**Title**
A Network Layer for Intelligent Vehicle Highway Systems



**Authors**
Eskafi, Farokh
Zandonadi, Marco

**Publication Date**
1999-07-01

CALIFORNIA PATH PROGRAM
INSTITUTE OF TRANSPORTATION STUDIES
UNIVERSITY OF CALIFORNIA, BERKELEY

# A Network Layer for Intelligent Vehicle Highway Systems

**Farokh Eskafi**
**Marco Zandonadi**

**California PATH Working Paper**
**UCB-ITS-PWP-99-11**

This work was performed as part of the California PATH Program of the University of California, in cooperation with the State of California Business, Transportation, and Housing Agency, Department of Transportation; and the United States Department Transportation, Federal Highway Administration.



Report for MOU 334

July 1999

ISSN 1055-1417

# A NETWORK LAYER FOR INTELLIGENT VEHICLE HIGHWAY SYSTEMS

MOU 334 Interim Report

Farokh Eskafi and Marco Zandonadi
E-mail: farokh@path.berkeley.edu, marcoz@path.berkeley.edu

California PATH - U.C. Berkeley
January 1999

## ABSTRACT

The objective of this paper is to design the network layer of a communication stack to be used in Automated Highway Systems (AHS). The communication model we propose allows cars to form private subnets, the configuration of which can change dynamically. Each car be part of multiple subnets and can send broadcast, multicast and point-to-point messages to other vehicles (both on the same subnet and on others, through a routing mechanism). Each subnet is managed by a server: a car that is in charge of accepting/rejecting join requests and of keeping a consistent state within the subnet. Finally we propose an AHS addressing scheme and an API implementing the discussed functionality.

## INTRODUCTION

In recent years intelligent vehicle systems have been the focus of considerable research. The automotive industry and government have launched numerous initiatives to put commercial and passenger vehicles on the information superhighway. We are interested in the development of a flexible and scalable communication architecture that structures the networking intrinsic to this cybernetic revolution. Desires for connectivity on the road range all the way from near-term "Net Vehicle" concepts that promise internetworking on the road, intelligent driving assistants designed to enhance the safety and comfort of driving through warnings, directions, or partial control, to the distant though inevitable reality of fully automated road travel.

Some of these future networked applications will be safety critical and others will enhance comfort and convenience. They will demand different qualities of service. Cars move rapidly with respect to each other and the roadside. Consequently the topology of the network is highly dynamic. In this paper we are primarily concerned with the design of the software that maintains connectivity in this rapidly reconfiguring environment, i.e., the network layer of our communication architecture.

It should be noted that the software is designed to support safety critical applications encompassing spatially distributed vehicle groups, in addition to applications with less stringent QOS requirements.

The communication system assumes the presence of roadside entities and it supports both vehicle-to-vehicle and vehicle-to-roadside communication. Automated vehicles have *n* communication devices each of which can implement a different technology (IR, MPI, Wavelan). All vehicles in a road section are connected to each other through a *common channel (CC)*, whose frequency is known and fixed. The CC is a broadcast channel and vehicles can use it to send one way messages. All roadside entities have the same known fixed address so that every vehicle is able to communicate with them a priori (roadside entities have to be distant enough from each other to avoid two of them receiving the same message).

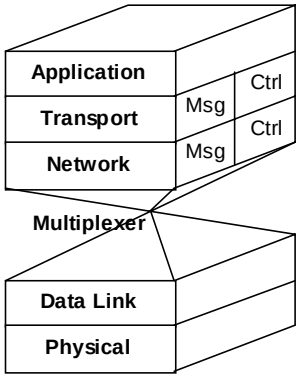
Vehicles can also setup private *subnets* among each other. The CC can be used to exchange setup information when creating the subnet.

