A Machine Learning-Based Protocol for Efficient Routing in Opportunistic Networks

Deepak K. Sharma, Sanjay K. Dhurandher, *Senior Member, IEEE*, Isaac Woungang, *Senior Member, IEEE*, Rohit K. Srivastava, Anhad Mohananey, and Joel J. P. C. Rodrigues, *Senior Member, IEEE*

***Abstract*—This paper proposes a novel routing protocol for OppNets called MLProph, which uses machine learning (ML) algorithms, namely decision tree and neural networks, to determine the probability of successful deliveries. The ML model is trained by using various factors such as the predictability value inherited from the PROPHET routing scheme, node popularity, node's power consumption, speed, and location. Simulation results show that MLProph outperforms PROPHET+, a probabilistic-based routing protocol for OppNets, in terms of number of successful deliveries, dropped messages, overhead, and hop count, at the cost of small increases in buffer time and buffer occupancy values.**

***Index Terms*—Decision tree, delay-tolerant networks, machine learning (ML), neural networks, opportunistic networks (OppNets), PROPHET+.**

I. Introduction

OPPORTUNISTIC networks (OppNets) [1] are a type of challenged networks where link performance is highly variable and the network contacts are intermittent. As such, routing in OppNets is a challenge since the nodes are required to buffer their packets till they find suitable forwarders that can eventually carry these packets to destination and with minimal delay.

Routing protocols in OppNets [2] can be broadly categorized into: 1) infrastructure-based routing protocols—for which the presence of some form of infrastructure that augments when forwarding the packets to their destinations is assumed; and 2) infrastructure-less-based routing protocols—where no such assumption prevails and only the contact opportunities are used to route the packets within the network. In this paper, a new infrastructure-less routing protocol for OppNets (called MLProph) is introduced as an improvement of the PROPHET+ routing protocol [4]. MLProph uses a machine learning (ML) technique to train itself based on various factors such as buffer capacity, hop count, node energy, node movement speed, popularity parameter, and number of successful deliveries. The ML algorithm is trained based on the past network routing data in order to yield an equation that computes the probability used to check whether the node in contact will be able to eventually deliver the message to its intended destination. This value is then used to decide on the next hop for the buffered message.

The rest of the paper is organized as follows. In Section II, some related work on routing protocols in infrastructure-less OppNets are presented. In Section III, the proposed MLProph routing protocol is described. In Section IV, the ML process in MLProph is described. In Section V, the simulation results are presented. Finally, Section VI concludes the paper.

II. Related Work

Various routing protocols for OppNets have been proposed in the literature. The most representative ones are described as follows. In [3], Vahdat *et al.* proposed the epidemic protocol, where the sender node floods the network with multiple copies of the message it intends to deliver to the destination node. It does so by distributing a copy of the message to every node that it comes in contact with, which in turn distributes their copy to every neighboring node. This process continues till one of the copies of the message get delivered to the destination node. This protocol has higher eventual delivery rate at the cost of high network resource consumption. In [2], Boldrini *et al.* proposed HiBOp, a context-based opportunistic routing protocol which involves the use of the Identity table that stores the context of the node and the History table that records the attributes from the Identity table of the neighboring nodes in the past. These data structures are used to determine the number of initial copies of original message to be distributed by the sender node and to calculate the delivery predictability based on which the next-hop for a message can be selected. In [5], Dhurandher *et al.* proposed a routing protocol for OppNets (called HBPR) that utilizes contextual information for next hop selection purpose. In their scheme, the behavioral information of a node is taken into account when determining the next best hop that the node should pass the message to, based on the calculation of a direction predictor of the destination node by using a Markov predictor and a utility metric. In [6], Lindgren *et al.* proposed PROPHET, a

Manuscript received July 15, 2016; revised September 20, 2016; accepted October 4, 2016. This work was supported in part by Finep, with resources from Funttel under Grant 01.14.0231.00, under the Centro de Referência em Radiocomunicações—CRR project of the Instituto Nacional de Telecomunicações—Inatel, Brazil, in part by the Government of Russian Federation under Grant 074-U01, and in part by Instituto de Telecomunicações, NetGNA, Covilhã Delegation. The work of J. J. P. C. Rodrigues was also supported by the Fundação para a Ciência e a Tecnologia funding Project UID/EEA/500008/2013.

D. K. Sharma and S. K. Dhurandher are with the CAITFS, Division of Information Technology, Netaji Subhas Institute of Technology, University of Delhi, Delhi 110021, India (e-mail: dk.sharma1982@yahoo.com; s.dhurandher.in@ieee.org).

I. Woungang is with the Department of Computer Science, Ryerson University, Toronto, ON M5B 2K3, Canada (e-mail: iwoungan@scs.ryerson.ca).

R. K. Srivastava is with the Ohio State University, Columbus, OH 43210 USA (e-mail: srivastava.141@osu.edu).

A. Mohananey is with the Department of Software Development Engineering, Infibeam.com, Nehrunagar, Ahmedabad 380015, Gujarat, India (e-mail: anhadmohananey@gmail.com).

