Highway Information System

Protocol Specification V1.0

Bruno Crisafulli - Agustin Rodriguez

Abstract

This document aims to explain the functioning of the protocol used by the core of the system to achieve the tracing of a fleet of vehicles and the distribution of information between them, in a way that its decentralized and fault tolerant.

General considerations

The protocols aims to support the creation of P2P networks between the vehicles that are “near” each other and also support a platform of accesspoint-monitor nodes (that know each other through a coordinator) for the vehicles to report to and that allow them to send and receive global messages.

Each node has a UUID string that identifies it on the net.

Three types of nodes are defined:

- Car : represents the vehicles to be monitored.
- Highway-Node (HwNode): gateway for the vehicles to the system, allowing the bootstrap of the peer-discovery process; also necessary for transmitting global messages.
- Highway-Coordinator (Coordinator): it is the node which defines the scope of the system, monitors the status of the Highway-Nodes and allows new Highway-Nodes to register dynamically in the system.

Transport Protocols

The communication between HwNode and Car nodes is done over UDP..

The communication between the Coordinator and HwNodes is done over TCP.

The protocol makes use of UDP for all the types of messages that do not need the properties of a TCP connection and thus avoid the overhead in the network that those connections would imply.

Messages

Currently the serialization of Objects and Standard Types is left handled by the Java language and no compression its contemplated in the protocol, so no actual byte level specification will be given by this document. Refer to the Java 9 serialization specification if needed.

Message Structure

{Msg_Type :

Hello | Hello Response | Register | Pulse | Alive | Redirect | Update |Broadcast

Id : UUID ,

Response_Id : String,

Origin_Ip: String,

Body: optional, depending on the Msg_Type}

Message Types

Some messages contain contact information to represent a node, this is:
Standar node contact info: { Id: UUID,
Ip: String,
Port: Integer }

The following message types are defined:

- **Hello**: the HELLO message represents a requisition of a Car to register to the receiver, soliciting a list of the other Cars known by it. This message also contains a Pulse in order to inform the sender´s position and velocity.

  If the receiver it's a HwNode, the list of known Cars in the response must contain all the cars currently in the Segments managed by it. In the case that the position of the sender doesn't belong to any of the Segments managed by the HwNode, it is expected to respond with a REDIRECT message to the HwNode that corresponds with that position, or an ERROR message in the case that the position doesn't belong to a valid Segment.

- **Hello Response**: This message can be sent by both Cars and HwNodes. This message must be sent in response to a HELLO message from a Car. The message will contain the list of Cars that the sender knows. In the case of a HwNode this message must be sent only if the Car of the corresponding HELLO is registered with success.

- **Register**: Represents a request of registration from a HwNode to the Coordinator Node, signaling the desire to form part of the network of HwNodes in the system.

- **Pulse:** this message must be emitted periodically by the registered Cars to all the nodes he knows including neighbour Cars and HwNodes which is registered in. The message contains the current location and velocity.

- **Alive**: This type of message is used to monitor the online status of the nodes.
  The Coordinator Node sends ALIVE messages periodically to monitor the status of the HwNodes and detect failures. In this version the Coordinator attempts to establish a TCP connection with the HwNode to send the message, if this fails, the node is considered offline and starts the **Coordinator routine of recovery from HwNode drop (see page 15)**.
  The HwNodes in turn, send ALIVE messages periodically to each one of its registered Cars over UDP, but this time the message is used by the Car to detect failures in the HwNode, in which case starts the **Car routine of recovery from selected HwNode drop (see page 15).**
  ALIVE messages does not have Body.

- **Redirect**: this type of message is sent by the HwNodes to the registered Cars in order to indicate which other HwNode the Car must try to Register on, given that because of his position it shouldn't communicate with the HwNode sender of the message.
  The Body of the message contains the contact-info of the HwNode which is being redirected to.

