There is no generic VANET protocol; however there is a set of DSRC primitives recommended for construction of interoperable VANET applications. Based on those building blocks, we can derive a generic application protocol for VANETs.

VanetApp ::= SEQUENCE{
    Priority     -- the urgency of this message, a relative
               -- degree of merit compared with other messages;
               -- it is a function of the application type
               -- (ApplicationID expanded further down)
    Duration     -- provides a gross level of range (distance) of
               -- applicability for the message (e.g. some messages will
               -- no longer will be relevant as intended recipients
               -- move some distance away
    CHOICE     
        Basic_Safety_Message
            -- message contains vehicle safety-related
            -- information, periodically broadcasted to surrounding vehicles
            -- transmitted on Public Safety Channel or Critical Safety channel
            -- depending on urgency. Certain data is sent with every instance
            -- of the message, other information is sent periodically
            -- selectively based on the requests of other nearby vehicles
        Common_Safety_Message
            -- allows for specific vehicle safety-related
            -- information requests to be made that are required by vehicle
            -- safety applications.
            -- to follow traditional request/response form, the initiator
            -- sends Common_Safety_Message; the responder replies by
            -- sending Basic_Safety_Message
        Probe
            -- contains status information (snapshots)
            -- about the vehicle for different periods of time that is
            -- broadcasted to the VANET nodes both (V->V and V->I).
        Emergency_Vehicle_Alert
            -- emergency vehicle alert message is used for broadcasting
            -- warnings to surrounding vehicles that an emergency vehicle
            -- is operating in the vicinity.
            -- NOT RELEVANT for the purpose of this paper, not expanded
        Generic_Transfer
            -- NOT RELEVANT for the purpose of this paper, not expanded
    }

Basic Safety Message consists of three primary parts with a slightly different message encoding plan used in each:

- **Part I** is VANET-specific. It part is mandatory and is always sent at all times without exception. The data structures, their order and their length are known in advance (fixed). There are no optional elements; therefore, there is no tagging.

- **Part II** is VANET-application specific. It consists of predetermined data frames and data elements of known length which have to follow in a defined order but may or may not be present in a specific instance of a message (optional elements). Since the number of elements differ from application to application, to avoid ambiguity, each element is prepended by a tag.

- **Part III** is application-vendor specific. It allows the creator of the message to add additional content (can be defined privately) by specifying his unique needs through a sequence of delimited tags and corresponding values.
Basic_Safety_Message ::= SEQUENCE {
  -- Part I, sent at all times without any tagging
  MessageID
  MessageType
  Timestamp       -- 2 bytes
  TempID     -- 6 bytes

  -- group position
  lat   -- 2 bytes
  long  -- 2 bytes
  elev  -- 1 bytes

  -- group motion
  speed  -- 2 bytes
  heading -- 1 byte
  accelSet -- accel set (four way)

  -- group control
  brakes -- 1 byte
  steering -- 2 bytes
  throttle -- 1 byte
  light   -- 1 byte
  ABS     -- 1 byte (antilock brakes)
  traction -- 1 byte
  etc.

  -- group basic
  length -- 1 byte (vehicle length)
  width   -- 1 byte (vehicle width)

  -- Part II, sent as required with short tags
  valueCnt    INTEGER (0..32) OPTIONAL, -- 1 byte
  items SEQUENCE (SIZE(0..32)) OF SEQUENCE {
    tag SomeTagList,                 -- 1 byte
    value CHOICE {
      -- pick any single item/group below,

      -- group Precisions
      positionPresision
      timePresision
      speedPresision
      accelYawPresision

      -- various status values
      brakePres
      brakeBoostApplied

      -- misc.
      health
      height
      type
      mass

      -- other status values, not important at this point
    } OPTIONAL, -- #UNTAGGED
  },