J. J. P. C. Rodrigues is with the National Institute of Telecommunications (Inatel), Santa Rita do Sapucaí-MG 37540-000, Brazil, also with the Instituto de Telecomunicações, Universidade da Beira Interior, Covilhã 6201-001, Portugal, and also with the ITMO University, St. Petersburg 197101, Russia (e-mail: joeljr@ieee.org).

Digital Object Identifier 10.1109/JSYST.2016.2630923

TABLE I
INPUT ATTRIBUTES USED FOR THE MLPROPH PROTOCOL

Attribute Name	Symbol	Description
Prophet probability	x_1	Probability used for deciding next hop selection in Prophet routing protocol. Described in detail in Section III-A3
Buffer occupancy	x_2	Amount of capacity left in buffer to keep more packets refer to Section III-A3 for details
Successful deliveries	x_3	Gives the number of successful message transfers from the start of simulation to current time, between the two given nodes
Success ratio	x_4	Indicates the ratio of successful message transferred to total number of transfers initiated between the two nodes
From node speed	x_5	Represent speed at which the sender is traveling
To node speed	x_6	Represent speed at which the receiver is traveling
Distance from message source	x_7	Distance of the location of contact between the two nodes from the point of origin of message
Distance to message destination	x_8	Distance of the location of contact to the final destination of the message
Message live time	x_9	Duration of time from the creation of the message to current time
From node energy	x_{10}	Energy of sender node
To node energy	x_{11}	Energy receiver node
Current hop count	x_{12}	Number of hops the message has traveled before reaching the current sender node

routing protocol for OppNets that relies on the calculation of a delivery predictability table that records the probabilities of successful deliveries of messages from source to destination nodes. Whenever a node encounters other nodes, their delivery predictability values are exchanged and the message is forwarded to those nodes that have the higher delivery predictability values. In [4], Huang *et al.* introduced PROPHET+, an improvement of PROPHET, which consists of using a weighted function to calculate the node's delivery probability when performing the routing as instructed by PROPHET. In [7], the encounter and distance routing protocol is proposed which depends on two parameters for next hop selection: the number of encounters and the distance of the node from the destination. The ratio of these parameters is used to decide on the next hop selection.

III. PROPOSED MLPROPH PROTOCOL

The proposed MLProph protocol uses a ML technique to perform the next hop selection for a message. Whenever a connection is established between two nodes, and the buffer of one node contains a message that needs to be transmitted, a decision needs to be made on whether or not the message should be forwarded to the other node (termed as next hop selection). Intuitively, the message must be only forwarded from the sender to the neighboring receiver node if the intermediate node has a high enough probability of forwarding it directly or indirectly to the destination node. Forwarding the message too frequently can lead to excessive packet loss and higher buffer overhead. On the other hand, less frequent forwarding is a definite precursor to less number of delivered messages. The probability of successful delivery is dependent on various factors that represent the history and capability of nodes to successful deliver the messages. The delivery probability at the next hop selection is computed by a trained ML model involving the following attributes: PROPHET probability, buffer occupancy, successful deliveries, success ratio, from node speed/energy, to node speed/energy, distance from message source, distance to message destination, current hop count, and message live time. The message live time parameter represents the duration of time from the creation of the message to the current time. The message is forwarded from the sender node if $P_m > K \times P_r$, where P_m is the probability of the final delivery (called ML probability) computed using ML techniques, P_r is the probability of the sender node delivering the message to the destination (obtained using PROPHET)—the so-called PROPHET probability—and $K \in [0, 1]$ is a normalization factor.

A. *Calculation of the ML Probability P_m*

Two ML models, namely neural network and decision tree, are used to calculate the value of P_m and to assess the performance of the proposed MLProph routing protocol.

1) Neural Network Model: A neural network model with multiple hidden layers is considered, where $(x_1, \ldots, x_{12})$ are the input parameters as described in Table I constituting the input layer and p_1 and p_2 are the outputs forming the output layer (p_1 and p_2 are, respectively, the probabilities of successful and unsuccessful delivery). The value p_1 is the ML probability P_m, which is the predicted probability of successful delivery based on given values of input, i.e., $(x_1, \ldots, x_{12})$. This value at each node in the neural network is a function of a linear combination of values of nodes in the previous layer. The value at node h_i is given as

$$h_i = F\left(\sum_{j=1}^{n} x_j w_{\text{ji}}\right) \tag{1}$$

where x_j is the value of the *j*th node of the previous layer, F is the activation function, and w is the weight matrix. Let us consider the value of node h_1, which is a linear combination of $(x_1, \ldots, x_{12})$ passed through an activation function F, i.e.,

$$h_1 = F(w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + \cdots + w_{121}x_{12}). \tag{2}$$

Thus, moving from the input layer, the value at each node can be computed, and thereby, the values of the output layer, which will produce P_m (from p_1). This process is referred to as feedforward operation. To perform this step, we must first calculate the appropriate values of the weight matrix w_{ij} for each possible linear combination. This is performed through the training of the set of data gathered by running the simulation on the training scenario, and the neural network is built by finding the corresponding weights.