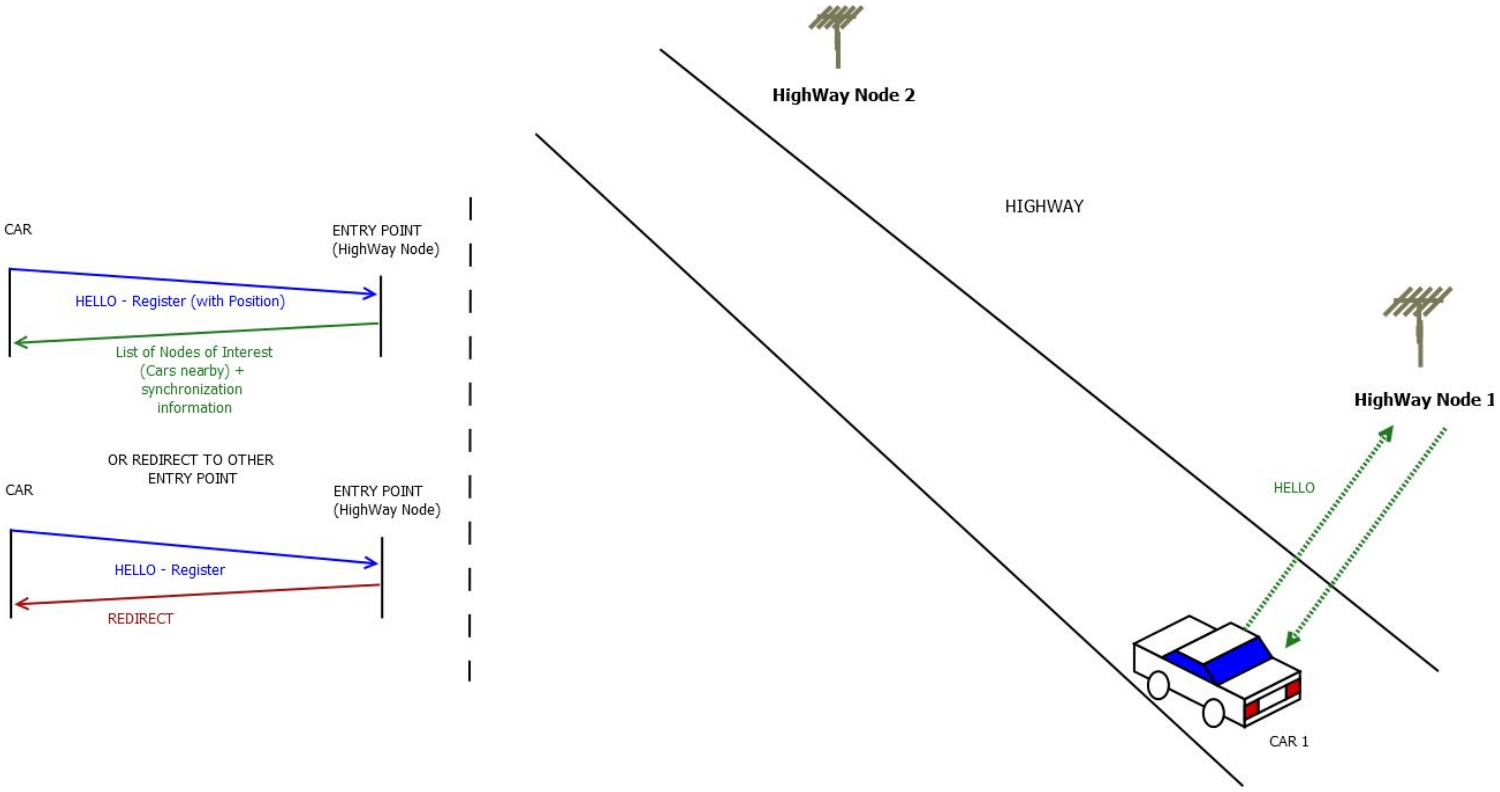
- **Update**: this type of message is sent by the Coordinator Node in order to communicate the global list of all the HwNodes including the Segments which each one has to administer. This message is emitted when a change occurs in the distribution of the Segments, caused either by a failure of a HwNode or the registration of a new one. The body of the Message consists in a list of HwNode contact-info including the list of Segments that the node is in charge of.

