Towards the Optimal Solution for the Routing Problem in Vehicular Delay Tolerant Networks: A Deep Learning Approach

R. Hernández-Jiménez, C. Cárdenas-Pérez, and D. Muñoz-Rodríguez

***Abstract*—Vehicular networks aim to provide a communication framework for moving vehicles, road infrastructure and pedestrians. Such kind of networks is the entrance to a new era of services that will make areas such as security and safety, information, transactions, entertainment and sustainability (green transportation) more efficient than they are today, especially in the upcoming era of autonomous vehicles and self-driving cars. However, the severe nature of vehicular environments makes efficient inter-vehicular communication very difficult to achieve. Vehicular Delay-Tolerant Networks (VDTN), as these networks are called, have very sparse, intermittent connections, and the lack of a fixed topology gives rise to one of the main challenges that they face: packet routing. A range of routing algorithms has been proposed in recent years to optimize communication in vehicular networks, and significant progress has been made in the matter but the quest for the optimal performance is still ongoing. In this paper, we explore a deep learning approach to the routing problem in VDTN, proposing a routing architecture and algorithm based on Deep Neural Networks that helps routers make packet forwarding decisions based on the current conditions of its surroundings. In order to assess the performance of the proposed architecture, simulations were run showing important gains in terms of network overhead and hop count with respect to popular routers, while maintaining acceptable packet delivery rates and average delivery delays.**

***Index Terms*—Vehicular Delay Tolerant Networks, Deep Learning, Neural Networks, Routing, VANETS, Packet Delivery Ratio, Average Delivery Delay.**

I. Introducción

Las redes vehiculares buscan dar capacidad de comunicación inalámbrica entre vehículos con otros vehículos, infraestructura y peatones, lo que permitiría importantes mejoras en los servicios de transporte actual, así como la creación de nuevas oportunidades [1][2]. Pero, dado que los nodos en estas redes se mueven a grandes velocidades, la conectividad y duración de los contactos es limitada, resultando en una topología no fija. Las interrupciones frecuentes en las conexiones agregan inconvenientes significativos a la red, como bajas tasas de entrega de paquetes y largos retardos. En consecuencia, uno de los más grandes retos en las comunicaciones vehículo a vehículo es el ruteo: se desea tener una alta tasa de entrega de paquetes, minimizando los tiempos de entrega y la saturación de la red. En los últimos años se han propuesto varios algoritmos oportunistas y algunas soluciones no deterministas al problema de ruteo [4][5], pero debido a las condiciones únicas de este tipo de redes, existen muchas restricciones que limitan su eficacia.

Deep Learning (*Aprendizaje Profundo*) es un subcampo de la Inteligencia Artificial que permite la modelación de relaciones complejas y está basado en algoritmos y arquitecturas que pueden "aprender" múltiples niveles de representación, de modo que abstracciones de más alto nivel terminan siendo definidas en términos de abstracciones de bajo nivel [21][19]. En general, *Deep Learning* hace referencia a redes neuronales de múltiples capas, que pueden variar desde unas pocas hasta varias miles de ellas, de modo que en cada siguiente capa se vayan teniendo diferentes niveles de abstracción de los datos, a diferencia del tradicional *Shallow Learning (Aprendizaje Superficial)*, en el que se tiene un solo nivel de abstracción (i.e., una sola capa) [21][22]. Entre otras, algunas técnicas de aprendizaje profundo son *Redes Neuronales Profundas de Propagación hacia Adelante* (también conocidas como modelo *Perceptrón Multicapa* o *Multilayer Perceptron* [23][24][25]), *Redes Neuronales Convolutivas, Redes Neuronales Recurrentes y Redes de Larga Memoria de Corto Plazo* [26][27].

En este trabajo se establece una arquitectura de ruteo y un algoritmo de enrutamiento para VDTN basada en Redes Neuronales Profundas de Propagación hacia Adelante; tanto la arquitectura como el protocolo de ruteo se explican con más detalle en la sección IV-B.

II. Trabajo Relacionado

El ruteador *Epidémico* [6] usa un principio de distribución epidémica de copias de los paquetes a los nuevos contactos que vaya encontrando en su camino. Este ruteador proporciona una tasa aceptable de entrega de paquetes y retardos aceptables también, pero a expensas de usar demasiados recursos en la red. El ruteador *Spray and Wait* [7] es un ruteador que distribuye varias copias en la red y luego espera hasta que uno de estos nodos se encuentre el destino. PRoPHET [8] es un ruteador que hace uso de encuentros probabilísticos para maximizar la posibilidad de que dos nodos se encuentren en el futuro, basándose en su historia de encuentros. Ellos introducen una métrica que llamada predictibilidad de entrega, establecida en cada nodo para cada destino conocido e indicando la probabilidad de esos nodos de entregar un mensaje a los respectivos destinos.

R. Hernández-Jiménez is with the School of Engineering and Sciences, Tecnologico de Monterrey, in Queretaro, Mx. (e-mail: rhernandj@tec.mx).

C. Cárdenas-Pérez is with the School of Engineering and Sciences, Tecnologico de Monterrey, in Guadalajara, Mx. (e-mail: ccardena@tec.mx).

D. Muñoz-Rodríguez is with the Research Division, Mexican Space Agency, in Mexico City, Mx. (e-mail: dmunoz@itesm.mx).

De este modo, cuando dos nodos se encuentran, intercambian información acerca de sus valores de predictibilidad y actualizan sus tablas acordemente, y de acuerdo a la cual se hace la decisión final de envío. SeeR [9] un ruteador basado en la técnica de "recocido simulado" (*simulated annealing*, como es conocido en inglés), que utiliza información de tiempos de inter-contacto del nodo, tiempo de vida restante del mensaje y número de saltos del paquete. Los resultados muestran ganancias importantes en la tasas y tiempos de entrega de paquetes con respecto a algoritmos que usan principios de avalancha como el enrutamiento *Epidémico* y *Spray and Wait*. KNNR, un ruteador basado en el algoritmo de clasificación de los K vecinos más cercanos (KNN, por sus siglas en inglés, que significan *K-Nearest Neighbors*), se propone en [11]. Los autores usan seis parámetros para tomar la decisión final. La clasificación utilizada se basa en la probabilidad de interacción, que es la misma que se usa en PRoPHET [8]. Sus resultados muestran una mejoría en la tasa promedio de entrega de paquetes y retardos aceptables con respecto a los ruteadores *Epidemic* y PRoPHET.