Training: The neural network is trained by using the Back propagation algorithm [8] and the training data obtained prior to simulation. Each training data sample represents the input values of $(x_1, \ldots, x_{12})$ for every next hop selection, and the obtained output determines whether the message that has been forwarded has eventually reached the destination node or not. If the output was successful $p_1 = 1$ and $p_2 = 0$, otherwise $p_1 = 0$ and $p_2 = 1$. Prior to training, a sample neural network is considered, i.e., initialized with random values, and subsequently, the neural network is learned by iterating over the training set so that it provides the optimum predictions on whether successful delivery will take place or not. To compute the value of p_m, the ML model must first be trained or built based on the data captured in a scenario called the training scenario. The corresponding data is called the training data and a particular entry in the data is called a training example. For each training example in the training data, the following actions are taken.

1) The input is propagated forward to generate the output of the activation functions at each layer. Lets call this $NN_{\text{Prediction}}$. Let NN_{Actual} be the actual values from the training set.
2) The training error is calculated, which is the sum over the output units of the squared difference between desired output (NN_{Actual}) and the prediction ($NN_{\text{Prediction}}$) as defined in the LMS algorithm [9]. The training error of weights w is obtained as

$$J(w) = \frac{1}{2}\sum(NN_{\text{Actual}} - NN_{\text{Prediction}})^2. \quad (3)$$

3) The weights are corrected to reduce the training error $J(w)$. $\delta(w)$, the change that needs to be made in the values of the weights w is computed using equation

$$\delta(w) = -n(dJ/dw) \quad (4)$$

where n is the learning rate representing the relative change in the weights based on the training error.
4) The value of w is updated as follows:

$$w(\text{new}) = w(\text{old}) + \delta(w). \quad (5)$$

The above steps are carried out for all the training examples, resulting in a trained neural network with learned weights, which gives a sufficiently good prediction, and hence, a low prediction error.

Computation of P_m (Feed Forward): The trained neural network is then used to compute P_m, based on the input parameters $(x_1, \ldots, x_{12})$ obtained at real time, during the next hop selection process. The process of moving from the input to the output layer is done by calculating the value at each node, which is the linear combination of values of nodes of the previous layer with an activation function applied to this combination. Equation (2) is used to perform the feedforward process, where w has been obtained from the training step. Moving from the input to the output layer across all the hidden layers, the value of each neuron is computed using (2), which is subsequently also used in the calculations of the node values for the next layer. The final output layer gives p_1 and p_2, where the value of p_1 corresponds to P_m.

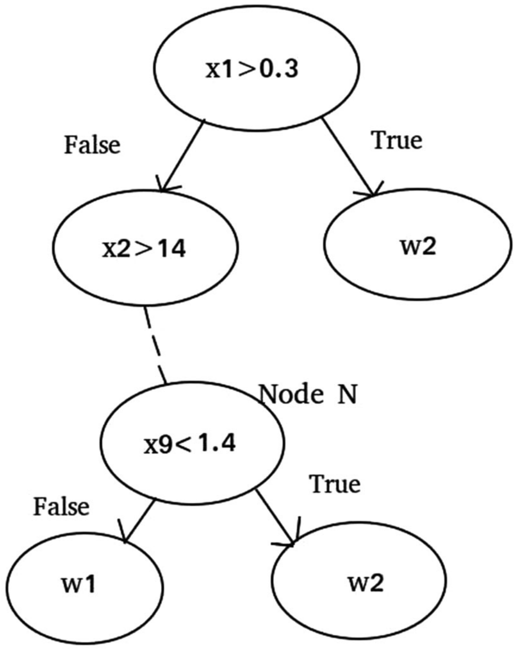


Fig. 1. Sample decision tree.

2) Decision Tree Model: The considered decision tree model is shown in Fig. 1, where w_1 and w_2 are the output classes representing the successful delivery and unsuccessful delivery, respectively, and $(x_1, x_2, \ldots, x_{12})$ are the input attributes, and the internal nodes represent the decisions made based on the input attributes. As an example, in Fig. 1, starting at the root, for a given set of input values, we move towards the leaf nodes. Say the value x_1 is 0.2, then we will move along the left subtree from the root node. On the other hand, if the value of x_1 is 0.8, then the input is predicted to belong to class w_2. The value of P_m is the probability distribution of the decisions falling into the category of the particular class (w_1 or w_2). Assume that we have reached the node N with a value of x_9 as 1.8, then the predicted class becomes w_1, and P_m is defined as the probability of nodes falling into class w_1 from the training set given that node N was reached.

Building the Decision Tree: The decision tree is built using the training data in a recursive manner as follows.