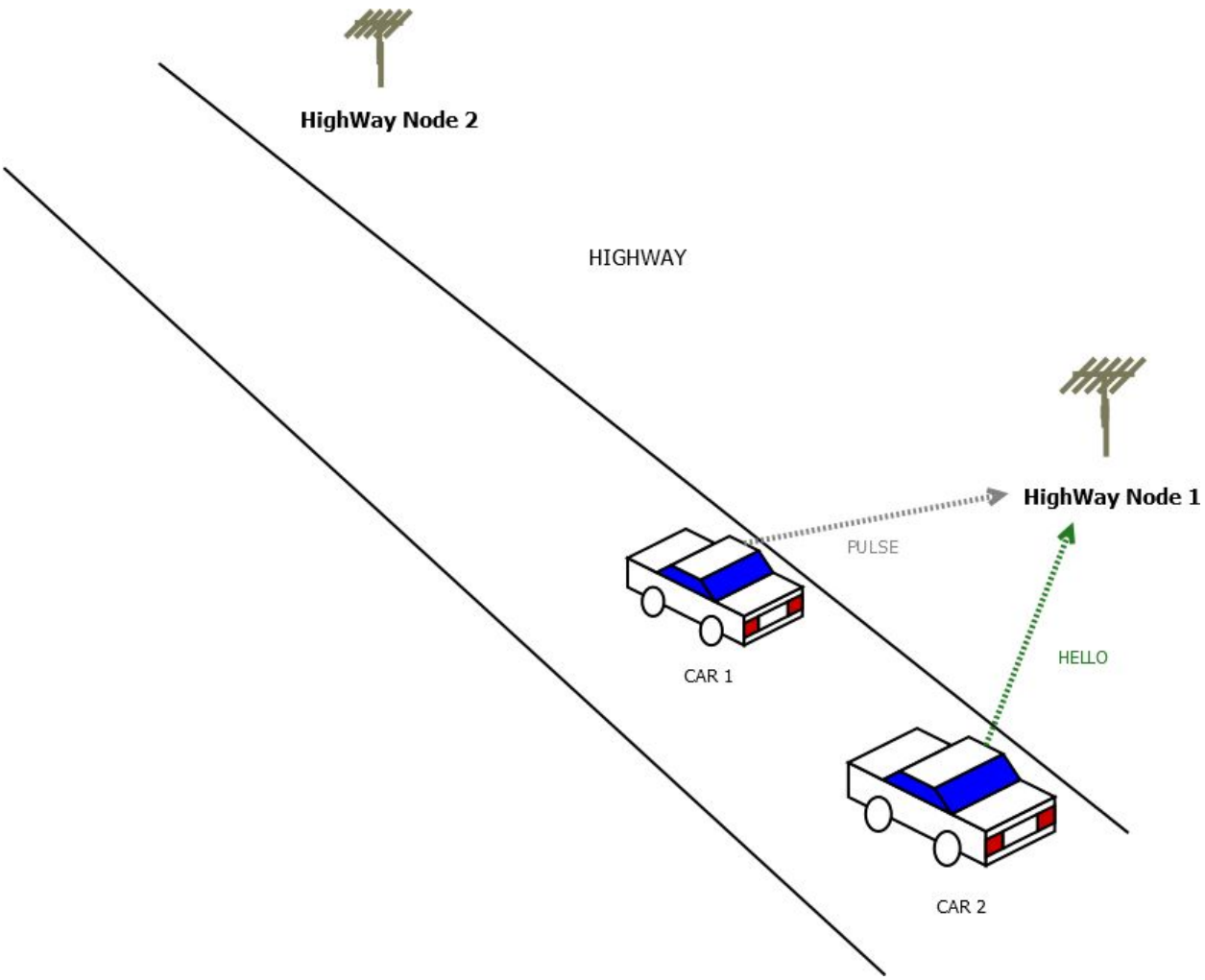
- **Broadcast** : this type of message must be propagated through all the net, reaching every node. It is asynchronous and its usage can be diverse depending on the content of it. The current method of propagation is Partial Flooding, limited by a hop counter (TTL) that decreases. Body of the message: Broadcast ID (string), TTL (int), Timestamp, Content (String) and a boolean that indicates whether it is sent by a HwNode or a Car.

