Algorithmes de Big Data adaptés aux réseaux véhiculaires pour modélisation de comportement de conducteur

Emilien Bourdy

UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

ÉCOLE DOCTORALE SCIENCES DU NUMÉRIQUE ET DE L'INGÉNIEUR

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE REIMS CHAMPAGNE-ARDENNE

Discipline : ***INFORMATIQUE***

Spécialité : Informatique

Présentée et soutenue publiquement par

EMILIEN BOURDY

Le 3 décembre 2018

Big Data Algorithms Adapted to Vehicular Networks for Driver's Behavior Modeling

---

Thèse dirigée par **MICHEL HERBIN**

JURY

Mme Lynda Mokdad,	Professeur,	Université de Paris 12,	**Présidente**
M. Herman Herbin,	Professeur,	Université de Reims Champagne-Ardenne,	**Directeur de thèse**
Mme Kandaraj Piamrat,	Maître de Conférences,	Université de Nantes,	**Co-Encadrant de thèse**
M. Herman Akdag,	Professeur,	Université de Paris 8,	**Rapporteur**
M. Lyès Khoukhi,	Professeur,	Université de Technologie Troyes,	**Rapporteur**
Mme Soumaya Cherkaoui,	Professeur,	Université de Sherbrooke,	**Examinateur**
M. Hacène Fouchal,	Professeur,	Université de Reims Champagne-Ardenne,	**Examinateur**
M. Éric Ollinger,	Ingénieur,	Ministère de la transition écologique et solidaire,	**Examinateur**

Cette thèse est financée par le projet Scoop@f

Remerciements

Je souhaiterais remercier l'ensemble du projet Scoop@f, qui m'a permis de découvrir de nouvelles voies avec les technologies innovantes développées lors de ce projet.

J'aimerais remercier le CReSTIC de m'avoir accueilli lors de l'élaboration de mon projet de thèse, et tout particulièrement la charmante Ida, sans qui, bien des mésaventures me seraient arrivées.

Je remercie le laboratoire LS2N de Nantes pour m'avoir accepté dans leur équipe de réseau.

Je remercie évidemment Michel et Kandaraj pour avoir supporter mes retards dans les différentes rédactions, et leurs tolérances quant à mon écriture. Mais aussi Hacène et Marwane qui m'ont épaulé, parfois violemment, et pour les voyages inoubliables passés avec eux. Et enfin Cyril, pour son accompagnement.

Je ne saurais omettre de mentionner Geoffrey avec qui j'ai le plaisir de partager le café depuis quatre ans. Mais aussi les autres membres du bureau qui ont supporté mon humour du jeudi : Brice, Kévin et Dimitri, ma plus vieille connaissance à ce jour (23 ans).

Je remercie aussi toute ma famille qui m'a soutenu tout le long de ce voyage sans forcément tout comprendre lors de nos discussions sur le sujet.

Enfin, je remercie grandement ma chère et tendre pour m'avoir supporté, surtout lors des vacances passées à écrire ce manuscrit. Et pour ses bons petits plats.

Abstract

Big Data is gaining lots of attentions from various research communities as massive data are becoming real issues and processing such data is now possible thanks to available high-computation capacity of today's equipment. In the meanwhile, it is also the beginning of Vehicular Ad-hoc Networks (VANET) era. Connected vehicles are being manufactured and will become an important part of vehicle market. Topology in this type of network is in constant evolution accompanied by massive data coming from increasing volume of connected vehicles in the network.

In this thesis, we handle this interesting topic by providing our first contribution on discussing different aspects of Big Data in VANET. Thus, for each key step of Big Data, we raise VANET issues.

The second contribution is the extraction of VANET characteristics in order to collect data. To do that, we discuss how to establish tests scenarios, and to how emulate an environment for these tests. First we conduct an implementation in a controlled environment, before performing tests on real environment in order to obtain real VANET data.

For the third contribution, we propose an original approach for driver's behavior modeling. This approach is based on an algorithm permitting extraction of representatives population, called samples, using a local density in a neighborhood concept.

Résumé

Les technologies Big Data gagnent de plus en plus d'attentions de communautés de recherches variées, surtout depuis que les données deviennent si volumineuses, qu'elles posent de réels problèmes, et que leurs traitements ne sont maintenant possibles que grâce aux grandes capacités de calculs des équipements actuels. De plus, les réseaux véhiculaires, aussi appelés VANET pour *Vehicular Ad-hoc Networks*, se développent considérablement et ils constituent une part de plus en plus importante du marché du véhicule. La topologie de ces réseaux en constante évolution est accompagnée par des données massives venant d'un volume croissant de véhicules connectés.

Dans cette thèse, nous discutons dans notre première contribution des problèmes engendrés par la croissance rapide des VANET, et nous étudions l'adaptation des technologies liées aux Big Data pour les VANET. Ainsi, pour chaque étape clé du Big Data, nous posons le problème des VANET.

Notre seconde contribution est l'extraction des caractéristiques liées aux VANET afin d'obtenir des données provenant de ceux-ci. Pour ce faire, nous discutons de comment établir des scénarios de tests, et comment émuler un environnement afin, dans un premier temps, de tester une implémentation dans un environnement contrôlé, avant de pouvoir effectuer des tests dans un environnement réel, afin d'obtenir de vraies données provenant des VANET.

Pour notre troisième contribution, nous proposons une approche originale de la modélisation du comportement de conducteur. Cette approche est basée sur un algorithme permettant d'extraire des représentants d'une population, appelés exemplaires, en utilisant un concept de densité locale dans un voisinage.

Table des matières

Introduction en français

Contexte

Ces dernières années, le volume de données a augmenté substantiellement et dans des domaines de plus en plus variés [1], et plus particulièrement, dans les véhicules provenant des systèmes de tranports intelligents (ITS pour *Intelligent Transport Systems*) coopératifs (C-ITS pour *Cooperative ITS*). Nous observons l'émergence des véhicules connectés qui génèreront de grandes quantités de données dès demain [2, 3]. Dans les réseaux véhiculaires (VANET pour *Vehicular Ad-Hoc Networks*), les données sont générées aussi bien par l'infrastructure *via* les unités de bord de route (UBR ou RSU pour *Road Side Unit*) que par les véhicules eux-mêmes *via* les unités embarquées dans les véhicules (UEV ou OBU pour *On-Board Unit*). En Europe, les UBR et les UEV communiquent entre eux en utilisant un Wi-Fi véhiculaire appelé ITS-G5 qui est basé sur le standard IEEE 802.11p [4]. Avec une prédiction de 35 % de la part du marché des véhicules qui se vendront en 2022 et des bénéfices avoisinant les 113 milliards d'euros [5, 6], le volume de données des ITS deviennent si massives, qu'elles peuvent généralement être considérées comme Big Data, et donc avoir les même problématiques. La topologie des VANET est hautement dynamique, ce qui est un défi important pour les mécanismes et algorithmes sous-jacent, et donc, les algorithmes classiques peuvent ne pas être efficace.

Contributions

Dans cette thèse, nous proposons deux contributions qui posent les questions sur ses deux problèmes.

La première contribution est la considération des VANET comme des Big Data, et donc, soulever les problématiques provenant des Big Data. Pour ce faire, nous regardons pour chaque étape clef des Big Data, de la génération des données à leurs analyses, quelles problématiques remontent les VANET.

La seconde contribution est le développement d'une architecture de tests, ainsi que l'implémentation des standards européens afin d'obtenir des données émulées provenant des VANET, mais également des données réelles, afin

de les analyser. Nous proposons aussi une nouvelle méthodologie afin d'extraire des comportements de conducteur types, et ainsi soulever les questions de vie privée.