BuildTree(S)

1) Find the most appropriate attribute and corresponding value to use for the decision of the root node. In the above example, x_1 is the attribute chosen in the first recursive call and $x_1 > 0.3$ is the corresponding decision. The most

popular measure for computing the attribute to be used to split the dataset is the entropy impurity [9]

$$i(N) = -\sum P(w_j) \log P(w_j) \tag{6}$$

where $P(w_j)$ is the fraction of the patterns at a particular node that are in category w_j. For splitting, we must choose the query that decreases the impurity as much as possible. The drop in impurity is defined by

$$\delta(i) = i(N) - P(x)i(N_x) - (1 - P(x))i(N_y) \tag{7}$$

where N_x and N_y are the left and right descendant nodes and $i(N_x)$ and $i(N_y)$ are their impurities. For example, if the attributes are $x_1, \ldots, x_{12}$, then the attribute that has the highest value of $\delta(i)$ will correspond to the root of S.

2) Based on the attribute computed from step (1), split the dataset S into S_x and S_y, where S_x corresponds to the left subtree of the root of S and S_y corresponds to the right subtree.
3) Call BuildTree(S_y) and BuildTree(S_y) in a recursive manner to build the descendants of the root.

The above procedure is stopped when the maximum value of $\delta(i)$ falls below a certain threshold. In this paper, the C4.5 implementation of the decision trees [11] and the gain ratio impurity equation are used to compute the value of $\delta(i)$, where a change in the impurity is scaled by dividing it with the entropy of the parent node. The optimized J48 decision tree by the Weka Library [11] is also utilized.

Calculation of P_m: Once the decision tree is fully grown, based on the input attributes, the decision tree conditions are applied until a leaf node is reached. The class computed at the leaf node is used to determine the value of P_m using the probability distribution from the training set. As an example, if class w_1 is predicted, P_m will be obtained as the number of elements in the set in which w_1 was reached from the predecessor node divided by the total number of times the predecessor was reached. The attributes $(x_1, x_2, \ldots, x_{12})$ that have been used by the MLProph protocol to compute P_m for the next hop selection in the neural network and decision tree techniques are given in Table I.

3) Calculation of the Normalization Factor K and PROPHET Probability P_r: Although the PROPHET router has not been used directly for next hop selection purpose, the PROPHET delivery probabilities are continuously updated and used as parameters for the ML algorithm. PROPHET works by updating the probabilities in such a way that the nodes that have frequent interactions have high delivery probabilities

$$P(x,y) = P(x,y)_{\text{old}} + (1 - P(x,y)_{\text{old}}) \times P_{\text{init}}. \tag{8}$$

If some nodes have not been connected recently, their delivery probabilities must also age. This is taken care of by the aging factor as follows:

$$P(x,y) = P(x,y)_{\text{old}} \times (\gamma)^k \tag{9}$$

where γ is the aging factor, and k is time units since the last aging occurred.

PROPHET also considers a transitive nature among nodes. As an example, assume that there are three nodes n_1, n_2, and n_3 such that nodes n_1 and n_2 frequently interact, as well as n_2 and n_3. In this case, a message to be delivered to n_3, if forwarded from n_1, would also have a high delivery probability

$$\begin{aligned} a_{(b_c)} P_{(n_1,n_3)} = \; & a + b + P_{(n_1,n_3)_{\text{old}}} + (1 - P_{(n_1,n_3)_{\text{old}}}) \\ & \times P_{(n_1,n_2)} \times P_{(n_2,n_3)} \times \beta. \end{aligned} \tag{10}$$

This delivery probability will then be used as a parameter for the ML algorithm since it gives a good indication of the node delivery and contact histories, which is essential for the next hop selection.

The normalization factor lies between 0 and 1. Its final value is determined by considering different values of K in between 0 and 1 and observing the corresponding number of delivered messages. The probability of final delivery versus the value of K is a Gaussian curve where the point corresponding to the maximum value of the delivery probability gives K. This value of K is the one to be considered in this work.

The buffer occupancy parameter is indicative of the capacity (in terms of storage) of the potential receiver node to absorb and subsequently forward the message. It is defined as

$$\begin{aligned} \text{Buffer Occupancy} = \; & \text{bufferSize}_{\text{available}} \\ & - \text{messageSize}_{\text{toBeForwarded}}. \end{aligned} \tag{11}$$

IV. ML Process in MLProph

Our proposed MLProph routing protocol solves the problem of next hop selection by computing a delivery probability at each next hop decision situation, which depends on parameters such as node speed, buffer size, contact history, and PROPHET probability. The ML process creates an optimum model where the input is a vector of the parameters at run time, and the output is a delivery probability. The training of the ML model happens prior to simulation based on the same input and output attributes, in an environment indicative of real world scenarios. The ML process is capable of computing the relative importance of the above-mentioned parameters on the next hop decision, yielding optimum calculations of P_m, the delivery probabilities.

A. MLProph Algorithm

The MLProph algorithm for next hop selection is divided into the following parts.

Training: The data is generated for the next hop by running the simulation on the training scenario. Each entry in the data depicts whether the final delivery has occurred for the hop selection in question or not, by sending the message from the sender to the receiver given the parameters described in Table I at that time. In order to ensure that a message is transmitted in all possible cases, this simulation is run by using the Epidemic routing protocol [3].

After the data has been generated, it is used to build the above-mentioned neural network-based and decision tree-based learning models.