En [12], los autores exploran la posibilidad de eliminar el protocolo de enrutamiento desde una red inalámbrica utilizando técnicas de aprendizaje profundo. Sin embargo, el enunciado del problema está formulado como un problema clásico de optimización para encontrar la ruta más corta en un gráfico conectado, que es diferente de una red vehicular. Los autores en [13] proponen MLProph, un modelo basado en Aprendizaje Máquina como protocolo de ruteo, ampliando las capacidades del ruteador PRoPHET. El resultado es un ruteador mejorado con respecto a su original, aunque realiza cálculos para cada ruteador conectado, aumentando el costo computacional y comprometiendo información sensible de los nodos conectados en el intercambio de parámetros. En [14], los autores presentaron CRPO (Protocolo de Enrutamiento Cognitivo para Redes Oportunistas), que utiliza una red neuronal como núcleo, donde la decisión parámetro es la probabilidad de encuentro definida en PRoPHET. CRPO es similar en naturaleza a MLProph, ya que ambos usan la probabilidad de PRoPHET como su principal parámetro de decisión, resultando en una versión mejorada de este ruteador.

En esta sección se han descrito brevemente las principales alternativas existentes como solución al problema de ruteo en VDTN. Como puede verse, cada una de estas propuestas proporciona niveles de mejoría en algún aspecto de la comunicación en redes vehiculares, pero no constituyen una solución óptima en absolutamente todas las métricas, y en particular tienden a usar muchos recursos de la red, a diferencia del ruteador propuesto.

III. Formulación del Problema de Ruteo

Sea $N = \{N_i | 1 \leq i \leq m\}$ el conjunto de nodos disponibles en una red vehicular con interrupciones constantes y topología cambiante, y sea $A \in N$ un nodo dado en ese conjunto (fig. 1). Ya que no hay caminos predefinidos y las conexiones son intermitentes, los nodos en la red deben actuar de manera oportunista, aprovechando cualquier nodo que entre en su rango de comunicación, porque cada vez que ocurren estos encuentros, surge la oportunidad de replicar un mensaje. En esas situaciones, A tiene que escoger un nodo para iniciar una transferencia y puede utilizar diferentes criterios para tomar esta decisión, aunque invariablemente a A le gustaría elegir el nodo con mejores capacidades para difundir aún más los mensajes, hasta que lleguen a su destino.

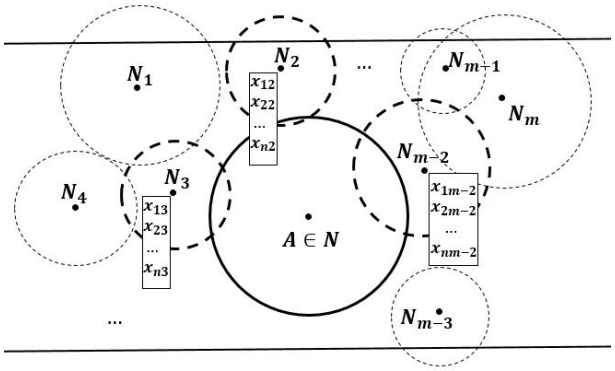

Fig. 1. Formulación del problema de ruteo en VDTN.

Bajo este enfoque, el problema de ruteo puede ser expresado como encontrar el mejor siguiente nodo para los mensajes. Esto es, de todos los k nodos a la que A está conectado en un momento dado, se tiene que determinar el que presente mejor "fitness" f, en términos de sus características actuales $x_1 \ldots, x_n$. Esto puede lograrse tratando el problema como una clasificación binaria para cada nodo, lo que nos permitiría cuantificar con precisión las capacidades de los mismos como una función F de algunas de sus características x_i de la forma $f = F(x_1, x_2, \ldots, x_n)$. Tal objetivo puede conseguirse con una red neuronal profunda de propagación hacia adelante, dado que éstas son capaces de detectar las no-linealidades del modelo.

IV. Síntesis del Ruteador Propuesto

El ruteador propuesto, al que llamaremos DLR, usa una Red Neuronal Profunda de Propagación hacia Adelante pre-entrenada como base de su arquitectura, para procesar la información de sus vecinos en tiempo real y seleccionar de ellos el mejor candidato para los mensajes.

A. Arquitectura del Ruteador

DLR tiene dos módulos fundamentales (fig. 2) que le permiten elegir el mejor siguiente nodo dentro de sus conexiones actuales, y enviarle un paquete, así como compartir información con otros nodos bajo previa solicitud.

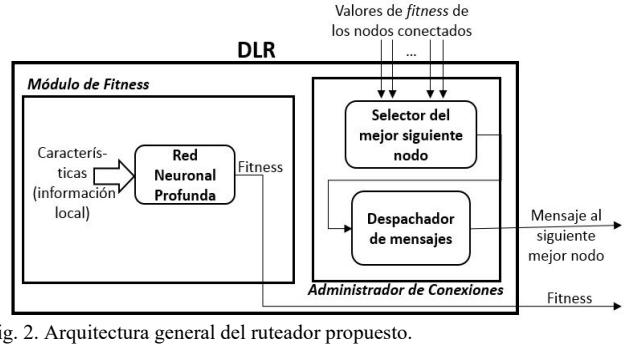

Fig. 2. Arquitectura general del ruteador propuesto.

1) El Módulo de Fitness

Este módulo tiene una red neuronal profunda pre-entrenada que utiliza información global disponible de manera local en ese momento para calcular la "salud" actual f del ruteador, con $f \in R \; y \; 0 \leq f \leq 1$, definida como el valor que determina su capacidad para entregar correctamente paquetes de datos hasta el destino final. Cuanto más cercano esté este valor a 1, más "en forma" estará el nodo (más *fit*). Este valor se actualiza automáticamente en cada ruteador justo después de que una conexión ha finalizado, por lo que está listo para ser utilizado de manera confiable por otros nodos que lo soliciten.

2) El Administrador de Conexiones

Este módulo gestiona las conexiones entrantes, solicitando sus valores f para seleccionar el nodo más *fit*. Después de esto, si está disponible, un mensaje de la cola de mensajes será enviado al nodo elegido.