### *Communication Stack*

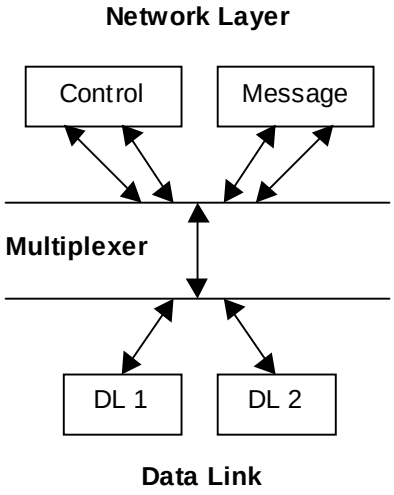
The model for the communication stack is an hybrid between the OSI model and TCP/IP (it is similar to the one adopted in (Tanenbaum 1996)). It is composed of six layers: *physical*, *data link*, *(de)multiplexing*, *network*, *transport* and *application*. The physical layer is the medium through which bits are actually transferred. The data link layer is a set of as many components as available communication devices; it's main function is to enhance the reliability of the medium. The network layer is

in charge of setting up subnets among vehicles and of queuing and routing messages. The transport layer performs message splitting, handles *Quality of Service (QOS)* and provides a way to create multiple transport addresses on a host.

Transport and Network layer are composed of two parts: *control* and *message*. The message part deals with user messages coming from the upper layers: it does not analyze the content of the message, it just passes them to the upper or lower layer, after adding or removing a header. The control part deals with configuration of the communication software.




The (de)multiplexing layer is needed to direct messages coming from a data link layer component to the message or control part of the network layer and viceversa.




The control component of the network layer contains a subnet manager to handle subnet operation and a routing manager to allow messages to be sent through different subnets. The control component of the transport layer contains a QOS manager to ensure that a user-specified QOS is provided when transmitting data.

## NETWORK LAYER

Cars in the highway may do the following things:

- use the CC;
- create subnets;
- communicate with the roadside and with other cars inside and outside the subnet.

The network layer provides the means by which such functionality is made available. Each subnet is managed by a car, called *server*. Other cars in the subnet are called *clients.* Subnets are connected to each other through vehicles that belong to more than one subnet. These cars are called *routers*: they can forward a message from a subnet to the next. Note that a vehicle could be both a server and a router. It could also be server for more than one subnet.

## *Addressing*

An addressing scheme is needed to make sure every vehicle on the highway has a unique way to be identified. A car address is a 32 bit number, to be read as follows:

- first 8 bits: *highway identifier*. This number identifies the highway a vehicle is on;
- second 8 bits: *section identifier* within the highway. Each highway is subdivided in a sequence of sections. This number identifies the section the vehicle is on;
- third 12 bits: subnet address within the section. This number identifies the subnet the vehicle is part of;
- fourth 4 bits: car address within the subnet.

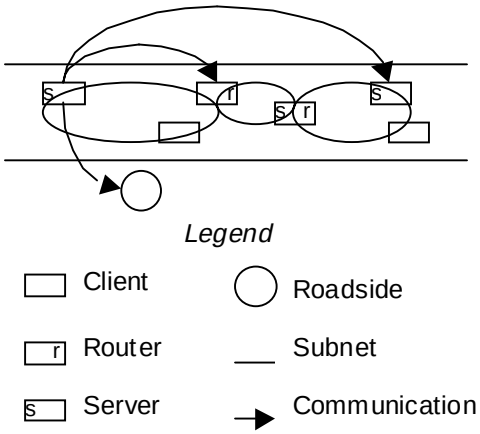
Note that the number of bits in each field could be changed in the future as a result of a design tune-up.

This scheme uses both static and dynamic information. The first two fields of the address refer to static data (highway and section identifiers are not likely to change very often). The third field is assigned dynamically by the system. The fourth field is assigned by the subnet server when new cars join the subnet.

The big difference with respect to IP addressing is that a car's address is not fixed. The first two fields are updated by the roadside when cars cross a highway or section border. The third field is assigned by the roadside when a subnet is created (the roadside makes sure that it's a unique number within the section). The third field also changes when crossing a highway or section border: in this case the roadside has to assign a new unique number within the new section.

In order for the roadside to update addresses, vehicles must provide the roadside with their position coordinates at fixed intervals on the CC (as explained below, not every car needs to send its position to the roadside, only the subnet server must do it).

Since routers belong to more than one subnet, they have more than one address (typically only the last two fields of their addresses are different).