Real Simulation: The simulation is started, and every time a message can be transmitted, or the next hop selection decision is to be made, the following actions are taken: 1) capture the

TABLE II
SIMULATION PARAMETERS USED FOR TRAINING THE GROUPS

No. of Groups	4
Interface	Bluetooth
Transmit speed	250 K
Transmit range	10
Buffer size	10 M
Message TTL	1200

PROPHET probability (P_r); 2) compute the ML probability (P_m) based on the considered trained ML model; and 3) forward the message from the sender to the receiver if $P_m > K \times P_r$.

V. PERFORMANCE EVALUATION

A. Simulation Setup

The ONE simulator [10] is used to evaluate the performance of the proposed MLProph routing protocol against that of the PROPHET+ routing protocol [4]. The Weka ML library [11] is used to train and deploy the neural network-based and decision tree-based learning models for the implementation of MLProph.

The simulation environment consists of various groups, where each group is a collection of nodes, which can be configured differently in terms of movement model, node speed, wait time, number of nodes, to name a few. Some simulation parameters such as buffer size, transmit range, Time-to-Live (TTL), and transmit speed, are kept common across all groups. The router can vary across groups, however, for validating the MLProph algorithm, the same router has been used for all groups. The following constitute the main configuration settings.

1) *Movement Model:* This defines the motion of nodes inside a community with respect to a particular scenario.
2) *Transmit Speed:* For an interface, this represents the speed at which the messages will move in the network.
3) *Transmit Range:* This is the maximum distance at which a sender node can send the data to a receiver.
4) *Buffer Size:* This is the space available in the node to store the packets before eventually forwarding or dropping them.
5) *Speed:* This defines the range of speeds that the nodes in the movement model can take.
6) *Number of Hosts:* This is the number of hosts in the group.
7) *Message TTL:* This defines the lifespan of the message in the network.
8) *Wait Time:* This defines the time a group has to wait since the simulation has started before initiating any movement.

For ML purpose, the training data is gathered by using the simulation configurations provided in Table II. After the simulation data has been gathered, the neural network-based and decision tree-based learning models are trained separately by using the Weka Library in Java. The obtained models are then used to build two routers, namely MLProph$_{NN}$ (from the neural network model) and MLProph$_{DT}$(from the decision tree model). The results obtained using PROPHET+, MLProph$_{NN}$, and MLProph$_{DT}$ routers are then compared against each other based on the simulation environment described in Table IV. To test the MLProph protocol, the following varying parameters are considered: 1) varying the buffer: the buffer size for all groups in the network is varied from 20 M → 40 M → 60 M → 80 M → 100 M; 2) varying the TTL: the message TTL is varied from 50 → 100 → 150 → 200 → 250; and 3) varying the number of nodes: the number of nodes in the network are varied from 130 → 180 → 230 → 280 → 330.

TABLE III
INDIVIDUAL GROUP CONFIGURATION FOR TRAINING OF MLPROPH PROTOCOL

Group No.	No. of Nodes	Speed	Wait Time	Movement Model
1	45	7–10	10–30	BusMovement
2	150	0.8–6	0	BusMovement
3	80	7–10	10–30	WorkingDay Movement
4	50	17–25	0	WorkingDay Movement

TABLE IV
CONFIGURATION PARAMETERS USED FOR SIMULATION OF GROUPS

No. of Groups	7
Interface	Bluetooth
Transmit speed	100 K
Transmit range	10
Buffer size	10 M
Message TTL	1433

TABLE V
INDIVIDUAL GROUP CONFIGURATION FOR RUNNING THE SIMULATION OF THE MLPROPH PROTOCOL

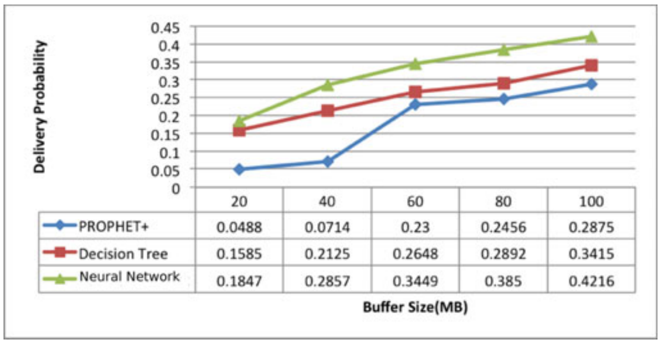
Group No.	No. of Nodes	Speed	Wait Time	Movement Model
1	150	7–17	10–30	BusMovement
2	50	0.2–1.8	0	WorkingDay Movement
3	40	7–10	10–30	BusMovement
4	25	1.7–10	0	WorkingDay Movement
5	14	2–6	5–30	BusMovement
6	50	0.2–1.8	0	WorkingDay Movement
7	40	7–10	10–30	BusMovement

The considered performance metrics are: buffer time, delivery probability, number of packets dropped, overhead ratio, latency, and hop count.