B. La Red Neuronal Profunda

La red neuronal utilizada en el *Módulo de Fitness* es una Red Neuronal de Propagación hacia Adelante de $K + 2$ capas, incluyendo las capas de entrada y salida, y $K \geq 2$ capas ocultas. El énfasis detrás de *Aprendizaje Profundo,* como fue explicado en la introducción, se refiere a la capacidad de una red para poder construir patrones más complejos a partir de otros más simples, y de acuerdo con el Teorema de Aproximación Universal [18] establece que "una red neuronal con una sola capa oculta con un número finito de neuronas puede aproximar cualquier función continua en espacios compactos en R^n"; esto implica que, encontrando los parámetros apropiados, una red neuronal con una sola capa oculta es suficiente para representar una gran cantidad de problemas. Sin embargo, la amplitud de tal capa podría volverse exponencialmente grande. En efecto, Ian Goodfellow, un investigador pionero en *Aprendizaje Profundo,* sostiene que "una red neuronal con una sola capa es suficiente para representar cualquier función, pero la capa puede ser inviablemente larga y puede fallar en aprender y generalizar correctamente" [19]. No obstante, mientras que no tener capas ocultas en la red neuronal únicamente serviría para representar únicamente funciones linealmente separables, una capa oculta puede aproximar funciones con un mapeo continuo de un espacio finito a otro, y dos capas pueden representar un contorno de decisión arbitrario con cualquier exactitud [20]. En resumen, esto significa que una capa oculta ayuda a captar características no lineales de una función compleja, pero dos capas ayudan a generalizar y aprender mejor [19], particularmente auxiliados por mecanismos de regularización como *deterioro de pesos (weight decay)* o *descarte (droput)* [19].

La arquitectura general de esta red neuronal se representa en la figura 3. Aquí, $X \in R^n$ es el conjunto de n entradas $x_i, \forall i \in \{1,2, \dots, n\}$ que reflejan algunas de las características del nodo en ese momento, como su velocidad de traslado y la ocupación de su búfer; $H_i \in R^{n_{hi}}$ es el vector que contiene los valores h_i(obtenidos de acuerdo a la ecuación 3) de las n_{hi} neuronas en la capa oculta $i, \forall i \in \{1, \dots, K\}$, y f es el valor (único) resultante de *fitness* del nodo en las condiciones dadas que aparecerá en la última capa (la capa de salida, con una sola neurona). El conjunto de pesos (sinapsis) de la red neuronal está dado por $S_0 \in R^{n \times n_{h1}}$ para las conexiones entre la capa de entrada y la capa oculta 1, y $S_i \in R^{n_{hi}}$ para las conexiones de la capa oculta i hacia la capa oculta $i + 1$, para $1 \leq i \leq K$, incluyendo las conexiones de la última capa oculta hacia la capa de salida, y los valores correspondientes de sesgo o *bias* de cada sinapsis están dados por $B_i \in R^{n_{hi}}, \forall i \in [0, K]$.

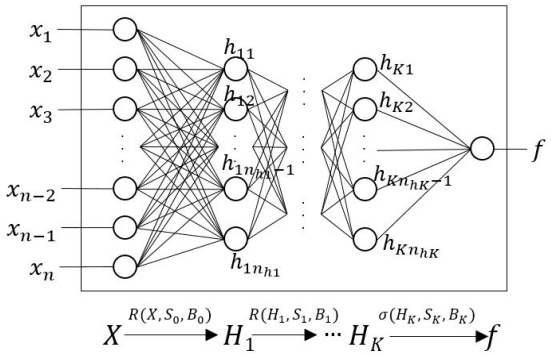

Fig. 3. Estructura de la red neuronal profunda del Módulo de Fitness.

Se usó la función *rectificador* $R(z)$ (definida en la ecuación 1; también llamada ReLU, por sus siglas en inglés, que significa *Rectified Linear Unit*) como función de activación para las neuronas de las capas ocultas, y la función *sigmoide* $\sigma(z)$ (definida en la ecuación 2) como función de activación para la neurona en la capa de salida, ya que queremos que este valor refleje el estado de *fitness* de los ruteadores, y la naturaleza de esta hace que se tengan valores que están entre 0 y 1. De este modo, el valor de final de "salud" para el nodo se calcula tomando el conjunto actual de características X del nodo y haciendo una propagación hacia adelante a través de la red neuronal, como se muestra matemáticamente en la ecuaciones 3 y 4, donde $P \cdot Q$ denota el producto punto (producto escalar) entre P y Q.

$$R(z) = \begin{cases} 0, & z \leq 0 \\ z, & z > 0 \end{cases} \tag{1}$$

$$\sigma \; = \frac{1}{1 + e^{-z}} \tag{2}$$

$$H_i = R(H_{i-1} \cdot S_{i-1} + B_{i-1}), \forall i \in \{1, \dots, K\} \tag{3}$$

$$f \; = \sigma \; (H_K \cdot \; S_K + B_K) \tag{4}$$

Dada la naturaleza matemática de la función sigmoide, entre más cerca esté f a 1, más *saludable* será, y viceversa.

V. El Algoritmo de Ruteo

Para tener cierta sensibilidad ante la "salud" de otros nodos, DLR incluye el parámetro α, con $0 \; \leq \; \alpha \; \leq \; 1$, bautizado como el *umbral de "salud"* o *fitness*, y que determina el valor mínimo (límite) de *fitness* en que conexiones actuales pueden ser directamente ignoradas por sus ruteadores vecinos. Este valor es un componente clave en el protocolo de ruteo en DLR, ya que diferentes valores de umbral dan como resultado diferentes dinámicas en el entorno oportunista. El algoritmo 1 resume el protocolo de enrutamiento, explicado en los apartados A y B de esta sección.

Algoritmo 1: Algoritmo de DLR. Acciones del nodo c conectado a un conjunto C de nodos.

Evento de Fin de Conexión – Actualización de *fitness*
Entradas:
X: el conjunto de características x_i de c
Salidas:
f: el valor actualizado de *fitness* de c
Pasos:
1. Obtener las características actuales x_i de c y normalizarlas de acuerdo con la ecuación 5.
2. Calcular el valor f de c de acuerdo con las ecuaciones 3 y 4.