Access to IP addresses outside the AHS (and in general to Internet protocols) could also be provided, through the roadside. This allows vehicle occupants to read/write e-mail, browse the net, etc. Of course such activities must be assigned a limited amount of bandwidth to make sure they don't interfere with AHS operations.

## *Subnet Management*

A car can create a new subnet, automatically becoming its server. After subnet creation the server waits for connection requests from other cars on the CC. Not every connection request is accepted: the application layer is in charge of providing the communication software with a list of *trusted hosts*. Only requests from trusted hosts are processed: this is necessary to

operate with a minimum amount of security (even though more work can be done in this field). The application gets the list of trusted hosts through an application dependent negotiation (that could take place on the CC).

All cars on a subnet use the same phisical channel to access that subnet (e.g. MPI or wavelan). It is the server's responsibility to ensure that cars requesting to join the subnet have a free channel of the needed type.

Each car stores the following information about each subnet it belongs to:

- server address;
- type of data link layer component used to access the subnet: e.g. Wavelan, MPI (note that there is a one to one relationship between data link layer types and physical channel types);
- list of all clients on the subnet.

The server sends out broadcast update messages whenever this information changes.

The subnet is dynamically re-configurable. When a car wants to leave the subnet, it informs the server. When the server wants to leave the net it chooses one of the clients to be the new server. In both cases all clients are updated.

Each car can send point-to-point, multicast or broadcast messages to other cars. The network layer must ensure that multicast and broadcast messages can be sent even using a point-to-point channel. Point-to-point communication is acknowledged.

In this respect the possible actions performed by a message sender at the network level are:

1. Send a message;

2. Receive an acknowledgment (for point-to-point messages only) and
   a) accept the acknowledgment or
   b) reject the acknowledgment;

3. Forward the acknowledgment to the next car (if the message is point-to-point on a point-to-point channel and the message is not for the current car).

The possible actions performed by a message receiver at the network level are:

1. Receive a message and
   a) Accept the message and pass it to the upper layer or
   b) Reject the message;

2. Forward the message to the next car (in the caseof a broad/multicast message on a point to point channel);

3. Send an acknowledgment (for point-to-point messages only).

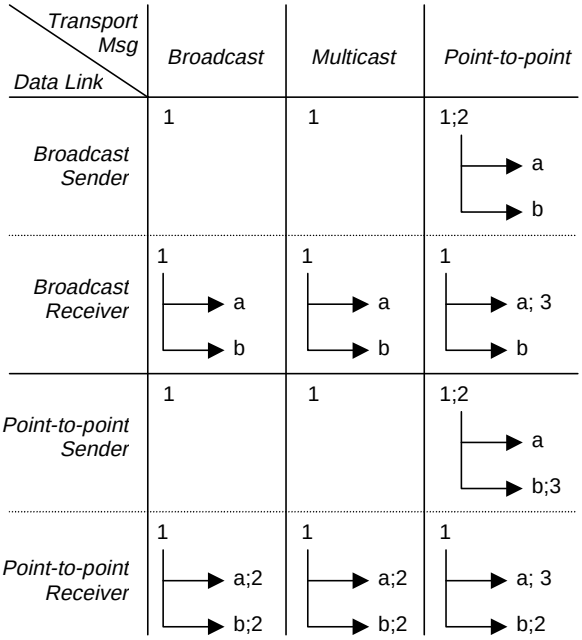
Since a transport message can be broadcast, multicast or point-to-point and a data link component can be broadcast or point to point, there are six possibile behaviors. In the figure arrows stand for selection and ";" stands for sequence.

## *Routing*

A car can send a message to a car that does not belong to the same subnet. To achieve this, two kinds of message routing are used: *link state* and *hierarchical routing*.

### Link State Routing

Link state routing relies on a highly dynamic algorithm. It doesn't make any assumptions on the configuration of the network: routers have to gather topology information periodically. This feature makes this algorithm very suitable to a dynamic environment like an AHS.




Link state routing is structured in five steps, to be repeated periodically because network topology changes rapidly on a highway (it is supposed that at time 0 routers know nothing about their neighbors).