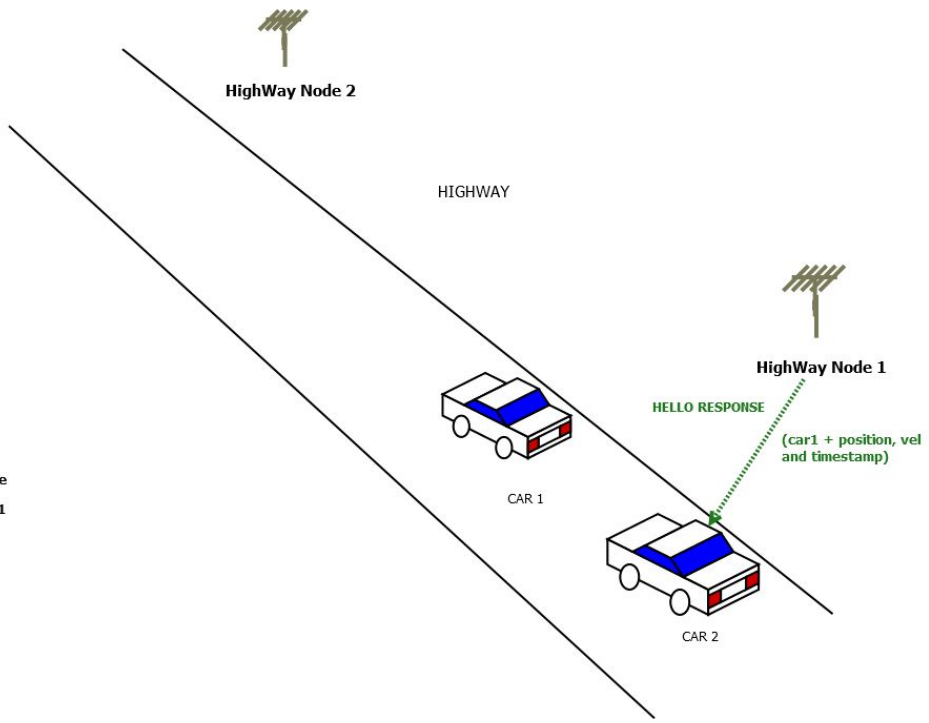
Summary of Body by type of message

- Hello:
    - {Pulse = [Position: (coordX:double, coordY:double),
      Velocity: double, TimeInstant]}
- Hello Response:
    - {Cars = List of CarStNode =[id, ip, port, Pulse] }
- Register:
    - {HwStNode =[id, ip, port_cars, port_hw]}
- Pulse:
    - {Pulse = [Position: (coordX:double, coordY:double),
      Velocity: double, TimeInstant]}
- Alive:
    - {}
- Redirect:
    - {RedirectNode = [id, ip, port]}
- Update:
    - {MT_Update[List<HwStNode>]}
    - {HwStNode = [StNode,StNode,List<Segments>]
    - {Segments = [double, double, double, double, int]}
- Broadcast:
    - {MT_Broadcast[broadcast_id, is_from_car: boolean, message: String]}