Evento de Nueva Conexión – Selección del mejor siguiente nodo
Entradas:
C: el conjunto de nodos conectados a c en ese momento
M: el conjunto de mensajes en la cola de c
Salidas:
F: el conjunto de tuplas de conexión ordenadas por fitness
Pasos:
1. Intercambiar mensajes cuyo destino final esté en C
2. **Hacer**:
   Para cada $c_i \in C$**:**
   conseguir f_i
   si $f_i \geq \alpha$**:**
   almacenar tupla (c_i, f_i) en el vector F
3. **Ordenar** F en orden descendiente
4. Replicar los mensajes en M a las conexiones en F

A. Actualización del Valor f

Esta primera etapa tiene lugar cada vez que una conexión entre el host y otro nodo de la red (vehicular) ha terminado. Debido a que algunas de sus características pueden haber cambiado (como la ocupación del búfer, tasa de descarte de paquetes o algunos otros), su valor *fitness* también debe ser recalculado. Para ello, primero, en el *Módulo de Fitness*, las características consideradas x_i son obtenidas y pasadas por un proceso de normalización para obtener características normalizadas x_i' de acuerdo con la ecuación 5, donde x es una característica que está siendo transformada, y x_M y x_m son los valores respectivos máximo y mínimo registrados de esa característica.

$$x' = \frac{x - x_m}{x_M - x_m} \tag{5}$$

Esto dará como resultado valores de entrada finales x_i' con $0 \leq x_i' \leq 1$, lo que hará el proceso de predicción más confiable. Estos valores normalizados se pasan a través de la red, de acuerdo con las ecuaciones 3 y 4, para dar el valor final (actualizado) f.

B. Selección del Mejor Siguiente Nodo y Reenvío de Paquetes

La segunda parte del proceso de ruteo ocurre cuando se establece un enlace entre el nodo actual y alguno de sus nodos vecinos. En este momento, el ruteador intentará intercambiar mensajes cuyo destino final se encuentra entre esas conexiones, si es que los hubiera. Luego, el ruteador anfitrión solicita a cada uno de los nodos conectados su valor de *fitness* (que, gracias a sus módulos de *acondicionamiento físico*, estarán listos y actualizados). Después, antes de entrar en la selección final, el ruteador descarta aquellas conexiones cuyo valor f no es al menos el *umbral de fitness* (α), y ordena en orden descendente las conexiones restantes. Con una lista completa de candidatos aptos, el proceso de selección es sencillo: el mejor próximo nodo será el que esté más "saludable" (el que tenga el valor f más alto), por lo que el ruteador intentará replicar un paquete de datos a los nodos en ese orden.

VI. Experimento

En esta sección se describe el diseño y la ejecución del experimento para validar la arquitectura propuesta.

A. Configuración de la Simulación

Se usó el simulador ONE (*Opportunistic Network Environment*), que es un entorno virtual diseñado para evaluar redes oportunistas [15].

Fig. 4. Calles y carreteras usadas en la simulación.

El escenario de prueba es una parte de la ciudad de Querétaro, capital del Estado con mismo nombre, un estado de México de tamaño mediano, con poco más de 2 millones de habitantes. El escenario fue delimitado por un terreno cuadrado de 1000m por 1200m con las calles y carreteras de la ciudad (fig. 4) y el tiempo de simulación fue de 43,200 segundos (12 horas).

1) Modelo de Movilidad

Esta característica ayuda a que la simulación sea más apegada a la realidad, proporcionando coordenadas, velocidades y tiempos de pausa para los nodos. Modelos populares incluyen *Random Waypoint* (*Puntos de Referencia Aleatorios*), *Map-based Movement* (Movimiento basado en Mapas) y S*hortest Path Map-based Movement* (*Ruta más corta con Movimiento basado en Mapas*) [3]. Utilizamos el último para la simulación, pues restringe el movimiento del nodo a rutas predefinidas, y utiliza el algoritmo de Dijkstra para encontrar la ruta más corta entre dos puntos en el mapa. Con este modelo, ya que un nodo ha alcanzado su destino, hace una pequeña pausa, y luego se elige otro nodo aleatorio del mapa como destino, repitiendo el proceso.

2) Grupos de Nodos

En esta simulación hubo un total de 85 nodos (tabla 1), divididos en 8 grupos diferentes, cada uno con características particulares. Se incluyeron conexiones a 6Mbps, 12 Mbps y 24 Mbps, que están dentro del rango de velocidad de transmisión especificado por el estándar IEEE 802.11p del conjunto de estándares de *Acceso Inalámbrico para Entornos Vehiculares* (WAVE, por sus siglas en inglés) [16]. Además, se incluyeron conexiones Bluetooth a 2Mbps. El tamaño del búfer, la velocidad máxima de los nodos y el número de nodos de cada

tipo se muestran en la tabla 1. El TTL de los mensajes se iteró de 15 a 600 minutos en incrementos de 15 minutos.

TABLA I
GRUPOS DE NODOS EN LA ESCENARIO VEHICULAR PLANTEADO

Gp.	Nodos	ID	Búfer (MB)	Velocidad (m/S)	Interfase	Descripción
1	10	p1	5	0.5 – 1.5	Bluetooth	Un grupo de peatones
2	10	p2	5	0.5 – 1.5	WAVE 802.11p @6Mbps	Otro grupo de peatones
3	4	b1	10	2.7 – 16.7	WAVE 802.11p @6Mbps	Un grupo de autobuses
4	10	b2	10	2.7 – 16.7	WAVE 802.11p @12Mbps	Otro grupo de autobuses
5	15	c1	15	5.5 – 22.22	WAVE 802.11p @12Mbps	Un grupo de carros de baja velocidad
6	15	c2	15	5.5 – 22.22	WAVE 802.11p @24Mbps	Otro grupo de carros de baja velocidad
7	10	c3	20	8.3 – 30.56	WAVE 802.11p @12Mbps	Un grupo de carros de alta velocidad
8	10	c4	20	8.3 – 30.56	WAVE 802.11p @24Mbps	Otro grupo de carros de alta velocidad

3) Ruteadores

La simulación principal se realizó con el ruteador propuesto (DLR), y se hicieron pruebas contra cuatro populares protocolos de ruteo: el ruteador *Epidémico*, el ruteador *Spray and Wait*, el ruteador *PRoPHET* y el ruteador *SeeR*.

B. La Red Neuronal en DLR

1) Estructura

Explicado a más detalle en la sección IV-B, una red neuronal con una sola capa oculta ayuda a captar aspectos no lineales de un sistema, en un primer nivel, pero dos capas es suficiente para generalizar mejor construyendo abstracciones a partir del nivel anterior; por eso, para este experimento se consideró $K = 2$ capas ocultas además de las capas de entrada y salida. En esa misma sección se presentó la arquitectura de la red con $K + 2$ capas, $K \geq 2$.