Dans cette thèse, six publications furent faites :

1. Hacène Fouchal, Geoffrey Wilhelm, Emilien Bourdy, Marwane Ayaida. *"A testing framework for intelligent transport systems"*. Dans : Computers and Communication (ISCC), 2016 IEEE Symposium on. IEEE. 2016, pp. 180–184

2. Hacène Fouchal, Emilien Bourdy, Geoffrey Wilhelm, Marwane Ayaida. *"A framework for validation of cooperative intelligent transport systems"*. Dans : Global Communications Conference (GLOBECOM), 2016 IEEE. IEEE. 2016, pp. 1–6

3. Hacène Fouchal, Geoffrey Wilhelm, Emilien Bourdy, Marwane Ayaida. *"An Extended Tester for Cooperative Intelligent Transport Systems"*. Dans : International Conference on Innovations for Community Services. Springer. 2017, pp. 47–55

4. Hacène Fouchal, Emilien Bourdy, Geoffrey Wilhelm, Marwane Ayaida. *"A validation tool for cooperative intelligent transport systems"*. Dans : Journal of computational science 22 (2017), pp. 283–288

5. Emilien Bourdy, Kandaraj Piamrat, Michel Herbin, Hacène Fouchal. *"New Method for Selecting Exemplars Application to Roadway Experimentation"*. Dans : International Conference on Innovations for Community Services. Springer. 2018, pp. 75–84

6. Emilien Bourdy, Kandaraj Piamrat, Michel Herbin, Hacène Fouchal. *"New Method for Exemplar Selection and Application to VANET Experimentation"*. Accepté pour publication au Global Communications Conference (GLOBECOM), 2018 IEEE.

Plan de thèse

Cette thèse est organisée comme suit : premièrement, dans le chapitre 2, nous survolerons comment fonctionne les VANET dans la section 2.1 et nous verrons plus précisément la standardisation européenne dans la section 2.2 et enfin, nous rappellerons les bases des techniques de Big Data dans la section 2.3. Ensuite, dans le chapitre 3, nous faisons les corrélations entre VANET et Big Data dans la section 3.1, avant de parler des projets européens

joignant les Big Data aux VANET dans la section 3.2. Après cela, nous verrons une implémentation d'un environnement ITS dans le chapitre 4, avec le développement des standards européens dans la section 4.1, la mise en place des scénarios de tests sur table et sur route, utilisés dans le projet Scoop@f, avec des données émulées et réelles dans les sections 4.2 et 4.3, ce qui nous permet d'obtenir des données VANET à analyser. Ensuite, nous présenterons une nouvelle méthodologie afin d'analyser ces données dans le chapitre 5, une revue des méthodes dites de kNN dans la section 5.1, notre méthodologie dans la section 5.2, la compléxité et le temps d'exécution de cette méthodologie dans la section 5.3, et, enfin, un cas d'usage de notre méthodologie avec une réelle expérimentation provenant du projet InterCor dans la section 5.4. Finalement, nous conclurons dans le chapitre 6.

Synthèse de la thèse

Informations générales

Que sont les réseaux véhiculaires ?

Les réseaux véhiculaires (VANET pour *Vehicular Ad-hoc Networks*) sont une spécialisation des réseaux mobiles (MANET pour *Mobile Ad-hoc Networks*), utilisant des véhicules mobiles comme nœuds. Comme ces nœuds sont des véhicules, ils s'organisent par eux-mêmes, et ajoutent des problèmes liés à la grande mobilité de la topologie du réseau [7].

Les réseaux des systèmes de transport intelligent (ITS pour *Intelligent Transport System*) coopératifs (C-ITS pour *Cooperative ITS*) sont basés sur les communications V2X (véhicule à X) : véhicule à véhicule (V2I) et véhicule à infrastructure (V2I). L'infrastructure est composée d'unités de bord de route (UBR ou RSU pour *Road Side Unit*) connectées au centre de gestion de trafic à travers une station ITS centrale (ITS-C). En Europe, toutes les stations ITS (ITS-S) sont basées sur la pile ITS standardisée par l'European Telecommunications Standards Institute (ETSI). Les UBR (appelés ITS-R) et les véhicules (appelés ITS-V) communiquent *via* un réseau sans fil appelé ITS-G5, qui est basé sur le standard IEEE 802.11p [4].

Le standard européen

Le standard de l'ETSI défini plusieurs couches :

1. *Access* qui correspond à la couche physique et détermine la bande de fréquence utilisée ;

2. *Network & Transport (N&T)* qui correspond à l'adressage, au routage des paquets et à l'émission de Beacon [9] ;

3. *Facilities* qui correspond aux services avec, entre autres, les *Cooperative Awareness Message* (CAM) [11] et les Decentralized Environmental Notification Message (DENM) [12]. Les CAM sont comme des Beacon en plus détaillés, tandis que les DENM sont des messages d'événements, qui peuvent être générés par l'utilisateur ou automatiquement par la station (un freinage d'urgence par exemple) ;

4. *Application* qui correspond à la couche application, où, par exemple, se trouve la demande d'envoi et la notification d'événement à et par l'utilisateur ;

5. *Management* qui gère les capteurs et l'état général de l'ITS-S ;

6. *Security* qui gère l'intégrité, l'authenticité et la vie privée aux niveaux *Access*, *N&T* et *Facilities*.

L'ère du Big Data

L'une des premières définitions du Big Data fut proposée par la société Apache Hadoop [16] : les Big Data sont « les données qui ne peuvent être capturées, gérées et traitées par des ordinateurs classiques dans un cadre acceptable ». En 2001, Doug Laney définit les défis du Big Data avec le model des 3 V : l'accroissement du Volume, de la Vélocité et de la Variété [17].

En 2011, l'International Data Corporation (IDC) ajoute un quatrième V : la Valeur. Et enfin, en 2013, Yuri Demchenko *et al.* proposent un cinquième V : la Véracité [18].

Nous avons donc aujourd'hui des modèles pouvant aller jusque 5 V :

1. **Volume** → l'échelle des données croissent rapidement ;

2. **Vélocité** → la collection et l'analyse des données doivent être menés rapidement ;

3. **Variété** → l'hétérogénéité des données inclue des structures de données semi et non structurées ;

4. **Value** → des données rares mais de grandes valeurs, ou inversement, pleins de données, mais de valeur moindre ;

5. **Véracité** → les données peuvent être gênées par du bruit, des données abérentes, la diversité des points de collecte etc.

Il est à noter qu'il n'est pas nécessaire de faire face aux cinq problèmes pour faire du Big Data.

Chacune des étapes du Big Data vont avoir à faire face aux différents problèmes précédents (les 5 V) et sont [7] :

1. Génération des données ;

2. Acquisition des données ;

3. Pré-traitement des données ;

4. Acheminement des données ;
5. Stockage des données ;
6. Analyse des données.

Appliquer les technologies Big Data avec les VANET

L'intégration des VANET dans les 4 V

Les VANET s'intègrent dans les Big Data, car ils posent quatre des cinq problèmes à chacunes des étapes cléf du Big Data :

Variété Du fait des options dans les données générées par les VANET, les constructeurs implémentent les standards différement (par exemple, certains peuvent avoir accès au bus CAN et d'autres non) ;

Vélocité Avec une fréquence de 1 à 10 messages par seconde par véhicule, et pour chaque message, certaines données sont modifiés à chaque fois, comme la position, la date, la vitesse etc. ;

Véracité Toutes les données ont besoin d'être cohérentes avec la situation si nous voulons des C-ITS efficaces. De plus, l'intégrité et l'authenticité sont gérées au niveau de la couche sécurité ;

Volume Avec toutes les données générées par les messages et les capteurs, plus le nombre de véhicules, le volume de données croît rapidement.

Obtenir des données VANET

Afin d'obtenir des données VANET, nous avons développé la pile ITS de l'ETSI, et des scénarios de tests sur table nous donnant des données émulées, ainsi que sur route nous donnant des données réelles.

Développement d'une pile ITS ETSI

Le développement de la pile ITS de l'ETSI s'est axé autour de la bibliothèque libre Qt (prononcé **/kju:t/**), afin de profiter de sa grande bibliothèque multi-plateforme et de son système de signal et *slot*, facilitant grandement le développement.