1. Discover Neighbor Routers.

   Given a router A, it broadcasts a special `HELLO` packet on each of the subnets it belongs to. The packet has the following form:

| `HELLO` | `Sender Address` |
|---|---|

   Every router receiving a `HELLO` packet sends back a `HELLO_ACK` packet on the same subnet the `HELLO` packet arrived on. The `HELLO_ACK` packet has the following form:

| `HELLO_ACK` | `Sender Address` |
|---|---|

   After receiving the acknowledgments, router A will have the following information:

| *Router* | *Subnet* |
|---|---|
| B | Subnet 1 |
| C | Subnet 1 |
| D | Subnet 2 |
| … | … |

   The subnet address is composed of the first three fields of each router's address.

2. Measure delay (and QOS) to Neighbors.

   Router A sends a special `ECHO` packet to each of the routers it discovered in step one. The packet has the following form:

| `ECHO` | `Sender Address` |
|---|---|

   Every router receiving a `ECHO` packet sends back a `ECHO_ACK` packet. The `ECHO_ACK` packet has the following form:

| ECHO_ACK | Sender Address |
|---|---|

Router A measures the time that is needed to receive an acknowledgment (it can also gather other information on QOS for each line; note that QOS is not defined in this context, since it is a transport layer concept).

After receiving the acknowledgments, router A will have the following information:

| *Router* | *Subnet* | *Delay* | *QOS* |
|---|---|---|---|
| B | Subnet 1 | 30 ms | QOS1 |
| C | Subnet 1 | 20 ms | QOS2 |
| D | Subnet 2 | 60 ms | QOS3 |
| … | … | … | … |

3. Build Link State Packets.

Router A creates a *Link State Packet*, which has the following structure:

| Sender Address | | | |
|---|---|---|---|
| Sequence Number | | | |
| Age | | | |
| Bounces | | | |
| B | Subnet 1 | 30 ms | QOS1 |
| C | Subnet 1 | 20 ms | QOS2 |
| D | Subnet 2 | 60 ms | QOS3 |
| … | … | … | … |

The first field of the packet is the sender address (in this case A. The second, third and fourth fields will be explained in step number 4. The other fields contain the information that was gathered in step number 2.

4. Distribute Link State Packets.

Router A sends out the link state packet on all the subnets it belongs to (*flooding*). The `Sequence Number` for the first packet is 0. Each time a new packet is sent out the `Sequence Number` is increased. Receiving routers keep track of the following information for each router they receive link state packets from: router address, latest `Sequence Number`.

When a new link packet is received, if its `Sequence Number` is less than the stored number, the packet is considered obsolete. If it's equal to the stored number the packet is considered a duplicate. Only if it's greater than the stored number the received packed is kept as an update and it is forwarded on every subnet, except the one it came from.

The `Bounces` field is initialized with a fixed number at packet creation time. Each time the packet is forwarded its `Bounces` field is decreased by one unit. When it reaches 0, the packet is not forwarded anymore. This prevents the packet from going too far.

The `Age` field is used to add robustness to the algorithm. It contains the number of seconds the packet can live in the router once it is received. After this time expires the packet is discarded by the router. To explain the use of the `Age` field let's suppose the sequence number of a packet is corrupted, becoming 64169 instead of 10. If the age concept wasn't introduced, all new packets would be discarded as obsolete until the sequence number reaches 64170.

5. Computing New Routes

Once a full set of link state packets is received, routers can build a network graph and they can use Dijkstra's shortest path algorithm (or any other such algorithm) to compute the shortest path to each node.

Once the path to a router is known, all cars on its subnet become accessible in the same way. Their address is the same as the router's, except for the fourth field. They can be reached using the same path that is used to reach the router.

### Hierarchical Routing

Link state routing helps reach cars in the immediate vicinity (a few subnets away), but it can't help reach a car that lies tens of miles ahead. To solve this problem another kind of routing must be used together with link state routing: hierarchical routing.

As said above, subnet servers are requested to send position information to the roadside periodically. In this way the roadside can update vehicle addresses when cars cross a highway section border. Other cars on the same subnet need not send position information. When a server crosses a border the roadside sends it the new subnet address (3 fields). The server then updates all the clients.