El número de neuronas en la capa de entrada en esta red es el número n de características a procesar de cada muestra en el proceso de clasificación. Considerar características de varios aspectos de los nodos (vehículos en la red) es importante para tener un panorama más completo del comportamiento de los mismos en el escenario planteado. Por eso, en esta versión se han incluido características físicas, de comunicación y de recursos. En total se consideraron $n = 10$ características x_i diferentes, enumeradas en la tabla 2.

En cuanto al número n_{hi} de neuronas en cada capa oculta H_i , no existe una fórmula mágica para el número óptimo, aunque se pueden usar algunas reglas empíricas [17]. Lo más común es que el tamaño óptimo de las capas ocultas esté generalmente entre el tamaño de la capa de entrada y el tamaño de la capa de salida. Esto significaría que $n = 10 \geq n_{hi} \geq 1$. Otra sugerencia es mantener este número como la media de las neuronas en las capas de entrada y salida, para la primera capa oculta, y a partir de aquí ir decreciendo paulatinamente el número de neuronas en cada capa oculta subsecuente, sin exceder 2 neuronas en la última capa, para capturar mejor las no-linealidades de la última abstracción en los datos, ya que de lo contrario esa capa oculta con una sola neurona se convertiría en la capa de salida [17].

TABLA II
CARACTERÍSTICAS DE LOS HOSTS CONSIDERADAS EN DLR

Característica	Nombre	Descripción
x_1	Velocidad del host	Velocidad (m/S) a la que el vehículo se está moviendo
x_2	Velocidad de transmisión	Velocidad de transmisión del link de comunicación (Mbps)
x_3	Rango de transmisión	Máxima distancia radial (m) a la que este nodo puede conectarse con otros nodos
x_4	Número promedio de conexiones	Número promedio de conexiones que este nodo es capaz de mantener
x_5	Tamaño del búfer	Tamaño del búfer, en MB
x_6	Ocupación del búfer	Porcentaje de ocupación del búfer
x_7	Número de mensajes finales	Número de mensajes que este host ha entregado a destino final
x_8	Tasa de entrega de mensajes finales	Tasa de entrega de mensajes finales por este host
x_9	Tasa de descarte	Tasa a la que este host descarta los mensajes
x_{10}	Tasa de abortos	Tasa de aborto de mensajes por este host

Esto implicaría que $n_{h1} = 5$ y $n_{h2} \leq 5$. Una última sugerencia para evitar *sobreajuste* durante el proceso de entrenamiento (lo que significaría que la red neuronal tendría grandes capacidades de memoria, pero no de predicción sobre datos no vistos durante su entrenamiento) es mantener el número de neuronas en las capas ocultas como en la ecuación 6, donde N_s es el número de muestras en el conjunto de entrenamiento, N_i es el número de neuronas en la capa de entrada, N_o es el número de neuronas en la capa de salida, y β es un factor de escala arbitrario, generalmente con $2 \leq \beta \leq 10$.

$$N_{hi} < \frac{N_s}{\beta\,(N_i + N_o)} \tag{6}$$

En este contexto, esto significaría:

$$n_{hi} < \frac{N_s}{\beta\,(10 + 1)} = \frac{N_S}{11\beta} \tag{7}$$

Siguiendo estas tres sugerencias y buscando un menor tiempo computacional, se decidió $n_{h1} = 5$ y $n_{h2} = 3$.

Finalmente, la capa de salida tiene una sola neurona, que, según la ecuación 2, tendrá un valor entre 0 y 1. Durante la etapa de entrenamiento, este valor se convierte en un valor digital, por lo que cada muestra (de entrenamiento) tiene una etiqueta única $l \in \{0,1\}$, dado por:

$$l = round\left(\frac{f + 0.5}{2}\right) \tag{8}$$

Este proceso de etiquetado sirve para comparar y evaluar la predicción de clase durante el entrenamiento. Sin embargo, hay que tener en cuenta que durante la aplicación de la red neuronal en el entorno VDTN este paso de etiquetado no se realiza, ya que sólo nos interesa seleccionar la muestra con el mejor estado de *"*salud*"* (es decir, la muestra con el valor f más alto), que es dado directamente después de la propagación hacia adelante (ecuaciones 3 y 4).

2) Entrenamiento

DLR utiliza $K+1$ matrices de sinapsis S_i, y sus correspondientes vectores de bias B_i, con $i \in \{0, \dots, K\}$, y en donde K es el número de capas ocultas de la red neuronal profunda, tal como se introdujo por primera vez en la sección IV-B. Estas matrices se determinan durante la etapa de entrenamiento con un conjunto de datos de muestras recopiladas previamente de un escenario de simulación con las condiciones definidas en la sección VI-A. Más particularmente, los hosts se configuraron con los tres ruteadores populares *PRoPHET*, *Spray and Wait* y *SeeR*, y un total de 11,016,000 vectores $X = [x_0, x_1, \dots, x_{10}]$ de muestras se obtuvieron de una simulación con tiempo de simulación de 43200 segundos (12 horas), reuniendo las características actuales x_i de cada uno de los 85 hosts cada segundo. Las etiquetas l para cada muestra se obtuvieron directamente de la característica *final delivery rate* (FDR, característica x_8), considerando que cuantos más mensajes entregue un host a un destino final, más *fit* tiene que ser. Para esto, las muestras se pasaron a través de un proceso de estandarización y luego las que tuvieron un puntaje z no negativo se consideraron como "saludables" ($l = 1$) de acuerdo con la ecuación 9, donde x es el valor de la característica x_8, $\bar{x}$ es la media de todos esos valores x_8 en el conjunto de datos, y σ es la desviación estándar muestral.

$$l = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}, \text{ con } z = \frac{x - \bar{x}}{\sigma} \tag{9}$$

En preprocesamiento, los registros duplicados se eliminaron del conjunto original de datos, y todos los valores restantes se normalizaron para cada característica x_i, según la ecuación 5, para tener un mejor mapeo y una convergencia más rápida durante el entrenamiento, y el conjunto de datos final fue permutado al azar. El conjunto de datos resultante se dividió en dos subconjuntos para entrenamiento real (80% de los datos) y validación (20%), para evaluar el proceso de aprendizaje y generalización. Otros hiperparámetros de la red fueron: optimizador *Adam* (más rápido que el tradicional *descenso de gradiente estocástico,* [10]), *entropía cruzada binaria* como función de error, y una tasa de descarte del 20% para promover la generalización (reducir sobreajuste). Así, se logró 96.08% de precisión en el grupo de entrenamiento y 99.90% en el conjunto de validación. Así se obtuvieron las matrices de sinapsis S_i y bias B_i, que fueron embebidos en DLR.