Au niveau physique, la disposition du projet se fait de la même façon que le fonctionnement de SVN : chaque projet et sous-projet sont des répertoires. Cependant, au niveau logiciel, nous avons implémenté la pile ITS de façon à être au plus proche du standard ETSI : la couche Application communique avec l'IHM (interface homme-machine) *via* du Bluetooth, et en UDP avec avec la couche Facilities afin de pouvoir s'abstenir de la couche Application si besoin (dans le cadre de tests sur table par exemple). La couche Facilities utilise des méthodes de la couche N&T pour émettre et reçois sur une socket UDP car nous considérons le BTP comme un port IP sur l'adresse locale. La couche GeoNet permet d'émettre aussi bien sur l'interface ITS-G5 que sur du TCP pour l'hybridation. Enfin, la couche Management envoie des signaux aux couches connectées lors de modifications de l'état de l'ITS-S.

Pour plus de détails, la documentation complète de notre pile ITS est disponible sur `http://scoop.univ-reims.fr/ITS_documentation`.

Tests sur table

Une fois que la pile ITS est développée, il est nécessaire de la valider. Pour ce faire, il faut dans un premier temps le tester sur table avec un environnement contrôlé. Cet environnement contrôlé nous permet de valider les comportements principaux de l'ITS-S plus facilement que sur route. L'architecture de test sur table est composé de cinq parties :

1. *Test Runtime* est le logiciel qui exécute la suite de tests ;
2. *Abstract Protocol Tester* (APT) est l'implémentation qui simule les couches à tester ;
3. *Component* est l'ITS-S à tester. Il implémente un *UpperTester* qui, par le biais de primitives, permet d'émuler l'environnment ;
4. *Test Adapter* (TA) fournit le transport et la traduction des messages entre le système à tester et l'APT ;
5. *Codecs* définit les règles d'encodage et du décodage des messages par le TA.

Dans le projet Scoop@f, afin de tester la functionalité des ITS-S, l'UpperTester a dû être étendu. Nous avons ajouté plusieurs primitives afin d'émuler les différentes situation pour lesquels l'ITS-S doit émettre des DENM automatiquement.

Aussi, l'interfaçage entre les ITS-C et les ITS-R se fait en DATEX II, langage qui est très largement utilisé par les gestionnaires de la route. Nous avons donc développé une émulation de l'ITS-C permettant de tester les envois et réceptions de messages DATEX II par l'ITS-R.

Enfin le dernier type de tests fait sur table est l'intéropérabilité. Pour cela, deux solutions sont possibles :

1. toutes les ITS-S se réunissent et utilisent leur UpperTester afin d'émuler l'environnement, et un par un, une ITS-S émet des messages et les autres, par le biais de l'UpperTester notifient la bonne réception ou non des messages ;

2. les fabricants émulent l'émission de messages et enregistrent les PCAP correspondants, les envoient au testeur, et lorsqu'une autre ITS-S doit tester son implémentation, ces PCAP sont rejoués, en modifiant les dates et lieux afin d'être cohérent avec la situation présente.

Tests sur route

Pour faire des tests sur route, nous ne pouvons plus contrôler l'environnement, et donc, nous devons utiliser un système de log afin de connaître l'état de l'ITS-S à tout moment. Ces logs peuvent être utilisés aussi bien hors ligne qu'en ligne. En effet, dans le projet Scoop@f, les ITS-R permettent le transfert de fichier d'une ITS-V vers une ITS-C. Pour ce faire, les CAM ont étaient étendus en des CAM-I (CAM Infrastructure). Ces CAM-I définissent les différents services proposés par l'ITS-R. Actuellement, deux services sont définis, le transfert de fichiers de log et le téléchargement de nouveaux certificats et listes de certificats révoqués. Malheureusement, par des contraintes industrielles, la définition de ces CAM-I n'a pu être optimisée. Par conséquent, un CAM-I ne peut avoir qu'un seul service à la fois. L'ITS-R envoie donc plusieurs CAM-I par seconde pour différents services. De plus, toujours par contraintes industrielles, nous n'avons pas pu automatiser l'analyse des logs, car il n'y a aucun moyen *a priori* de connaître l'origine du log et donc de sa définition.

Pour pallier ces problèmes, nous proposons d'améliorer les CAM-I afin de pouvoir définir une liste de services, et une définition générique des logs, permettant en décodant l'en-tête de savoir le type de log présent dans le fichier.

Extraction de comportement de conducteur

Revue des techniques utilisant les k plus proches voisins

Les techniques de k plus proches voisins (kNN pour k *Nearest Neighbors*) sont des techniques classique utilisées depuis des décennies pour la classification et l'extraction d'exemplaires [75]. La première de ses techniques fut le k *Nearest Neighbors* [76]. Cette méthodologie est simple à utiliser : elle utilise une base de connaissance afin de déterminer la classe d'une donnée inconnue. Pour cela, nous définissons une distance entre les données de l'ensemble de base. Pour chacune des données de cette base, nous calculons sa distance avec la donnée non classée. Nous trions les données de la base par cette distance, et nous sélectionnons les k premières données. Nous comptons pour chaque classe combien de fois elles sont représentées dans cette sélection, celle qui est le plus représentée est la classe que nous choisissons pour notre nouvelle donnée. Ensuite, cette technique fut améliorée en ajoutant des poids dans la méthode de calcul de distance [76].

D'autres améliorations ont été faites dont une qui modélise les données afin de sélectionner les plus proches voisins [80], et une autre utilisée dans la reconnaissance faciale, cherche la donnée localement plus proche de notre donnée non classée [82].

Dans tous les cas, le choix de la valeur k dans les méthodes précédentes et déterminant pour le choix de la classification.

Une nouvelle méthodologie de k *Nearest Neighbors* utilisant la densité locale

Nous proposons une nouvelle méthodologie utilisant les k plus proches voisins afin d'extraire des exemplaires d'une base de données. Les exemplaires sont des données réelles permettant de représenter une catégorie, à la différence des prototypes qui sont des valeurs statistiques (par exemple, si on dit qu'une famille moyenne à 2,7 enfants, c'est un prototype, et les représentants sont les familles à 1 ,2 et 3 enfants). Notre méthodologie, contrairement aux autres, n'utilise pas de base de connaissance pour extraire ses représentants.

La méthodologie utilise des ensembles de données multi-dimensionnels normalisée entre 0 et 1. C'est-à-dire que chaque donnée à plusieurs variables. Pour chaque données et chaque variables, nous calculons les distances euclidiennes. Ensuite, pour chaque donnée, nous regardons ceux qui ont une distance inférieur à un seuil ($\frac{k}{n}$ avec n le nombre de données) pour chacune des variables. Notre premier exemplaire et la donnée ayant le plus grand

nombre de voisins avec ce seuil. Nous retirons de la base ce représentant ainsi que son voisinage, et nous continuons le processus jusqu'à ce que la base de données soit vide.

Comme cette méthodologie est ajoute une notion de densité locale du voisinage, nous l'avons appelé Fuldon pour *Fuldon Uses Local Density Of Neighborhood*.

Complexité du kNN utilisant la densité locale

Notre méthodologie à une complexité qui n'est pas très élevée. En effet, elle est composée d'une boucle qui sélectionne les exemplaires. La sélection des exemplaires est une succession de méthodes qui sont en $O(1)$, $O(n)$, $O(n^2)$ et $O(n^3)$. La complexité de notre algorithme est donc en $O(n^3)$.

L'algorithme a d'abord été développé sur R en utilisant la bibliothèque Shiny afin d'avoir des résultats graphique, mais l'exécution étant plutôt lente, nous avons l'avons implémenté en C++ afin de déterminer si c'est l'algorithme ou le langage qui peine. En C++, l'algorithme donnait de bons résultats. De plus, cette algorithme étant fortement parallélisable, nous avons fait une première parallélisation en parallélisant chacune des méthodes, mais le temps d'exécution était plus longue que la version séquentielle car les méthodes parallélisées s'exécutaient trop rapidement, et donc nous passions plus de temps à allouer les processus qu'à les exécuter. Une autre méthode de parallélisation fut donc implémentée, en parallélisant l'ensemble de la sélection d'exemplaire. Ainsi, avec la parallélisation et en utilisant huit cœurs, nous divisons par deux le temps d'exécution par rapport à la version séquentielle.