  -- Part III, sent as required with open ended tags
  tailCnt    INTEGER (0..32) OPTIONAL, -- 1 byte
  tail
  ... -- # LOCAL_CONTENT
}
MessageID, ::= ENUMERATED {
  reserved          (0),
  BasicSafety       (1),
  CommonSafety      (2),
  EmergencyVehicleAlert  (3),
  GenericTransferMsg (4),
  ProbeVehicleData (5)
}
MessageType ::= ENUMERATED {
  -- Safety System messages
  item_1      (0),  -- Basic
  -- Safety Vehicle messages
  item_2      (1),  -- Basic
  item_3      (2),  -- Common
  item_4      (3),  -- Connect
  item_5      (4),  -- Identify
  item_6      (5),  -- Notice
  item_7      (6),  -- Synchronize
  item_8      (7),  --
  item_9      (8),  --
  item_10     (9),  --
  item_11     (10), --
  item_12     (11), --
  item_13     (12), --
  item_14     (13), --
  item_15     (14), --
  item_16     (15), --
  -- Safety Infrastructure messages
  item_xx     (16), -- Basic
  item_xx     (17), --
  item_xx     (18), --
  item_xx     (19), --
  -- Safety Environment messages
  item_xx     (20), -- Local Basic
  item_xx     (21), --
  item_xx     (22), -- Global Basic
  item_xx     (23), --
  -- Rest to be defined later,
  -- will there be any 'local' types here?
  ...
}

Common Safety Message provides a means by which a vehicle participating in the exchange of the basic safety message can broadcast requests to other vehicles for addition information which it required for the safety applications it is actively running. Responding vehicles will add this information to the appropriate place in the basic safety message when replying.

CommonSafetyMessage ::= SEQUENCE {
  MessageID
  request2Cnt INTEGER (0..32) OPTIONAL, -- 1 byte
  requests2  SEQUENCE (SIZE(0..32)) of
    PartIIRequests, -- Part II items
  request3Cnt INTEGER (0..32) OPTIONAL, -- 1 byte
  requests3  SEQUENCE (SIZE(0..32)) of
    PartIIIRequests, -- Part III items
...
}
PartIIRequests ::= ENumerated {  
  reserved                       (0),  
  reqAccelandYawPrecision        (1),  
  reqAntiLockBrakeStatus         (2),  
  reqBrakeAppliedPressure        (3),  
  reqDFullTime                   (4),  
  reqFullPosition                (5),  
  reqHeadlightsStatus            (6),  
  reqPositionPrecision           (7),  
  reqPrecisionSet                (8),  
  reqSpeedandHeadingPrecision    (9),  
  reqSpeedPrecision             (10),  
  reqSystemHealth               (11),  
  reqTimePrecision              (12),  
  reqTractionControlState       (13),  
  reqVehicleHeight              (14),  
  reqVehicleMass                (15),  
  reqVehicleType                (16),  
  ... -- # LOCAL_CONTENT
}  
-- values to 127 reserved for std use  
-- values 128 to 255 reserved for local use  
PartIIIRequests ::= UTF8String (SIZE(1..20))

A Probe Data Message consists of a series of Probe Data Snapshots taken autonomously as the vehicle travels. These snapshots consist of all Probe Data elements that are available on the vehicle along with the time and location when each snapshot was taken. Probe Data Snapshots are taken either based on elapsed time since the last snapshot (i.e. snapshots are taken every x seconds) or based on an event trigger. For time driven snapshots, the faster the vehicle is traveling the shorter the interval between snapshots. For event triggered snapshots, there is a set of event on which a snapshot will be generated (e.g. complete stop for more than 5 sec, AntiLock Break system gets engaged, vertical acceleration exceeds some threshold, etc.)

Probe ::= SEQUENCE {  
  MessageID  
  initialVector -- space vector of the first measurement set  
  ProbeHeader -- contains the message id, sender, time, etc.  
  Snapshot+ -- concatenation of one or more snapshots  
  -- snapshot is a collection of all vehicle status elements  
  -- at one place and time often referred to as beacon in  
  -- many VANET papers. It is sent periodically, on event,  
  -- and when a vehicle stops
}

SpaceVector ::= SEQUENCE {  
  lat -- a 4 byte value  
  long -- a 4 byte value  
  heading -- a 1 byte value from geoids north  
  speed -- a 2 byte field in units  
  -- of 0.01 meters per second  
  elev -- 3 byte field (elevation) OPTIONAL  
  techType -- a 3-bit value (how the position was obtained)  
  quality -- a 5-bit value (estimate of accuracy for this location)
}  
lat ::= INTEGER (-90000000..90000000)  
-- Valid Encoding rules:  
-- none(0x00) - no latitude value available
long ::= INTEGER (-180000000..180000000)
-- Valid Encoding rules:
-- none(0x00) - no longitude value available
-- range (0xD515AC00..0x2AEA5400) corresponding to the range of 180°
-- invalid (0xFFFFFFFF) latitude solution is not valid