C. Umbral de Fitness α

Se encontró que $\alpha = 0.1$ ofrecía el mejor rendimiento (sección VII-B), por lo que ese es el valor predeterminado sugerido para este parámetro en DLR.

D. Métricas de Evaluación

Se consideraron las siguientes métricas para evaluar el rendimiento de DLR:

1) Tasa de Entrega de Paquetes (PDR)

Llamaremos a esta métrica PDR, para abreviar (del inglés *Packet Delivery Ratio*, que es un término conocido en el argot de VDTN). Este valor muestra la tasa de mensajes creados que llegaron a su destino (ecuación 10).

$$PDR = \frac{\#\ de\ mensajes\ entregados}{\#\ de\ mensajes\ creados} \tag{10}$$

Lo ideal es que este número fuera 1, pero en la práctica esto parece más bien imposible, ya que existen otras restricciones en la red, como el tamaño del búfer y tiempo de vida de los mensajes (TTL), generando políticas de destrucción o descarte que impiden que algunos mensajes lleguen a su destino.

2) Retardo Promedio de Entrega (ADD)

También conocido como latencia, este parámetro es el tiempo transcurrido desde que se crea un mensaje hasta que llegue a su destino. En otras palabras, este número muestra cuánto tiempo transcurre para que se entregue un mensaje. Idealmente, nos gustaría que este valor sea 0, pero esto es obviamente imposible; en su lugar, se busca la minimización de este parámetro. Llamaremos a este parámetro ADD, para abreviar (*Average Delivery Delay*).

3) Sobrecarga de la Red (OVH)

Este parámetro (OVH, para abreviar; del inglés *Overhead*) muestra la proporción de los mensajes que se transmitieron a la red y que no llegaron a su destino con respecto a la cantidad de mensajes que sí lo hicieron (ecuación 11).

$$OVH = \frac{\#\ msjs\ transmitidos - \#\ msjs\ entregados}{\#msjs\ entregados} \tag{11}$$

El impacto de OVH se observa directamente en el uso de recursos en toda la red. Idealmente, este valor debería minimizarse para reducir los problemas relacionados con mala asignación de ancho de banda, como congestiones de red y demoras consecuentes e interrupciones innecesarias.

4) Cuenta de Saltos (HOP)

Abreviado HOP (del inglés *hop*, que significa *salto*), este parámetro muestra el número de nodos que un mensaje atravesó para llegar a su destino final. Entre más pequeño es este parámetro, menos sobrecarga administrativa en los nodos anteriores se generó a causa de este mensaje. Por lo tanto, lo ideal también es mantener este valor bajo.

VII. Resultados

A. Efecto del Tiempo de Vida de los Mensajes (TTL)

Como se puede ver en las gráficas que siguen, el tiempo de vida de los mensajes (TTL) tiene un impacto significativo en las métricas hasta cierto punto, ya que cuanto más largo sea el tiempo de existencia de un mensaje, mayor es la probabilidad de que sea entregado (aunque a expensas de más overhead y ocupación de búfer).

Cualquier métrica, sin embargo, tiende a estabilizarse a medida que se otorga más TTL. El valor de estabilización para este escenario es de alrededor de 150 segundos. Esto significa que agregar más tiempo de vida a los mensajes normalmente no agregará ninguna mejora. Además, dependiendo del ruteador, algunos de ellos exhibirán un mejor rendimiento cuando el TTL es más pequeño que el del punto de asentamiento, como vemos, por ejemplo, en los ruteadores *PRoPHET* y *Epidémico* (fig. 9). Así, recomendamos al menos un máximo TTL = 150s al evaluar el rendimiento del ruteador para capturar el comportamiento completo.

B. Efecto del Umbral de Fitness α

Este parámetro determina hasta qué punto algunas de las conexiones se descartan como posibles candidatos (ver sección V). Intuitivamente, un valor muy pequeño significaría que la mayoría de las conexiones tienen la oportunidad de ser elegidas, y viceversa. Para comprender mejor el efecto de este umbral, se hizo un barrido de $\alpha = 0$ a $\alpha = 1$ en incrementos de 0.05 y un TTL de 0 a 600 minutos en incrementos de 15 minutos. Los resultados mostraron que se tenía el mayor rendimiento en PDR con $\alpha = 0.1$, a expensas de un poco de retardo adicional en la entrega de paquetes, y el mejor rendimiento en ADD con $\alpha = 0$, a expensas de menos tasa de entrega de paquetes. Ninguno de los otros valores de α ofrecía ventajas visibles respecto a estos dos valores clave de umbral. Ambos ruteadores ($\alpha = 0$ y $\alpha = 0.1$) son incluidos para comparación con ruteadores de otros tipos en las figuras 5 a 8.

C. Rendimiento de DLR.

Como se puede ver en la fig. 5, DLR ($\alpha = 0.1$) ofrece un PDR mejor que los ruteadores PRoPHET y Epidemic, para TTL grandes, y su rendimiento es aceptable ya que está muy cerca de aquellos que ofrecen los mejores valores, solo 10.7% por debajo de sus mejores contrapartes y alrededor de 17.45% y 7.2% por encima de los ruteadores que tuvieron un rendimiento inferior. Adicionalmente, con $\alpha = 0$, se tuvo el PDR más bajo, pero relativamente aceptable también, ya que está en alrededor de la mitad en rendimiento que los otros ruteadores.

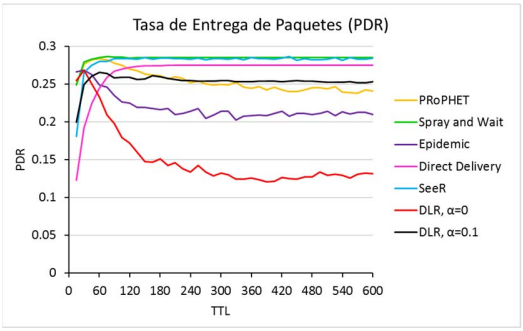

Fig. 5. Tasa de Entrega de Paquetes (PDR).

En contraste, en tiempo de entrega (ADD), DLR ($\alpha = 0.1$) presentó mejor promedio que el ruteador *Direct Delivery*, lo que resultó en un tiempo de entrega aproximadamente 23% mayor que el ruteador propuesto, que presentó aproximadamente el 34.5% más latencia que los ruteadores *Epidemic* y *PRoPHET*, y 63.2% más de retraso que *Spray and Wait* y *SeeR*, que son definitivamente los dos mejores ruteadores cuando se trata de tiempos de entrega (fig. 6).