Expérimentation des VANET dans un monde réel

Notre méthodologie fut utilisée avec des données provenant du projet Scoop@f. Les premières données étaient celles d'une simulation de route glissante. Le véhicule roule et reçoit un DENM indiquant une route glissante. Le véhicule enregistre dans ses logs l'état du véhicule, 30 secondes avant et après l'événement. Nous avons récupéré ces logs et utilisé notre méthodologie dessus afin d'en extraire des représentants.

Ensuite, lors du second TestFest du projet InterCor [90], nous avons récupéré les positions des véhicules pendant leurs trajets. Connaissant le fabriquant de chaque données, nos données étaient classées, nous permettant d'utiliser la méthodologie de façon supervisée. Chaque exemplaire extrait de cette façon définit un comportement. Nous avons vu que même avec des

seuils élevés, les classes étaient peu mélangées, et par conséquent, les comportements des conducteurs sont distincts.

Conclusion de la thèse

Conclusion

Dans le chapitre 2, nous avons survolé les technologies VANET de façon générale dans un premier temps, puis de façon plus spécifique avec le standard ETSI.

Les VANET sont la base des C-ITS, et communiquent en V2X : véhicule à véhicule (V2V) et véhicule à infrastructure (V2I/I2V). En Europe, nous utilisons le standard ETSI appelé ITS-G5, qui est basé sur l'IEEE 802.11p.

Après ça, nous avons vu ce que sont globalement les Big Data, qui étaient d'abord définis par un modèle utilisant 3 V : Volume, Vélocité et Variété. Plus tard, deux autres V ont été ajoutés : Valeur puis Véracité. Le Volume concerne l'échelle des données. La Vélocité concerne le fait que les données évoluent rapidement, et donc, elles ne peuvent être traitées d'en un temps raisonnable. La Variété indique l'hétérogénéité des données. La Valeur peut concercer deux cas, soit nous avons des données rares, mais de grade valeurs, soit au contraire, nous avons beaucoup de données, mais de peu valeur. Enfin, la Véracité est liée à la cohérence des données.

Les Big Data peuvent être utilisés dans de nombreux domaines, comme par exemple, l'Internet des objets, les entreprises, les villes intelligentes etc.

Dans le chapitre 3, nous avons vu que les VANET ont besoin des Big Data, car elles respectent quatre des cinq V : la Variété, la Vélocité, la Véracité et le Volume. La Variété est liée aux différentes options dans les standards et générateurs de données. La Vélocité est intrinsèque au réseau et au nombre de messages différents envoyés par seconde par véhicule. La Véracité est nécessaire pour avoir des C-ITS efficaces, et la couche sécurité du protocole ajoute une autre dimensions à cette problématique. Enfin, le Volume est lié au nombre croissant de véhicules communiquant.

À cause de ces différentes problématiques, de nombreux projets européens utilisent les ITS avec des technologies issues du Big Data.

Dans le chapitre 4, nous avons vu une implémentation de la pile ETSI basée sur la technologie Qt, nous permettant de simplifier le développement, en particulier par l'apport du système de signal et *slot*. Notre implémentation est au plus proche du standard ETSI d'un point de vue logiciel.

Nous avons aussi vu comment valider cette implémentation d'abord sur table avec les tests écrits en TTCN-3, puis sur route en utilisant un système

de log. Le système de log utilisé dans le projet Scoop@f a des améliorations possibles, qui n'ont pu être faits dûs à des contraintes industrielles.

Dans le chapitre 5, nous avons vu une revue des méchanismes utilisant les k plus proches voisins. Ensuite nous avons proposé notre approche originale utilisant la densité locale des données afin de sélectionner des exemplaires sans avoir besoin d'utiliser de base de connaissance. Cet algorithme a une complexité de $O(N^3)$, et a été développé en R, puis en C++, et enfin, une version parallélisée fut développée.

Cet algorithme fut utilisé dans des cas réels de tests sur route, et nous avons pu en extraire des comportements distincts.

Perspectives

Dès demain, les VANET auront besoin de changements. Le premeir concerne la législation. En effet, avec le nombre croissant de véhicules communiquant, il est nécessaire de pouvoir garantir la vie privée (entre autres). Par la mise à l'échelle, les VANET devront s'adapter afin de pouvoir rester intéropérable. Pour ce faire, il est nécessaire d'utiliser des solutions agiles.

De plus, par notre méthodologie, nous arrivons à extraire des comportements distincts de conducteur, ce qui pose des problèmes de vie privée. En effet, même si le véhicule change de pseudonyme, son conducteur ne change pas son comportement, et, de fait, reste toujours traçable. Dans nos futurs travaux, nous allons expérimenter plus finement cette hypothèse qui pourrait remettre en cause la vie privée du conducteur.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Context

Over the past few years, the volume of data has increased tremendously and in a very large scale within various domains [1], particularly, in vehicles with cooperative Intelligent Transport Systems (ITS). We can observe the emergence of connected vehicles, which will generate huge amount of data in the near future [2, 3]. In Vehicular Ad-hoc Networks (VANET), the data are generated either from the infrastructure by the Road Side Unit (RSU), or from the vehicles themselves by the On-Board Unit (OBU). RSU and OBU communicate with each other using a vehicular Wi-Fi called ITS-G5 in Europe, based on the IEEE 802.11p standard [4]. With a prediction of 35 % of market share among vehicles that will be marketed in 2022 along with revenues of around 113 billion euros [5, 6], ITS data volume is becoming massive in such a way that they can generally be considered as Big Data, and they will have the same problem. The VANET topology is a highly dynamic topology, which is an important challenge for the underlying mechanisms and algorithms, and so classical algorithms may be inefficient.

1.2 Contributions

In this thesis, we propose two contributions to pose questions on these two problems.

The first contribution is the consideration of VANET as Big Data, and so, raise the issues that raise Big Data. To do that, we check for each steps of Big Data from data generation to data analysis, which issues VANET raise.

The second contribution is the development of a testing architecture and the European standard implementation to get emulated and real VANET data to analyze. We also wade a new methodology to extract representative driver's behavior, and raise privacy issues.

In this thesis, six publications was made:

1. Hacène Fouchal, Geoffrey Wilhelm, Emilien Bourdy, Marwane Ayaida. *"A testing framework for intelligent transport systems"*. In: Computers

and Communication (ISCC), 2016 IEEE Symposium on. IEEE. 2016, pp. 180–184

2. Hacène Fouchal, Emilien Bourdy, Geoffrey Wilhelm, Marwane Ayaida. *"A framework for validation of cooperative intelligent transport systems"*. In: Global Communications Conference (GLOBECOM), 2016 IEEE. IEEE. 2016, pp. 1–6

3. Hacène Fouchal, Geoffrey Wilhelm, Emilien Bourdy, Marwane Ayaida. *"An Extended Tester for Cooperative Intelligent Transport Systems"*. In: International Conference on Innovations for Community Services. Springer. 2017, pp. 47–55

4. Hacène Fouchal, Emilien Bourdy, Geoffrey Wilhelm, Marwane Ayaida. *"A validation tool for cooperative intelligent transport systems"*. In: Journal of computational science 22 (2017), pp. 283–288

5. Emilien Bourdy, Kandaraj Piamrat, Michel Herbin, Hacène Fouchal. *"New Method for Selecting Exemplars Application to Roadway Experimentation"*. In: International Conference on Innovations for Community Services. Springer. 2018, pp. 75–84

6. Emilien Bourdy, Kandaraj Piamrat, Michel Herbin, Hacène Fouchal. *"New Method for Exemplar Selection and Application to VANET Experimentation"*. Accepted for publication in Global Communications Conference (GLOBECOM), 2018 IEEE.