This also means that the roadside always knows the approximate position of every subnet (and therefore of the vehicles within it). When the destination address of a message is not in the routing table of the router, the message is forwarded to the nearest roadside on the CC. The roadside knows the position of the destination subnet's server and forwards the message to the roadside entity on the proper highway section. From there the message is sent to the subnet server and then to the destination car.

This dual routing approach, mixes dynamic and static information and enables the system to deal with a wide variety of situations.

## *The Network API*

The network layer API contains both server-side and client-side functions. The API is presented in pseudo-C code

The following data types are needed:

- `CommAddress`: a 32 bit integer that can be interpreted both as an AHS address (as described above) or as an IP address;
- `GroupId`: a 32 bit integer that identifies a group of cars. It is used for multicast communication;
- `NetMessage`: a data structure containing a Network Layer message.

Server-side functions:

- `CommAddress setServer()`:
  this function is used to create a new subnet and to define the calling vehicle as the server for the subnet. The return value is a unique address for the vehicle (it is assigned by the roadside). The address contains the id of the highway, the id of the section, a new subnet id and the id of the server (typically 1).

- ```
  void waitForConnection(
    CommAddress addr,
    char * trustedHosts[]):
  ```
  this function is used to wait for connections on subnet `addr` (only the first three fields are considered). The reason to pass `addr` as a parameter is that a car could be server for more than one subnet. The `trustedHosts` array contains a set of car names from which the server is going to receive join requests. This list is known at the application layer (e.g. as a result of a negotiation on the common channel between server and clients). It is the application's responsibility to pass the proper list as a parameter (of course the application layer cannot access directly the network layer, so it has to pass the trusted host list to the transport layer, which in turn passes it to the network layer). Only connection requests from trusted hosts are accepted. This process is needed to ensure a minimum level of security in AHS operations.

- ```
  void serverLeaveNet(
    CommAddress addr):
  ```

this function is used by the server to leave a subnet it is managing. The reason to pass `addr` as a parameter is that a car could be server for more than one subnet. Before leaving, the server chooses a new server among the clients and warns them about the change with a broadcast message.

Client side functions:

- `CommAddress requestConnection(`
  `CommAddress server)`:
  This function is used to join the subnet that is managed by `server`. The returned value is a unique address for the client: the first three fields of the address are the same as the server's address, the fourth one is a unique number that is assigned by the server. If there are no available channels of the same type as the one that is used by subnet hosts, the function fails.

- `void clientLeaveNet(`
  `CommAddress server):`
  this function is used by the client to leave the subnet that is managed by `server`.

Both sides functions:

- `GroupId setGroup (`
  `CommAddress addr1,`
  `…)`:
  this function returns a group identifier for a set of car addresses. The group identifier can then be used when sending multicast messages.

- `void send(`
  `NetMessage * msg,`
  `CommAddress dest,`
  `int type)`:
  this function sends a point-to-point message `msg` to car `dest`. The type parameter specifies whether `dest` is an IP address or an AHS address.

- `void sendMulticast(`
  `NetMessage * msg,`
  `GroupId dest)`:
  this function sends a multicast message to all vehicles in group `dest`.

- `void sendBroadcast(`
  `NetMessage * msg,`
  `CommAddress server)`:
  this function sends a broadcast message to all cars in the subnet managed by `server`.

- `NetMessage * receive(`
  `CommAddress server1,`
  `…)`:
  this function blocks execution flow and waits until a jmessage is received on any of the indicated subnets (it's useful to remember that there is a one to one relationship between the server and the subnet). The calling vehicle must belong to all indicated subnets.

## CONCLUSIONS AND FUTURE WORK

Subnet managment protocols have already been designed (they were not presented in this paper because of lack of space). Future work includes: finishing routing protocols and validating the system through simulation. Transport layer design should be the object of the next phase of the research.

## REFERENCES


- A. Tanenbaum. 1996. "Computer Networks". Prentice Hall.
- Jean Walrand and Pravin Varaiya. 1996. "High-Performance Communication Networks". Morgan Kaufmann Publishers.
- Farokh Eskafi and Valerie Murgier. 1998. "Smart-AHS Components For Simulating Communication". PATH internal report (available at http://www.path.berkeley.edu/smart-ahs/sahs-manual/communication_devices.html).