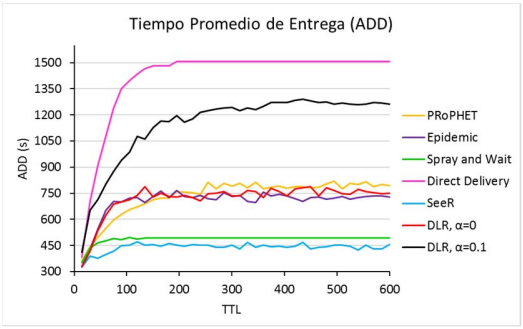

Fig. 6. Tiempo Promedio de Entrega (ADD).

Para esta misma métrica, sin embargo, DLR ($\alpha = 0$) tuvo mejores resultados, al nivel de los ruteadores *PRoPHET* y *Epidemic*, a solo 38% sobre *SeeR*, que exhibió el mejor rendimiento. Por otro lado, en la sobrecarga de la red (OVH, fig. 7), DLR ($\alpha = 0$) obtuvo mejores valores, superando en gran medida el rendimiento de los ruteadores *Epidemic*, *PRoPHET* y *SeeR*, que presentan más OVH que el ruteador propuesto con aproximadamente 6.7, 3.4 y 0.9 veces más.

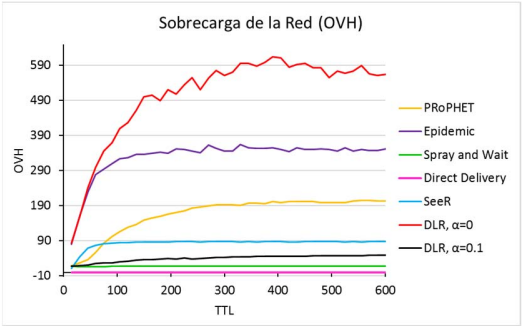

Fig.7. Sobrecarga de la Red (OVH).

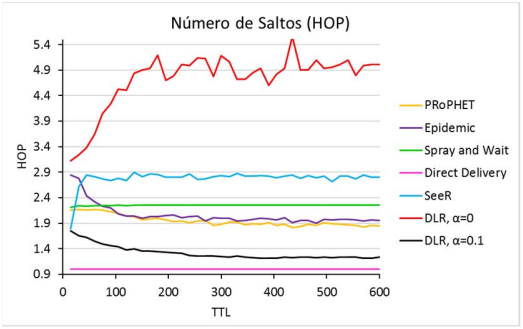

Fig.8. Número de saltos intermedios (HOP).

En la Fig. 9 se muestra el resumen de mejorías de cada ruteador, en donde ✓ significa que el ruteador tuvo mejorías en la métrica y con respecto al ruteador correspondientes. Se puede ver que ningún ruteador es mejor que todos en ninguna de las métricas, a diferencia de *Seer*, que es el mejor en cuanto a retardos. Esto significa que debe haber un intercambio entre las métricas en un ruteador dado. Normalmente, se puede tener una mejor tasa de entrega de paquetes y/o una mejoría en los retados a expensas de sobrecarga de red y número de saltos.

Con esto, DLR puede utilizarse en situaciones en donde los recursos sean limitados o la red sufra de congestiones muy grandes, ya que este ruteador presenta mejorías en cuanto a sobrecarga de red y número de saltos, y presenta mejores tiempos de entrega que *Direct Delivery*, con una tasa de entrega de paquetes muy cercana a la óptima. En contraste, si los recursos y sobrecarga de la red no son una limitante, otros ruteadores podrían preferirse a esta versión de DLR, dependiendo de la métrica que se desee optimizar; por ejemplo, *Spray and Wait*, para maximizar la tasa de entrega de paquetes, o *Seer*, para minimizar los retardos.

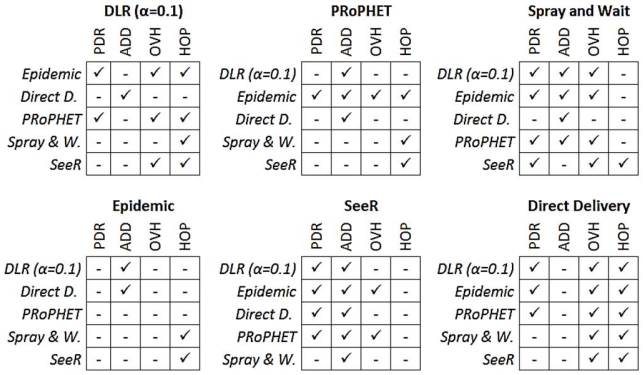

Fig. 9. Resumen de mejorías de cada ruteador.

VIII. Conclusiones y Trabajo Futuro

Hemos presentado una nueva arquitectura de ruteo para VDTN basada en *Deep Learning*, cuyo algoritmo aprovecha el poder de las Redes Neuronales Profundas de Propagación hacia Adelante para aprender de su entorno y usar esa información para tomar decisiones de reenvío de paquetes. El ruteador propuesto presenta un rendimiento aceptable en comparación con ruteadores populares y se ha explicado a detalle en la sección VI. Direcciones de trabajo a partir de aquí incluyen la incorporación de la red neuronal para trabajar en tiempo real usando técnicas semi-supervisadas y no supervisadas, así como la combinación de los enfoques de *selección del mejor siguiente nodo* y la *selección del mejor siguiente mensaje* para tener una perspectiva más completa del entorno de comunicación.

Referencias

[1] Hernández, R., Cárdenas, C. and Muñoz, D. (2014). On the Importance of Delay-Tolerant Networks for Intelligent Transportation Systems in Smart Cities. Mexican International Conference on Computer Science (ENC 2014), Nov 3-5, 2014; Oaxaca, México.

[2] Hernández, R., Cárdenas, C. and Muñoz, D. (2015). Epidemic Routing in Vehicular Delay-Tolerant Networks: the use of Heterogeneous Conditions to Increase Packet Delivery Ratio. IEEE International Smart Cities Conference, 2015.

[3] Shahzamal, M., Pervez, M. F., Zaman, M. A. U. and Hossain, M. D. (2014). Mobility Models for Delay Tolerant Networks: A Survey. International Journal of Wireless and Mobile Networks. Vol. 6, No. 4.