1.3 Thesis Outline

This thesis is organized as follows: first of all, in Chapter 2, we will see an overview of the communication between vehicles in Section 2.1, and then we will focus on the European standardization in Section 2.2, and we will remind the basics of Big Data techniques in Section 2.3. Then, in Chapter 3, we will see the correlations between VANET and Big Data in Section 3.1, before talking about some European projects joining Big Data and VANET in Section 3.2. After that, we will see an implemented architecture in Chapter 4, with an example of an ITS stack development in Section 4.1, laboratory tests and road tests scenarios, used in the Scoop@f project, with emulated and real VANET data in Sections 4.2 and 4.3, which enable us to get VANET data to analyze. After that we will present a methodology to analyze them in Chapter 5, a review of the kNN methods in Section 5.1, our methodology in Section 5.2, the complexity and time to execute the methodology in

Section 5.3, and finally, a use case of the methodology with a real experimentation from the InterCor project in Section 5.4. Finally, we will conclude in Chapter 6.

Chapter 2
Background

Before starting with the integration of Big Data in VANET, we have to see how works the VANET communication in Section 2.1, and particularly the European standard in Section 2.2. After that, we will see an overview of the basics of Big Data in Section 2.3.

2.1 What is Vehciular Ad-hoc Networks?

Vehicular Ad-hoc Networks (VANET) are a specification of Mobile Ad-hoc Networks (MANET), using moving vehicles as network nodes. Since nodes are vehicles, they are self-organized, and add issues linked to the high mobility of the network topology [8].

Cooperative Intelligent Transport Systems (C-ITS) networks are based on the V2X (Vehicular to X) communications: Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) communications. The infrastructure is composed of Road Side Units (RSU) connected to the traffic management center through a central ITS station (ITS-C). In Europe, all the ITS stations are based on the European Telecommunications Standards Institute (ETSI) ITS stack. RSUs (called ITS-R) and vehicles (called ITS-V) will communicate through wireless networks (access layer) based on ITS-G5 network (based on IEEE 802.11p standard). The application layer uses mainly the following standard protocols (see Figure 2.1):

- Access layer corresponds to the Physique and Data Link layers from the OSI stack (i.e. an Ethernet layer using the *EtherType 0x8947* on a 5.9 GHz band [9]);
- GeoNetworking corresponds to the Network layer from the OSI stack, where we define the routing method of the packet (Section 2.2.1);
- Payload, which is optional, corresponds to the Transport and Session layer with the Basic Transport Protocol (Section 2.2.2), and Presentation and Application layer, with the payload encoding and the Facilities layer (Section 2.2.3):

  - Cooperative Awareness Messages (CAM) (Section 2.2.3.1) give continuous vehicle information, positions to all neighbor stations located within a single hop distance.
  - Decentralized Environmental Notification Messages (DENM) (Section 2.2.3.2) are mainly used by the Cooperative Road Hazard Warning (RHW) application in order to send information about particular events (as accidents, heavy raining, ...)

- Application layer is related to usual applications embedded by ITS components. For example, the following applications could be implemented:
  - In-vehicle signaling: it shows driving head, static signaling, dynamics of speed.
  - Data collection: vehicle data (position, speed, direction, cape), road event data input manually by driver (animal on the road, etc.), automatic data (impact, emergency brake, ...)
  - Road hazard signaling, unexpected and dangerous events: alerts from the European directive (temporary slippery road, pedestrian on the road, reduced visibility, ...)
  - Information about the road traffic: traffic light color, journey time, recommended route, access to services...
  - Parks relay and multi modal system: location and availability of parking relays, schedules of public transportation.

2.2 The European Standard

In Europe, VANET use the European Telecommunications Standards Institute (ETSI) standardization, which defines the communication protocol, based on the IEEE 802.11p, called ITS-G5.

2.2.1 GeoNetworking

The GeoNetworking layer is the routing layer of the ETSI ITS-G5 stack [10]. This standard defines different headers used by the Networking & Transport (N&T) layer. These headers determine how the packet will be routed (the corresponding figures are available in Appendix A). We have three headers:

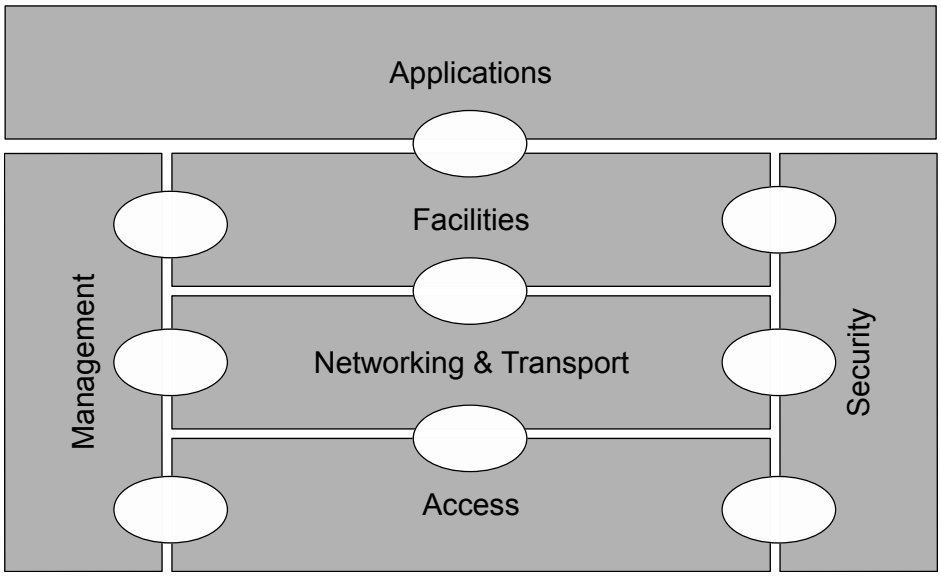


Figure 2.1 – GeoNetworking layers

1. *Basic Header*, which is used to get the packet's Time To Live (TTL) / lifetime and type of packet (secured or not).

2. *Common Header*, which determines the type of payload (BTP, IPv6, none), the type of the *Extended Header*, the traffic class for messages prioritization, flag to know if the station is a mobile station or not, payload length (if applicable), and the initial TTL.

3. *Extended Header*, which contains the routing information. This layer depends on the situation and is one of the eight following headers:

   (a) *Beacon*. This header is used with no payload and is composed of the ITS station (ITS-S) position.

   (b) *GeoUnicast* (GUC). This header is used to send packet to a specific ITS-S. This header is composed of a sequence number, the source and destination positions.

   (c) *Topologically-Scoped Broadcast* (TSB). This header is used to send packet to neighbors with multi-hop to extend the neighborhood. This header is composed of a sequence number and the source position.

   (d) *Single-Hop Broadcast* (SHB). This header is used to send packet to the neighborhood, without multi-hopping. This header is of the same size as TSB, but without sequence number and so is only

composed of the source position (plus a reserved field to align the size with the TSB).

(e) *GeoBroadcast* (GBC). This header is used to send packet to an area and broadcast it inside the area. This header is composed of a sequence number, the source position, the position of the center of the area, the dimensions of the area, and the angle in degrees from North (for non-circular area).

(f) *GeoAnycast* (GAC). This header is used to send packet to an area, but unlike GBC, it is not rebroadcasted inside the area. This header has the same format as the GBC.

(g) *Location Service* (LS). This header is used to update the neighborhood table of the ITS-S. When requesting for an ITS-S position, it is composed of a sequence number, the source position and the requested GN-ADDR, and when replying, of a sequence number, the source position and the destination position.

2.2.1.1 Packet handling

When the ITS-S is the generating station of the packet, it fills the headers depending on the requests needs. It uses the routing algorithm corresponding to the Extended Header to fill the destination address in the Ethernet Header, and send the packet to the ITS-G5 interface, and add it to its different buffers.