B. Simulation Results

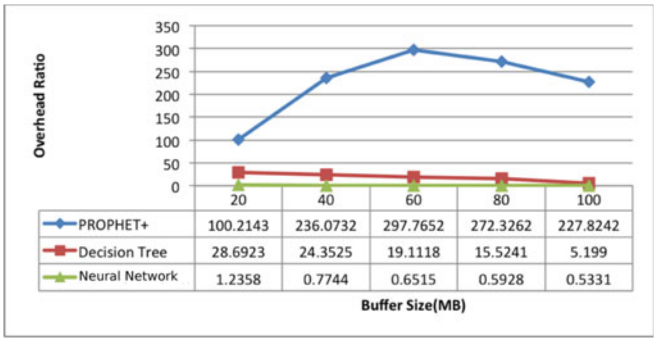
In this section, the performances of MLProph$_{NN}$, MLProph$_{DT}$, and PROPHET$+$ are compared under the aforementioned varying parameters. Table III describes the individual group configuration parameters used for the training of the MLProph protocol, whereas Table V describes the individual group configuration parameters used for running the simulation of the proposed MLProph protocol.

1) Varying the Buffer Size: The buffer size for all groups in the network is varied and the aforementioned performance metrics are evaluated. The results are captured in Figs. 2–4. It can be observed that the average buffer time increases as the buffer size is increased. This is understandable since a large buffer means



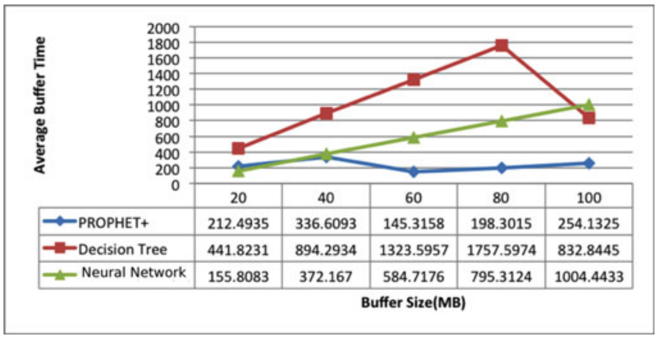
	20	40	60	80	100
◆ PROPHET+	0.0488	0.0714	0.23	0.2456	0.2875
■ Decision Tree	0.1585	0.2125	0.2648	0.2892	0.3415
▲ Neural Network	0.1847	0.2857	0.3449	0.385	0.4216

Fig. 2. Delivery probability with varying buffer size.



	20	40	60	80	100
◆ PROPHET+	100.2143	236.0732	297.7652	272.3262	227.8242
■ Decision Tree	28.6923	24.3525	19.1118	15.5241	5.199
▲ Neural Network	1.2358	0.7744	0.6515	0.5928	0.5331

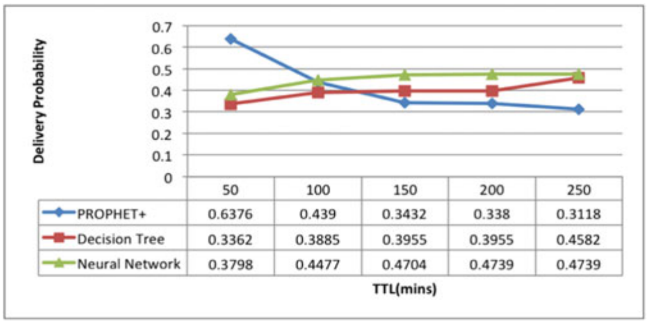
Fig. 3. Overhead ratio with varying buffer size.



	20	40	60	80	100
◆ PROPHET+	212.4935	336.6093	145.3158	198.3015	254.1325
■ Decision Tree	441.8231	894.2934	1323.5957	1757.5974	832.8445
▲ Neural Network	155.8083	372.167	584.7176	795.3124	1004.4433

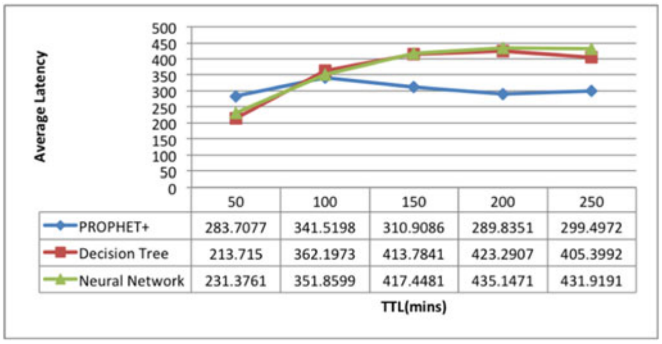
Fig. 4. Average buffer time with varying buffer size.

that the messages are held by nodes for a larger duration of time. The buffer time for the decision tree and neural network models are greater than that obtained for PROPHET+ since the messages are held for a longer time for both models, owing to a strict forwarding criterion, which becomes a precursor to smarter delivery logic. In Fig. 2, it is observed that the delivery probability is directly proportional to the buffer size since a larger buffer means less packet loss. Clearly, MLProph using both the neural network-based and decision tree-based models shows a superior performance compared to PROPHET+ in terms of delivery probabilities. In addition, MLProph$_{NN}$ yields a lower average hop count (which decreases with the buffer size) compared to MLProph$_{DT}$, which indicates that the application of the ML to MLProph produces a smarter next hop selection compared to that generated by PROPHET+. In Fig. 3, it is observed that the



	50	100	150	200	250
◆ PROPHET+	0.6376	0.439	0.3432	0.338	0.3118
■ Decision Tree	0.3362	0.3885	0.3955	0.3955	0.4582
▲ Neural Network	0.3798	0.4477	0.4704	0.4739	0.4739

Fig. 5. Delivery probability with varying TTL.