Peer-discovery example




HighWay Node 2
HIGHWAY
HighWay Node 1
PULSE
HELLO
CAR 1
CAR 2

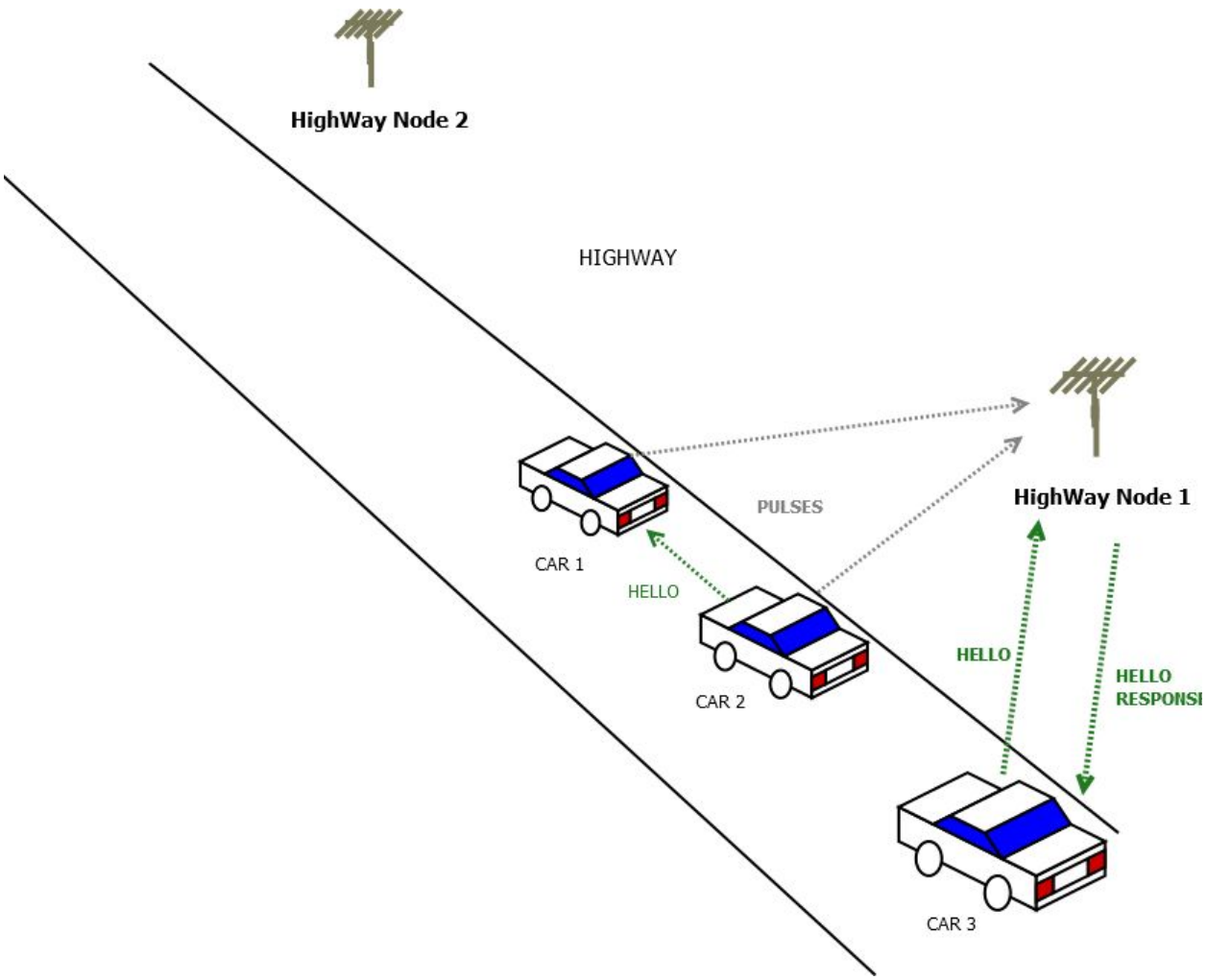


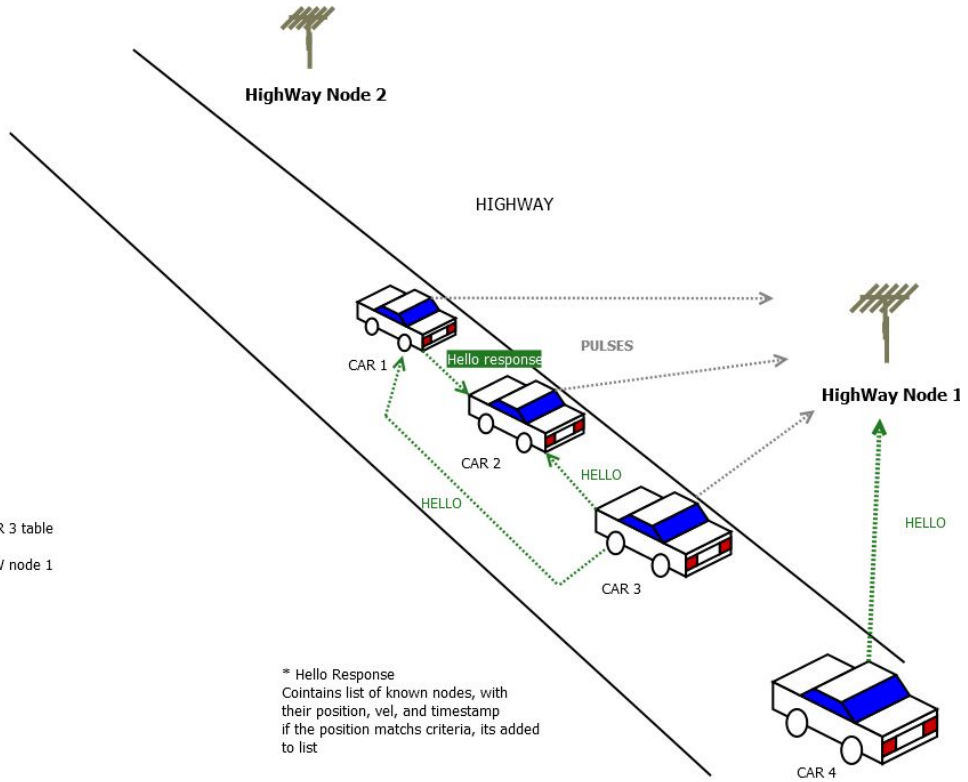
CAR 1 table

. HW node 1

CAR 2 table

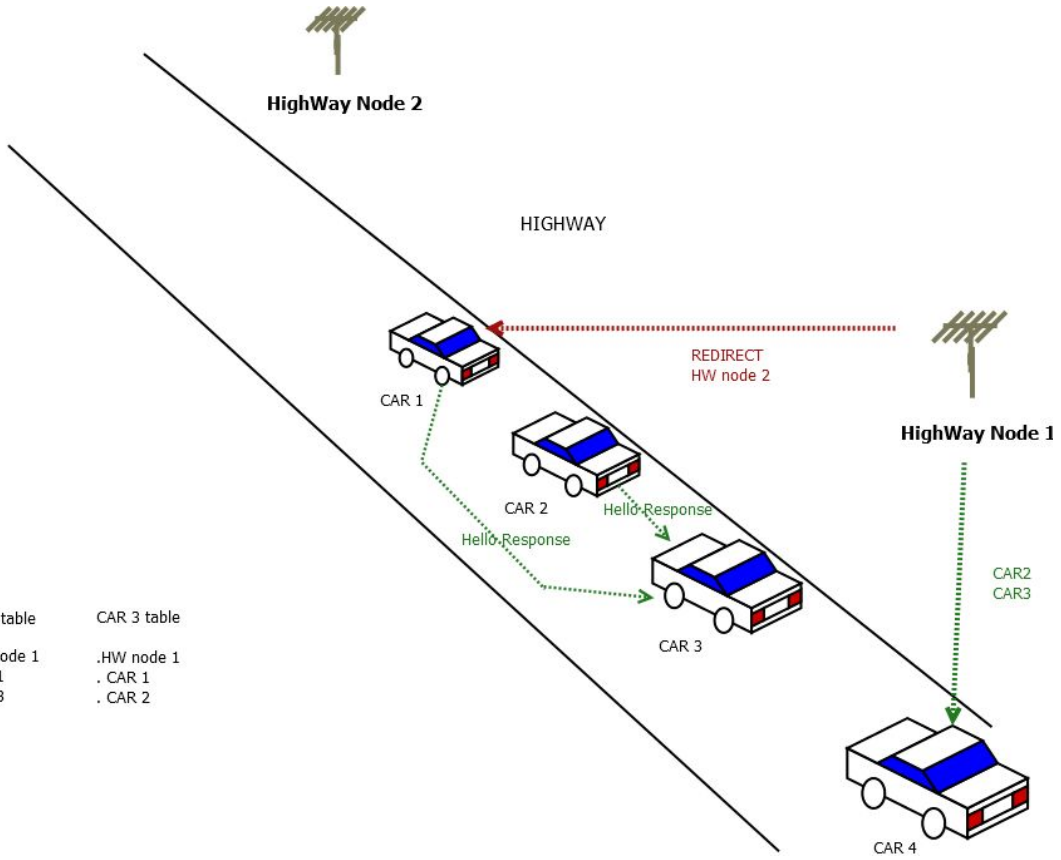
. HW node 1