When the ITS-S is receiving a packet, it checks if there is a duplicate packet (DPD), or if it has the same GeoNetworking Address (DAD). If the GeoNetworking Address of the source is the same as the receiving ITS-S, it changes its GeoNetworking Address and discard the packet. If needed, the ITS-S forward it using the algorithm corresponding to the Extended Header, and add it in its different buffers. If the packet has a payload, it sends it to the upper layer.

2.2.1.2 Packet Routing

When we use packets with multi-hops to a destination (area or ITS-S), GeoNetworking provides some algorithms to route them:

1. Greedy Forwarding (GF) algorithm for GUC
2. Contention-Based Forwarding (CBF) algorithm for GUC
3. Simple Forwarding algorithm for GBC
4. CBF algorithm for GBC

5. Advanced Forwarding algorithm for GBC

- **Greedy Forwarding Algorithm for GUC**

  The Greedy Forwarding (GF) algorithm is used by GUC packets to determine the neighbor used for the next hop. This algorithm applies the most forward within radius (MFR) policy, which selects the neighbor with the smallest distance to the destination, thus providing the greatest progress when the GUC packet is forwarded. The GF is also used to determine the next hop with GBC and GAC packets.

- **Contention-Based Forwarding Algorithm for GUC**

  Unlike the GF algorithm, the Contention-Based Forwarding Algorithm (CBF) utilizes timer-based rebroadcasting with overhearing of duplicates in order to enable an implicit forwarding of a packet by the optimal node. With CBF, the GeoAdhoc router broadcasts the GUC packet. All neighbors, which receive the packet, process it: those routers with a positive progress buffer the packet and start a timer with a timeout that is inversely proportional to the forwarding progress of the GeoAdhoc router (more details in Appendix B). Upon expiration of the timer, the GeoAdhoc router rebroadcasts the GUC packet. Before the timer expires, the GeoAdhoc router may receive a duplicate of the packet from a GeoAdhoc router with a shorter timeout, i.e. with a smaller distance to the destination. In this case, the GeoAdhoc router inspects its packet buffer, stops the timer and removes the GUC packet from the packet buffer.

- **Simple Forwarding Algorithm for GBC**

  This algorithm utilizes a function F(x,y), described in Appendix C, in order to determine whether the GeoAdhoc router is located inside, at the border or outside the area. In the simple forwarding algorithm, if the ITS-S is inside or at the border of the area, the packet shall be rebroadcasted. If it is outside the area, the packet shall be forwarded be the previous GF algorithm.

- **Contention-Based Forwarding Algorithm for GBC**

  As CBF for GUC, CBF for GBC uses a timer proportional inversely to the distance with the destination area to avoid collisions and so the station, which rebroadcast the packet, is the nearest station of the destination.

- **Advanced Forwarding Algorithm for GBC**

  The advanced algorithm utilizes the GF and CBF mechanisms and adds a sectorial contention area. CBF is used to deal with uncertainties. In

order to minimize the additional forwarding delay introduced by CBF: upon reception of the packet, the next hop — in case of correct reception — forwards the message immediately. The CBF efficiency is improved by choosing potential forwarders only from a specific sector; i.e. GeoAdhoc routers located inside the sector (see Figure 2.2) refrain from retransmission of the packet (sectorial backfire). The reliability is increased by a controlled packet retransmission scheme within the destination area.

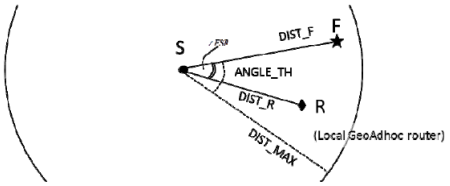


Figure 2.2 – Sectorial contention area [10]

For the sectorial backfire, the algorithm is described in Appendix C. The controlled packet retransmission scheme is a counter, which is increased every time the packet is received. When the number of retransmissions for this packet reaches a threshold, the router stops contending for the packet. In practice, the threshold is set to 3.

2.2.2 Basic Transport Protocol

The Basic Transport Protocol (BTP) [11] (which make part of the GN payload) has the same utility as the classical TCP/IP port: when we receive a GN packet, we only know its routing method (i.e. SHB, GBC, GUC etc.). But the routing methodology do not talk about the payload type and the service behind. To do this, we use the BTP, which consists of a pair of two ports. If we use the old BTP-A, the first port is the destination port and the second the source port. It's used when we need an interactivity between stations, but in practice, this kind of BTP is not used anymore. The second type of BTP (the BTP-B), is composed of the destination port and a second port to precise the type of the service if needed. For the moment, the second port is always set to 0. Table 2.1 summarizes the defined values in ETSI for the BTP destination port.

2.2.3 Facilities Layer

The Facilities layer is the layer, which communicates with the application layer (e.g. HMI or automatic event trigger application) and defines the GN

Service	BTP destination port
CAM	2001
DENM	2002
MAP	2003
SPAT	2004
SAM	2005

Table 2.1 – List of well-known BTP port numbers [11]

payload. Here, we will only focus on two types of messages: the Cooperative Awareness Message (CAM) [12] and the Decentralized Environmental Notification Message (DENM) [13], since they are used for safety and cooperative purposes, and the only ones used in the Scoop@f project [14].

2.2.3.1 Cooperative Awareness Message

The CAM [12] is a Beacon in Facilities layer, which is more precise than simple Beacon. As illustrated in Figure 2.3, CAM is composed of five elements:

1. *ItsPduHeader*: general header, which indicate type of message, protocol version and station identifier,
2. *BasicContainer*: generation time and ITS station (ITS-S) position,
3. *HighFrequencyContainer*: containing data, which evolve quickly for vehicle (speed, acceleration etc.) and list of protected communication zone for road side unit like tolling zone,
4. *LowFrequencyContainer*: containing data, which evolve slowly for vehicle (exterior lights, path history and vehicle role for vehicle and tolling zone information for RSU).
5. *SpecialVechicleContainer*: containing data depending on vehicle's role to specify how vehicle is a special vehicle (light bar and siren for example).

- **Sending operation**

  The CAM generation is controlled by the Cooperative Awareness (CA) Basic Service (CABS). The CABS works as follows (see Appendix D for more details):

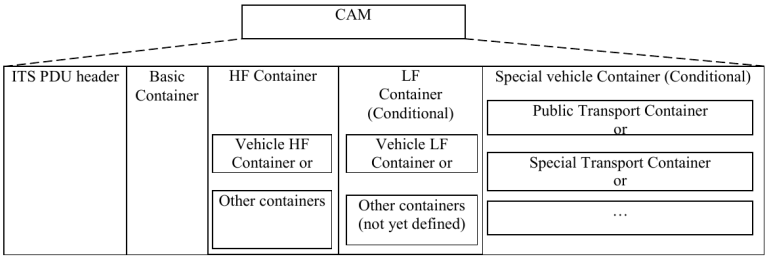


Figure 2.3 – CAM structure [12]

1. Check if there is enough time (100 ms) from the last generated CAM.
2. If yes, it looks if the ITS-S position evolve enough, or if the last generated CAM was late (1 s) to notify potential neighbors.
3. If yes, it fills the *ItsPduHeader*, the *BasicContainer* and the *HighFrequencyContainer*, and if the last *LowFrequencyContainer* filled was at least 500 ms ago, it fills it
4. When the CAM is filled, it is encoded in ASN.1 UPER and sent to the N&T layer.

- **Reception operation**

  The CAM reception is controlled by the CA Reception Management, which executes the following operations:

  1. Decode received CAM
  2. Make CAM data available by e.g. passing to the ITS application layer or to the Local Dynamic Map (LDM)
  3. End of operation, wait for the next CAM reception

2.2.3.2 Decentralized Environmental Notification Message

The Decentralized Environmental Notification Message (DENM [13]) is used to warn about events and is composed of five elements as presented in Figure 2.4:

1. *ItsPduHeader*: it is the same as for CAM, but with the message type set to DENM,
2. *ManagementContainer*: containing event position, validity duration, etc.

3. *SituationContainer*: containing information about the type of event, information quality of the event (from 1 to 7, with 1 for very poor quality and 7 for the most reliable information about the event), etc.