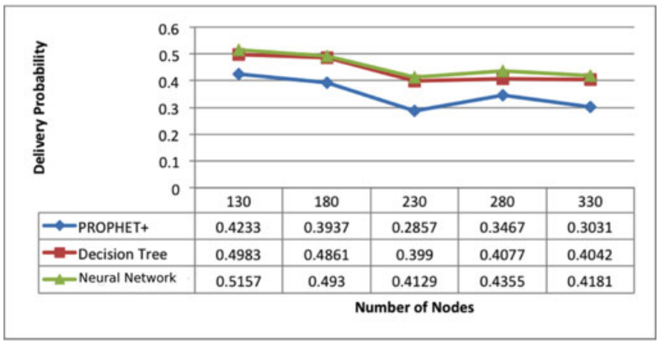
	50	100	150	200	250
◆ PROPHET+	283.7077	341.5198	310.9086	289.8351	299.4972
■ Decision Tree	213.715	362.1973	413.7841	423.2907	405.3992
▲ Neural Network	231.3761	351.8599	417.4481	435.1471	431.9191

Fig. 6. Average latency with varying TTL.

overhead ratios produced by MLProph$_{NN}$ and MLProph$_{DT}$ are significantly lower than that produced by PROPHET+, which is understandable given the low dropped message count and high number of delivered messages experienced by MLProph$_{NN}$ and MLProph$_{DT}$.

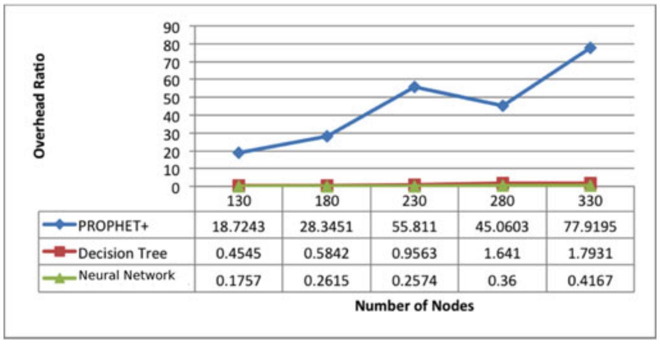
2) Varying the Message TTL: The message TTL is varied and the aforementioned performance metrics are evaluated. The results are captured in Figs. 5 and 6. It can be observed that the delivery probabilities do not vary much when the TTL is varied. The median delivery probabilities across varying TTL values are 0.47, 0.39, and 0.34 for MLProph$_{NN}$, MLProph$_{DT}$, and PROPHET+, respectively, validating the hypothesis that MLProph delivers more messages due to significantly lower packet loss. In terms of overhead ratio, we found that the overhead ratios obtained from the MLProph algorithms are much lesser than that obtained using PROPHET+. Additionally, in Fig. 6, it can be observed that the average latency is higher for MLProph compared to PROPHET+. This is attributed to the fact that the criteria for forwarding a packet is more constrained in MLProph than in PROPHET+, causing the messages to remain in the buffer of nodes for a longer period time, thereby, the messages will eventually take more time to get delivered, albeit with a better delivery probability.

3) Varying the Number of Nodes: The number of nodes in the network are varied and the aforementioned performance metrics are evaluated. The results are captured in Figs. 7–9. It can be observed that both MLProph implementations yield higher values of the delivery probability compared to that obtained by the PROPHET+ router, and MLProph$_{NN}$ outperforms



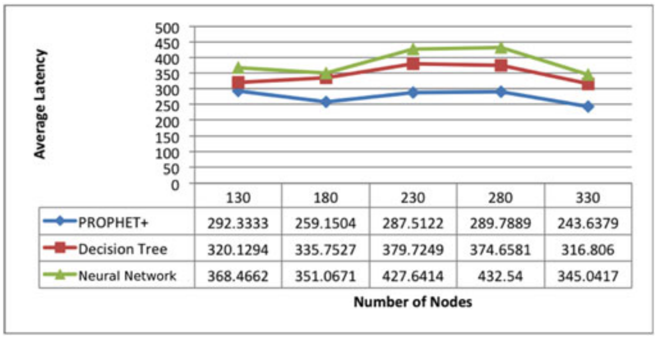
	130	180	230	280	330
PROPHET+	0.4233	0.3937	0.2857	0.3467	0.3031
Decision Tree	0.4983	0.4861	0.399	0.4077	0.4042
Neural Network	0.5157	0.493	0.4129	0.4355	0.4181

Fig. 7. Delivery probability with varying number of nodes.