heading ::= 1 byte value. Valid Encoding rules:
0 = due north, larger going clockwise(east of north)
(0..253)  360/254 degrees per bit
254 = no heading at all (stationary)determined by examining velocity
255 = heading not known

speed ::= INTEGER (-32765..32765) -- Units of 0.01 m/s

accelSet ::= SEQUENCE {
  longAccel -- Along the Longitudinal axis
  latAccel -- Along the Lateral axis
  vertAccel -- Along the Vertical axis
  yawRate
}

longAccel, latAccel, vertAccel ::= INTEGER (0..4095) -- LSB units are 0.01 m/s^2 (that is SAE recommendation; however, it can be expressed in g (9.80665 m/s^2)

AccelAndYawPrecision ::= SEQUENCE {
  yawRatePrec -- 3 bit
  accelPrec -- 3 bit
  steeringWheelAnglePrec -- 2 bit
}

quality ::= ENUMERATED {
  (0), -- quality better than 1 meter
  (1), -- quality better than 5 meters
  (2), -- quality better than 12.5 meters
  (3), -- quality better than 50 meters
  (4), -- quality better than 125 meters
  (5), -- quality better than 500 meters
  (6), -- quality better than 1250 meters
  (7), -- quality value unknown
} -- 3 bits, appends with techType to make one octet

technology ::= ENUMERATED {
  (0), -- technology type unknown
  (1), -- GPS technology only
  (2), -- differential GPS (DGPS) technology
  (3), -- dead reckoning system w/GPS
  (4), -- dead reckoning system w/DGPS
  (5), -- dead reckoning only
  (6), -- autonomous navigation system on-board
..., (31) -- feature is not working
} -- (0..31) 5 bits, appends with quality to make one octet

Other precision fields (e.g. speedPrecision, accelerationPrecision, etc.) are defined in similar manner. I will leave them alone for now not to clutter the big picture.
ProbeHeader ::= SEQUENCE {
    SpaceVector
    Timestamp
    ACID -- application class ID\footnote{May be used in conjunction with Application Context Marker (ATM) in future iterations [8]}
    length -- a 4 byte value
    num_snapshots -- a 1 byte value
}

Snapshot ::= SEQUENCE {
    cntVSDTs INTEGER (0..31), -- a count of how many vehicle
    -- status devices are reported type entries
    deviceTypes SEQUENCE (SIZE(0..31)) OF VehicleStatusDeviceType
    -- a sequence of name-value pairs
    -- which me may encode in the
    -- style as other msgs, need to see
    offset UpdateVector OPTIONAL,
    -- data to adjust the last PositionVector
    -- to reflect the next set of data,
    -- not present in the very last entry
    ... -- # LOCAL_CONTENT
} -- Estimated size 2 bytes per payload item plus 10 for next position

Timestamp ::= SEQUENCE {
    date -- formatted as YYYYMMDD (will not expand, self—explanatory)
    time -- formatted as HHMMSSssss, where ssss are decimal seconds
}

BrakeSystemStatus ::= SEQUENCE {
    BrakeAppliedStatus -- 4 bits
    TractionControlState -- 2 bits
    AntiLockBrakeStatus -- 2 bits
}

BrakeAppliedStatus ::= BIT STRING {
    allOff      (0), -- The condition All Off
    leftFront   (1), -- Left Front Active
    leftRear    (2), -- Left Rear Active
    rightFront  (4), -- Right Front Active
    rightRear   (8), -- Right Rear Active
    allOn       (15) -- The condition All On
}

TractionControlState, AntiLockBrakeStatus ::= ENUMERATED {
    notEquipped (0), -- Not Equipped
    off         (1), -- Off
    on          (2), -- On
    engaged     (3), -- Engaged
}

ApplicationID ::= ENUMERATED {
    (0), -- system
    (1), -- automatic-fee-collection
    (2), -- freight-fleet-management
    (3), -- public-transport
    (4), -- traffic-traveler-information
    (5), -- traffic-control
    ...,
    (12), -- emergency-warning
    ...}
(19), -- public-safety
(20), -- vehicle-safety
(21), -- on board diagnostics
} -- for all values refer to [8]