4. *LocationContainer*: containing more information about the location of the event (paths to go to the event, type of road, etc.)

5. *AlacarteContainer*: containing more information depending on the use case.

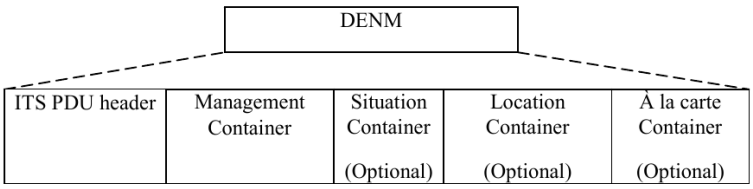


Figure 2.4 – DENM structure [13]

The DENM can be used in three situations:

1. as a source: we generate a DENM and send it,

2. as a receiver: we receive a DENM and notify the application layer if necessary,

3. as a forwarder: we receive a DENM, and we forward it to stations, which could not receive it before.

All the algorithms used for these situations are available in Appendix E.

- **Source operation**

  When the ITS-S is as a source ITS-S, it has three types of operation:

  1. *AppDENM_trigger* to create a new DENM;
  2. *AppDENM_update* to modify a previous DENM;
  3. *AppDENM_termination* to terminate a previous DENM.

  – *AppDENM_trigger*

  When the DEN basic service receives a request of type *AppDENM_trigger*, it shall execute the following operations:

1. It checks that the DENM is not expired.
2. It fills the DENM by assigning an unused identifier, and fills all container with values from the request.
3. It encodes the DENM in ASN.1 UPER and send it to the N&T layer.
4. It creates an entry in the originating message table and insert the DENM.
5. If needed, starts timers, and notify the Application layer of the success or failure of DENM generation and sending.

– *AppDENM_update*

When the DEN basic service receives a request of type *AppDENM_update*, it shall execute the following operations:

1. It checks that the DENM is not expired.
2. It checks that the DENM is in its originating message table.
3. It stops potential timers.
4. It changes the values of the DENM as asked by the request.
5. It encodes the new DENM in ASN.1 UPER and send it to the N&T.
6. It updates the entry in the originating message table.
7. If needed, starts timers, and notify the Application layer of the success or failure of DENM update and sending.

– *AppDENM_termination*

When the DEN basic service receives a request of type *AppDENM_termination* it shall execute the following operations:

1. It checks that the DENM is not expired.
2. It checks that the DENM is in its originating message table or in the receiving message table.
3. It stops potential timers.
4. It changes the values of the DENM as asked by the request. The termination value is set to *isCancellation* if it is the originating of the DENM, and set to *isNegation* otherwise.
5. It encodes the new DENM in ASN.1 UPER and send it to the N&T.
6. It updates the entry in the originating message table.
7. If needed, starts timers, and notify the Application layer of the success or failure of DENM termination and sending.

- **Receiving operation**

The DENM reception is controlled by the DEN Reception Management, which executes the following operations:

1. Decode the received DENM
2. Check that the DENM is not expired.
3. If the DENM is a termination DENM, end the corresponding DENM (if exists in one of the ITS-S tables).
4. Update the receiving message table.
5. Start the timer.
6. Notify the Application layer from receiving DENM.
7. If needed, start the Keep-Alive Forwarding (KAF).

- **Forwarding operation**

  When an ITS-S receives a DENM, it can perform a Keep-Alive Forwarding (KAF) to forward the DENM when there is no neighbor to receive the original.

  The KAF works only if there is the *transimissionInterval* set in the received DENM and as follows:

1. Check if the DENM is a new or an updated DENM by using the forwarding message table.
2. Stop the potential timers.
3. Calculate the needed timers.
4. Change the *ItsPduHeader* with the ITS-S identifier.
5. Start the set timers.
6. Reconstruct the DENM with the ASN.1 UPER.
7. Add or update the forwarding message table.

Technologies evolution permits us getting and threat more and more of data. VANET technologies will face these evolution that we need to anticipate, and so, VANET are also concerned by Big Data.

2.3 The Era of Big Data

Before starting with Big Data, it is important to remind some laws that govern the technological evolutions. The main laws are the Moore's and Kryder's laws, about conjecture on microprocessors "power" and memory available on computer respectively:

1. "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly, over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years." [15];

2. The number of microprocessor's transistors on a silicon wafer double each two years [16].

3. "Inside of a decade and a half, hard disks had increased their capacity 1,000-fold, a rate that Intel founder Gordon Moore himself has called "flabbergasting"." [17]

But these conjectures have physical limits: frequency and high temperature for microprocessor and difficulty for finer hard drive spaces to remain stable. The microprocessor limit is avoided by the parallelization of chipsets. However, when we use a very large-scale of data, even with parallelization, it is still difficult to manage a lot of data with classical computers and hard drives. That's what we call Big Data.

One of the first definition of Big Data is a definition proposed by Apache Hadoop [18]: Big Data are "datasets which could not be captured, managed, and processed by general computers with an acceptable scope". In 2001, Doug Laney defined challenges of Big Data with a 3Vs model: Volume, Velocity, and Variety increasing [19].

Volume the data scale becomes increasingly big;

Velocity data collection and analysis must be rapidly and timely conducted;

Variety heterogeneous of data including structured and semi- / unstructured data (see Section 2.3.5).

In 2011, the International Data Corporation (IDC) defined Big Data as "a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling the high-velocity capture, discovery, and analysis" [1]. They add a fourth V: Value.

Value huge value but low density, and inversely, few value, but with high density.

In 2013, Yuri Demchenko et al. proposed a fifth V: Veracity [20].

Veracity of data can be threatened by noises, irrelevant data, diversity of collection points, etc.

Figure 2.5 summarizes the 5V concept, but when we talk about Big Data, it is not necessary to have all the 5V. We use the three basic properties (Volume, Velocity, Variety), and if needed, we can increase the Big Data properties by adding the two other V [21]: the technological choices are oriented by these properties of big data.

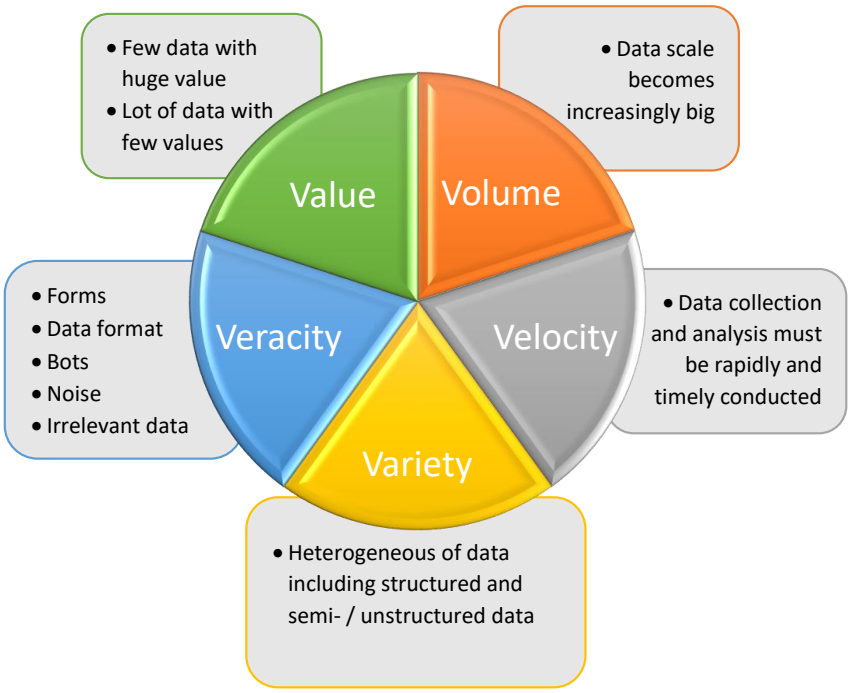


Figure 2.5 – The 5V of Big Data

With Big Data, there are some steps from getting the data to analyze them [7]:

1. Data generation: which kind of data are "Big Data"? We will see three main type of data which can be considered as Big Data, the **Enterprise** data, the **Internet of Things** data, and the **Social Network** data.