	130	180	230	280	330
PROPHET+	18.7243	28.3451	55.811	45.0603	77.9195
Decision Tree	0.4545	0.5842	0.9563	1.641	1.7931
Neural Network	0.1757	0.2615	0.2574	0.36	0.4167

Fig. 8. Overhead ratio with varying number of nodes.



	130	180	230	280	330
PROPHET+	292.3333	259.1504	287.5122	289.7889	243.6379
Decision Tree	320.1294	335.7527	379.7249	374.6581	316.806
Neural Network	368.4662	351.0671	427.6414	432.54	345.0417

Fig. 9. Average latency with varying number of nodes.

MLProph$_{DT}$ for the same. It can also be observed that MLProph yields a significant lower overhead ratio compared to that obtained using PROPHET+. Fig. 9 shows that MLProph yields a higher average latency compared to PROPHET+. This is due to the fact the criteria for forwarding a packet is more constrained in MLProph compared to PROPHET+.

4) Confidence Interval of Results: A unique confidence interval was computed for each observable parameter on the combined dataset of values observed when the buffer size, TTL, number of nodes, and transmit range are varied. The 95% confidence intervals is summarized in Table VI. From this table, it can be seen that MLProph in both implementations (neural network and decision tree) yields substantially lower values of the number of packets dropped and overhead ratio compared to PROPHET+. The delivery probability is higher at the expense of a minor increase in the average buffer time.

TABLE VI
95% Confidence Interval for Different Results

Observed Attribute	PROPHET+	MLProph (Neural Network)	MLProph (Decision Tree)
No. of packets dropped	14331–30258	492–559	660–1725
Delivery probability	0.23–0.38	0.36–0.46	0.31–0.43
Overhead ratio	93–195	0.36–0.65	2–11
Average latency	210–295	297–402	275–367
Average hopcount	2.48–3.15	1.04–10.8	1.07–1.19
Average buffer time	234–300	690–1049	810–1183

VI. Conclusion

This paper has proposed MLProph a novel ML-based routing protocol for OppNets. Simulation results have shown that the performance of MLProph is superior to that of PROPHET+ in terms of delivery probability, overhead ratio, with a payoff in the average latency, and buffer size due to the constraints imposed on the next hop selection process by MLProph. It has also been shown that in terms of delivery probabilities, packet loss, and overhead ratio, the MLProph$_{NN}$ model is preferable for use compared to the MLProph$_{DT}$ model. In future, we will simulate MLProph using real mobility traces. We will also investigate the use of other ML classifiers such as support vector machines.

References

[1] L. Lilien, Z. H. Kamal, V. Bhuse, and A. Gupta, "Opportunistic networks: The concept and research challenges in privacy and security," in *Proc. NSF WSPWN*, Miami, FL, USA, Mar. 2006, pp. 134–14.

[2] C. Boldrini, M. Conti, I. Iacopini, and A. Passarella, "HIBOP: A history based routing protocol for opportunistic networks," in *Proc. IEEE WOWMOM*, Espoo, Finland, Jun. 2007, pp. 1–12.

[3] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Dept. Comput. Sci., Duke Univ., Durham, NC, USA, Tech. Rep. CS-2000-06, 2000.

[4] T. K. Huang, C. K Lee, and L.-J. Chen, "PROPHET+: An adaptive prophet-based routing protocol for opportunistic network," in *Proc. 24th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Perth, Australia, Apr. 2010, pp. 112–119.

[5] S. K. Dhurandher, D. K. Sharma, I. Woungang, and S. Bhati, "HBPR: History based prediction for routing in infrastructure-less opportunistic networks," in *Proc. 27th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Barcelona, Spain, Mar. 2013, pp. 931–936.

[6] A. Lindgren, A. Doria, and D. Schelen, "Probabilistic routing in intermittently connected networks," *ACM SIGMOBILE Mobile Comp. Commun. Rev.*, vol. 7, no. 3, pp. 19–20, 2003.

[7] S. K. Dhurandher, S. J. Borah, D. K. Sharma, I. Woungang, K. Arora, and D. Agarwal, "EDR: An encounter and distance based routing protocol for opportunistic networks," in *Proc. 30th IEEE Int. Conf. Adv. Inf. Netw. Appl.*, 2016, pp. 297–302.

[8] Y. LeCun, L. Bottou, G. B. Orr, K-R. Muller, "Efficient BackProp," *Neural Networks: Tricks of the Trade*, vol. 1524, Lecture Notes Comput. Sci., pp. 9–50, Mar. 2002.

[9] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Hoboken, NJ, USA: Wiley, Oct. 2012.

[10] A. Keranen, "Opportunistic network environment simulator," Dept. Commun. Netw., Helsinki Univ. Technol., Espoo, Finland, Special Assignment Rep. ONE v. 1.4.1, May 2008.

[11] S. Drazin and M. Montag, "Decision tree analysis using Weka," Machine Learning-Project II, Univ. Miami, Miami, FL, USA, 2012. [Online]. Available: http://www.samdrazin.com/classes/een548/