[4] Dua A., Kumar N. and BawaA S. (2014). Systematic review on routing protocols for Vehicular Ad Hoc Networks. Vehicular Communications, Volume 1, Issue 1, January 2014, pp 33-52

[5] Asgari M., Jumari K. and Ismail M. (2011). Analysis of Routing Protocols in Vehicular Ad Hoc Network Applications. In: Software Engineering and Computer Systems. Communications in Computer and Information Science, 2011. Vol 181. Springer, Berlin, Heidelberg

[6] Vahdat, A. and Becker, D. (2000). Epidemic Routing for Partially-Connected Ad Hoc Networks. In: Handbook of Systemic Autoimmune Diseases. Technical Report.

[7] Spyropoulos, T., Psounis, K. and Raghavendra, C.S. (2005). Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In: ACM SIGCOMM'05 Workshops, 2005.

[8] Lindgren A., Doria A. and Schelén O. (2004). Probabilistic Routing in Intermittently Connected Networks. In: Service Assurance with Partial and Intermittent Resources. SAPIR 2004. Lecture Notes in Computer Science, vol 3126. Springer, Berlin, Heidelberg

[9] Saha, B. K., Misra, S. and Pal, S. (2017). SeeR: Simulated Annealing-based Routing in Opportunistic Mobile Networks. IEEE Transactions on Mobile Computing; vol. 16, no. 10, pp. 2876{2888, Oct. 2017.

[10] Kingma, D. P., and Ba, J. (2014). Adam: A Method for Stochastic Optimization. Proceedings of the 3rd International Conference on Learning Representations (ICLR), 2014.

[11] Sharma, D.K., Aayush.,Sharma, A. and Kumar, J. (2017). KNNR: K-nearest neighbor classification-based routing protocol for opportunistic networks. 10th International Conference on Contemporary Computing (IC3), 10-12 August 2017, Noida, India.

[12] Tang, F., et al. (2018). On Removing Routing Protocol from Future Wireless Networks: A Real-time Deep Learning Approach for Intelligent Traffic Control. IEEE Wireless Communications; vol. 25, no. 1.

[13] Sharma, D. K., Dhurandher, S. K., Woungang, I., Srivastava, R. K., Mohananey, A., and Rodrigues, J. J. P. C. (2018). A Machine Learning-Based Protocol for Efficient Routing in Opportunistic Networks. In: IEEE Systems Journal; vol. 2, no. 3, pp. 2207-2213. Sep. 2018.

[14] Gupta, A., Bansal, A., Naryani, D. and Sharma, D.K. (2017). CRPO: Cognitive Routing Protocol for Opportunistic Networks. Proceedings of the International Conference on High Performance Compilation, Computing and Communications; vol. 1, pp. 121-125, March 2017.

[15] Keränen, A., Ott, J. and Kärkkäinen, T. (2009). The ONE Simulator for DTN Protocol Evaluation. 2nd International Conference on Simulation Tools and Techniques; SIMUTools 2009, Rome, Italy.

[16] Wang, Y. Duana, X., Tiana, D., Lu, G. and Yu, H. (2013). Throughput and Delay Limits of 802.11p and Its Influence on Highway Capacity. In: Intelligent and Integrated Sustainable Multimodal Transportation Systems. Special Issue. Procedia – Social and Behavioral Sciences; vol 96, pp. 2096-2104, Nov. 2013.

[17] Hyndman, R. (2017). How to choose the number of hidden layers and nodes in a feedforward neural network? Retrieved on June, 2019, from https://stats.stackexchange.com/q/181

[18] Balázs Csanád Csáji. (2001). Approximation with Artificial Neural Networks; Faculty of Sciences; Eötvös Loránd University, Hungary.

[19] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press. http://www.deeplearningbook.org

[20] Heaton, J. (2008). Introduction to Neural Networks for Java. Heaton Research, Inc.

[21] Deng, L., and Yu, D. (2014). Deep Learning: Methods and Applications. Foundations and Trends in Signal Processing. Vol. 7, Issues3-4.

[22] Sharma, O. (2019). Deep Challenges Associated with Deep Learning. 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (Com-IT-Con), India.

[23] Torfi, A. (Sup.) (2019). Multi-layer Perceptron. Machine Learning Course. Deep Learning. https://machine-learning-course.readthedocs.io/en/latest/content/deep_learning/mlp.html

[24] Zhang, A. Lipton, Z., Li, M., and Smola, A. J. (2019). Dive Into Deep Learning. https://d2l.ai/index.html

[25] LeDun, Y., Bengio, Y and Hinton, G. (2015). Deep Learning. Nature. 521, 436–444

[26] Ayinde, B. O., Inac, T. and Zurada, J. M. (2019). Regularizing Deep Neural Networks by Enhancing Diversity in Feature Extraction. IEEE Transactions on Neural Networks and Learning Systems. VOL. 30, No. 9. Sep. 2019.

[27] Wan, H. (2019). Deep Learning: Neural Network, Optimizing Method and Libraries Review. 2019 International Conference on Robots & Intelligent System.

Roberto Hernández-Jiménez received his B.S. degree in Electronics in 2010, from the Tecnologico de Monterrey. He is currently pursuing his PhD degree in Engineering Sciences, while being a part-time 2036rofesor at same Institute, in Mexico. His research interests include the application of Artificial Intelligence and Deep Learning to optimization problems in engineering and social sciences.

César Cárdenas-Pérez received in 1991 his B.S. degree in Communications and Electronics from the Tecnologico de Monterrey, Mexico (honors), his M. S. in Satellite Communications (1995) and his PhD in Networks and Computer Science (2010, Trés Honorable) both from Télékom ParisTekh, France. He is member of the Mexican Association of Computing, Mexican Society of Computer Science, IEEE, and founder member of the Mexican Association of Embedded Software. His research interests include computer communications and embedded systems.

David Muñoz-Rodríguez received the B.S., M.S., and Ph.D. degrees in electrical engineering from the Universidad de Guadalajara, México, Cinvestav México City, and University of Essex, Colchester, England, in 1972, 1976, and 1979, respectively. He was formerly Chairman of the Communication Department and Elektrical Engineering Department at CINVESTAV, and director of Center for Electronics and Telecommunications in Tek de Monterrey, Monterrey, where he became emeritus professor. He is member of the Mexican Academy of Sciences, Engineering Academy, and Senior Member of the IEEE Communications Society. His research interests include transmission and personal communication systems.