HighWay Node 2
HIGHWAY
HighWay Node 1
PULSES
CAR 1
HELLO
CAR 2
HELLO
HELLO
RESPONSI
CAR 3



HW Node 1 Table
. CAR 1
. CAR 2
. CAR 3

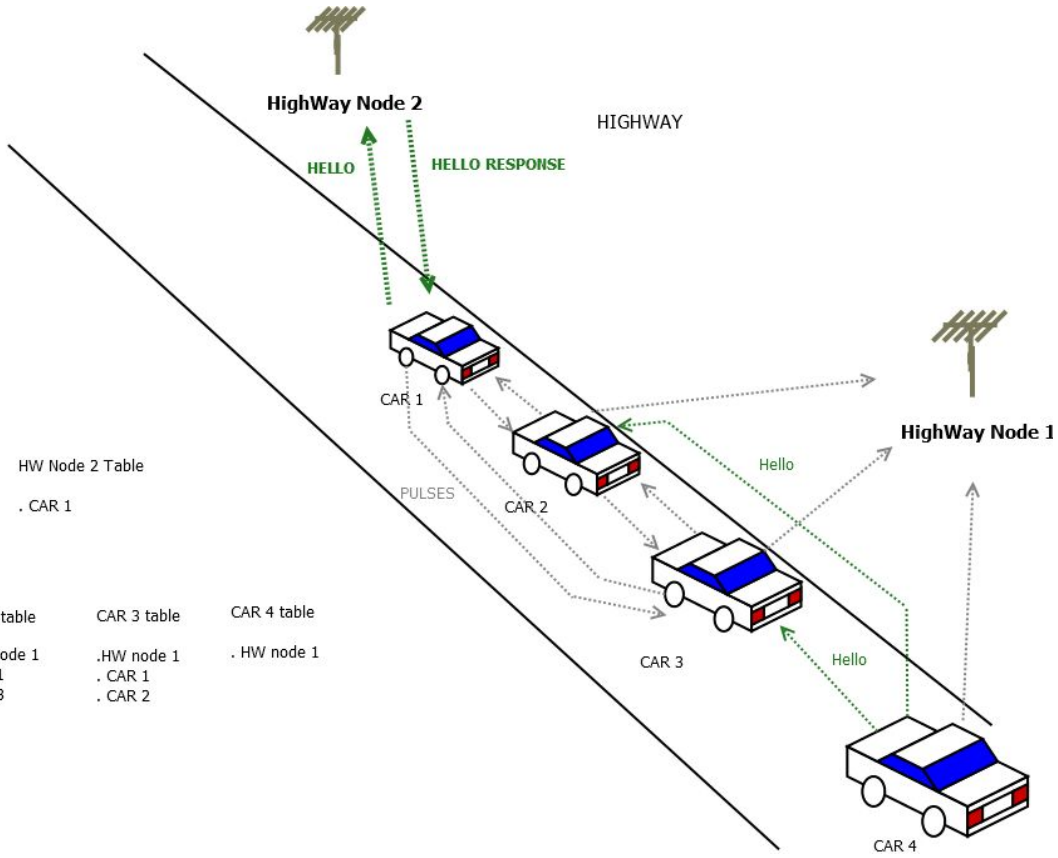
CAR 1 table	CAR 2 table	CAR 3 table
. HW node 1 . CAR 2	. HW node 1 . CAR 1	.HW node 1



