN methodology, Int. J. Eng. Res. Technol. 7 (5) (2018) 206–208.
[27] M.I. Hamed, B.M. ElHalawany, M.M. Fouda, A.S.T. Eldien, A new approach for server-based load balancing using software-defined networking, in: The International Conference on Intelligent Computing and Information Systems, 2017.
[28] H. Zhang, X. Guo, SDN-based load balancing strategy for server cluster, in: IEEE International Conference on Cloud Computing and Intelligence Systems, 2014.
[29] U. Ahmed, J.C.-W. Lin, G. Srivastava, M. Aleem, A load balance multi-scheduling model for OpenCL kernel tasks in an integrated cluster, Soft Comput. 25 (1) (2021) 407–420.
[30] U. Ahmed, M. Aleem, Y. Noman Khalid, M. Arshad Islam, M. Azhar Iqbal, RALB-HC: A resource-aware load balancer for heterogeneous cluster, Concurr. Comput.: Pract. Exper. 33 (14) (2021) e5606.
[31] P. Kumar, A. Gupta, Active learning query strategies for classification, regression and clustering: A survey, J. Comput. Sci. Tech. 35 (4) (2020) 913–945.
[32] R.K. Vigneswaran, R. Vinayakumar, K. Soman, P. Poornachandran, Evaluating shallow and deep neural networks for network intrusion detection systems in cyber security, in: The International Conference on Computing, Communication and Networking Technologies, 2018.
[33] M.T. Luong, H. Pham, C.D. Manning, Effective approaches to attention-based neural machine translation, 2015, https://arxiv.org/abs/1508.04025.
[34] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, in: Advances in Neural Information Processing Systems, 2017, pp. 5998–6008.
[35] A. Geron, Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools and Techniques to Build Intelligent Systems, vol. 1, 2019.
[36] F.A. Narudin, A. Feizollah, N.B. Anuar, A. Gani, Evaluation of machine learning classifiers for mobile malware detection, Soft Comput. 20 (1) (2020) 343–357.