HW Node 1 Table

. CAR 2
. CAR 3
. CAR 4

CAR 1 table	CAR 2 table	CAR 3 table
. HW node 1 . CAR 2 . CAR 3	. HW node 1 . CAR 1 . CAR 3	.HW node 1 . CAR 1 . CAR 2



HW Node 1 Table

. CAR 2
. CAR 3
. CAR 4

HW Node 2 Table

. CAR 1

CAR 1 table

. HW node 2
. CAR 2
. CAR 3

CAR 2 table

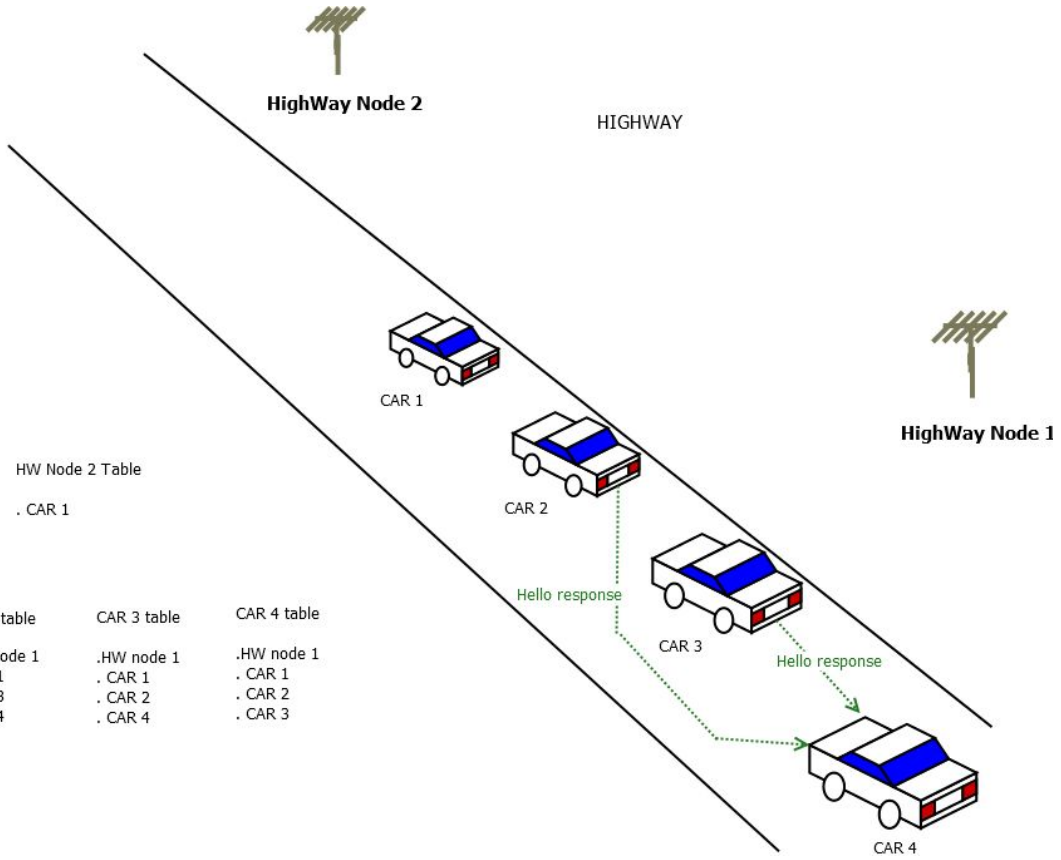
. HW node 1
. CAR 1
. CAR 3

CAR 3 table

.HW node 1
. CAR 1
. CAR 2

CAR 4 table

. HW node 1



HW Node 1 Table

. CAR 2
. CAR 3
. CAR 4

HW Node 2 Table

. CAR 1

CAR 1 table

. HW node 2
. CAR 2
. CAR 3

CAR 2 table

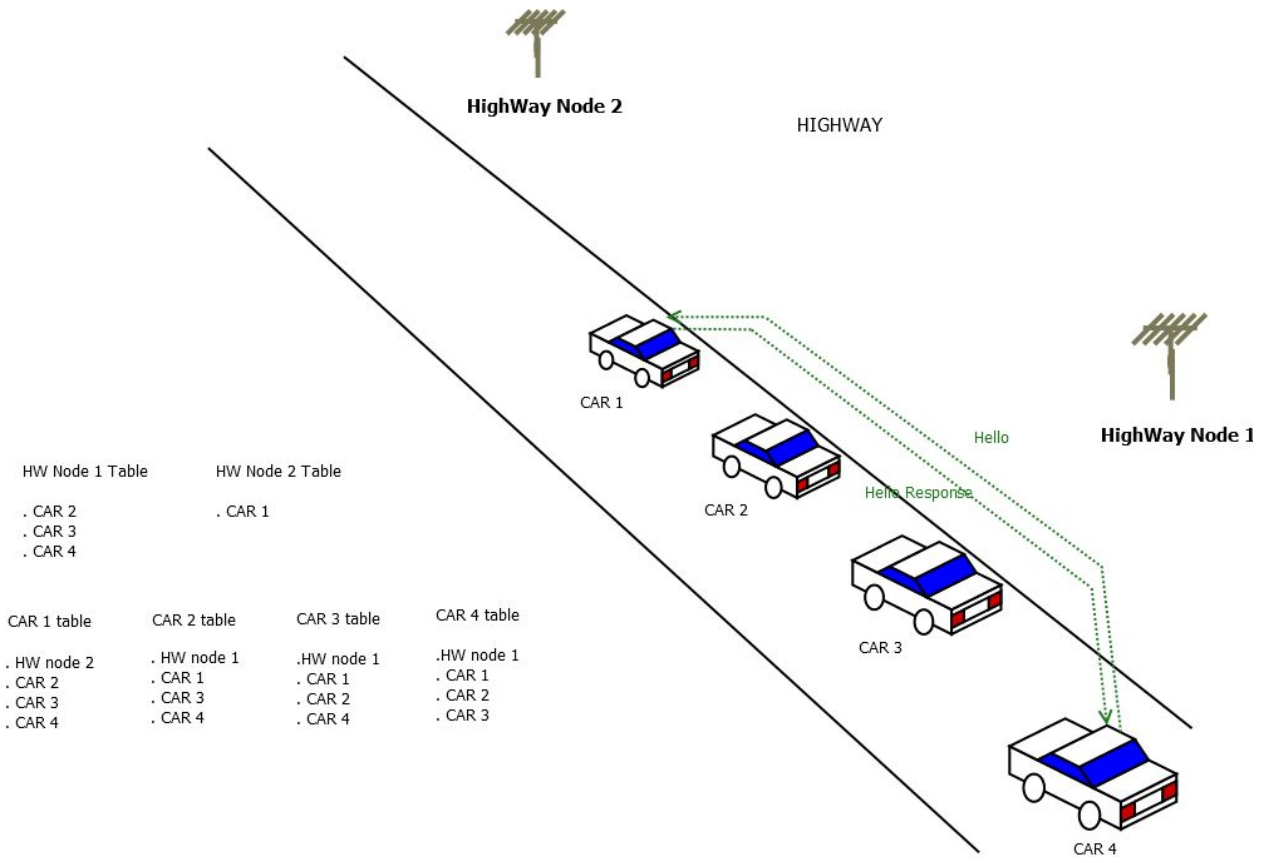
. HW node 1
. CAR 1
. CAR 3
. CAR 4

CAR 3 table

.HW node 1
. CAR 1
. CAR 2
. CAR 4

CAR 4 table

.HW node 1
. CAR 1
. CAR 2
. CAR 3



Note that the ALIVE messages that the HwNode must send periodically to his vehicles was not visualized in these diagrams.

Fault tolerance

Coordinator routine of recovery from HwNode drop

Once the coordinator detects the drop of a HwNode, it must re-assign the Segments belonging to the fallen HwNode to the immediately adjacent one. Then it must communicate the change to all the remaining HwNode through UPDATE messages.

Car routine of recovery from HwNode drop

When a Car detects the drop of the assigned HwNode, it must re-attempt the registration in the system in the same way as in its entry, that is, sending a HELLO message to each HwNode preloaded in its configuration until it gets a response of type HELLO-RESPONSE.

What happens when the Coordinator drops?

Once a HwNode detects the offline status of the Coordinator, due to the amount of time passed since last ALIVE received, it must retry the registration in the same way as in its instantiation, that is, sending a REGISTER message to each possible coordinator pre-loaded in its configuration. Until it receives an UPDATE, it should continue to operate normally as if the coordinator never went down.

In this way, the Cars and the system in general are not affected immediately by the drop of the Coordinator: everything continues to work as before the drop, except that now the registration of new HwNodes and the reorganization due to HwNode drops is disabled until a new Coordinator goes Online on any of the domains defined in the HwNode´s configuration files.

Possible Improvements

Coordinator dynamic replacement

The HwNodes upon detection of the Coordinator´s drop, could initiate a consensus algorithm to determine which of them will assume the role of coordinator. But it will need a mechanism to redirect new aspiring HwNodes to know the ip and port of the new coordinator.

Shared Resources Versioning

Some versioning system could be used to keep track of the shared resources like the list of HwNodes and the Segments.

Data centralization

Other possible inclusion in the protocol could be methods and entities to back-up, aggregate and centralize the information generated by the pulses of the Cars.

Data compression

Given that the protocol makes use of UDP to avoid overhead, at least some compression has to be contemplated to reduce network usage.