2. Data acquisition: how can we gather the generated data? For example, the equipment can gather data explicitly (with log files for example), or we can gather them without their knowledge by sniffing the network.

3. Data preprocessing: how can we reduce the gathered data volume or redundancy?

4. Data transportation: how can we transport data from the equipment to the server, which will store and analyze them. We will define here the three main layers of Big Data transportation.

5. Data storage: how can we technically store Big Data, depending on the type of data?

6. Data analysis: how can we analyze data?

After that, we will see some application examples using Big Data technologies (see Section 2.3.7).

2.3.1 Data Generation

2.3.1.1 Enterprise Data

The main sources of Big Data are the enterprises [22], and it is estimated that the business data volume of all company in the world may double every 1.2 years [23]. For example, Amazon processes millions of terminal operations and more than 500.000 queries from third-party sellers per day [19], Walmart processes one million customer trades per hour with a database with a capacity of over 2.5 PB [24] and Akamai analyzes 75 million events per day for its target advertisements [25], and all these transactions need to be secured. With the Enterprise data, we tackle the Volume, the Velocity, the Variety and the Veracity.

2.3.1.2 IoT Data

The Internet of Things (IoT) is an important source of Big Data. IoT consists of all objects that can be connected, it goes from mirror to watch, smart home to traffic light and so come from many sources (agriculture, families, medical care, traffic). Data generated by IoT has the following features [7]:

Large-scale with the number of connected objects and their data acquisition: location, temperature, multimedia;

Heterogeneity with the variety of connected objects;

Strong time and space correlation for statistical purposes, we need a good time and location precision;

Effective data accounts for only a small portion of the Big Data a great quantity of noises may occur during the acquisition and transmission of data.

2.3.1.3 Social Network Data

Social Network is used increasingly and generates a huge amount of data. In January 2018, Facebook had 2 196 million active users, YouTube 1 900 million, Instagram 1 000 million, and these Social Network continue to grow: plus 527 million of users for Facebook or 400 million for Instagram [26, 27]. Social Network can be considered as a graph with everything such as a user, a book is a node, and the edges are the relation between them [28]. That growing number of users impose big security and privacy challenges.

2.3.2 Data Acquisition

Once data are generated, we need to get them, and there are many ways to collect them [7] (each part concerns properties of the 5V model we use).

Log files one of the widely used data, log files are adhoc files where we will record everything we need, from the click position to the number of visitors of a website. Log files will massively be in the ASCII text format, but databases can also be used;

Sensing they are common in daily life to measure physical quantities and transform them into readable data. Sensed data is transferred to a data collection point through the network (wired or not);

Methods for acquiring network data the main method to acquire network data is the use of a Web crawler, which download all the content of a website (not only the visited page) [29];

Libpcap-based packet capture technology widely used with network data, this tool sniffs all data on selected network interfaces and capture them. The first problem with Libpcap-based capture is the considerable packet losses, which may occur under a high-speed network environment;

Zero-copy packet capture technology is the same method as Libpcap, but without copy between internal memories and so packet arrives directly to the terminal and there is no packet loss under a high-speed network environment;

Mobile equipment with the huge amount of smartphone, they can be used to acquire customer data (geographical position, pictures, video).

2.3.3 Data Preprocessing

Because of the variety of data sources, datasets may have many problems of noises, redundancy, or consistency. To avoid them, we can use some preprocessing techniques.

Integration which consists of an aggregation of data from many sources into a single one [30];

Cleaning which consists of find inaccurate, incomplete, or unreasonable data and then modify or delete them. Generally, there are five complementary procedures [31]:

1. Defining and determining error types,
2. Searching and identifying errors,
3. Correcting errors,
4. Documenting error examples and error types,
5. Modifying data entry procedures to reduces future errors.

Redundancy elimination to reduce the volume of data, we can eliminate useless redundancies.

But all these preprocessing techniques need to guarantee the data integrity, reliability, and Veracity. The preprocessing will so act principally on Volume, Variety and Veracity.

2.3.4 Data Transportation

To transport data from the source to the Data Center (DC), which will store them, we have three layers [32] summarized in Figure 2.6:

1. Access network is directly connected to end devices, such as personal computers, mobile devices. It's composed of gateways, Wi-Fi access point, or cellular antenna;

2. Internet backbone, which is the network to transmit data from the Access network to the Data Center Network (DCN). To do this, there are many approaches depending on the application, for example, *mPath* avoids bottleneck and is used for an end-to-end transmission [33] and Wittie et al. reduces the service latency for social networks [34];

3. Inter- and intra- DC networks are the networks between DCN (inter-DCN) and inside DCN (intra-DCN). As for the Internet backbone, many approaches are used depending on the application. For example, for inter-DCN, we have NetStitcher, which improve the network utilization for data backup and migration [35] and Jetway minimizes the link cost for video delivery [36]. For the intra-DCN, we can cite Orchestra, which reduce the job duration with a MapReduce and Dryad (see Section 2.3.5) utilization [37], or a 3D beamforming with mirror on the ceiling and antenna on DC: a laser from a DC is sent to another DC by reflection on the mirror [38].

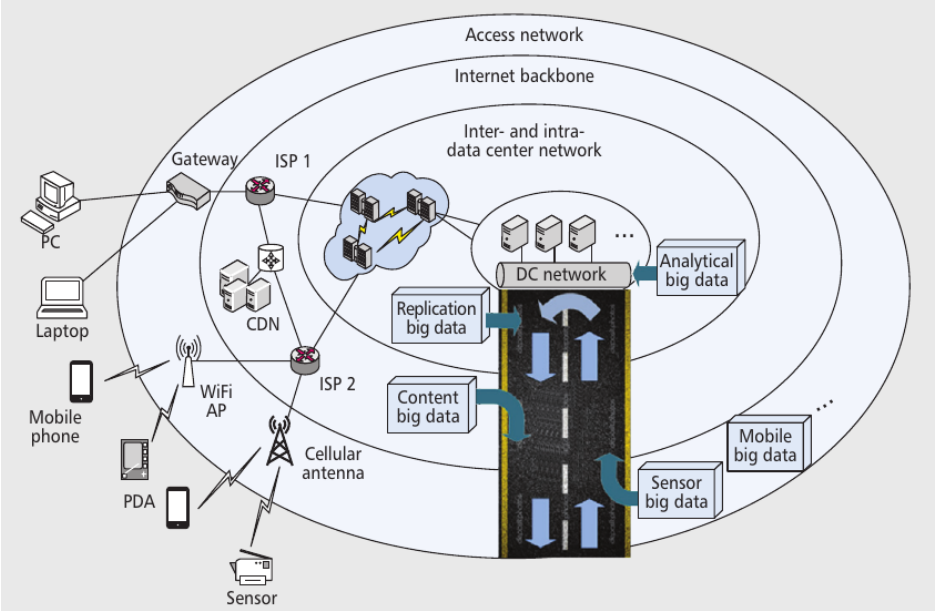


Figure 2.6 – Three-layered network architecture from the perspective of Big Data applications [32]

For security reasons, and to be fault-tolerant, we need to use decentralized communications, and redundancy [39]. We still have the Volume (by using numbers of DC), the Velocity (with the needs of fast communications, and fast ways to change data inside DC), and the Veracity (with all the security problematic in the communication).

2.3.5 Data Storage

When we talk about storage, we need to invoke the three types of data:

1. Structured data, the simple way of data, which are used with classical databases;
2. Semi-structured data, which are more complex with some structured and unstructured part like web pages;
3. Unstructured data, which are the most complex because there is no structure inside the data like video or audio.

The first one is usable with structured RDBMSs (Relational DataBase Management System), and the two others will need a NoSQL (Not only SQL) to use them. Note that NoSQL techniques can also be used with structured data.

To store Big Data databases, we will have many solutions depending on the problem [7].

Key-value Databases are constituted by a simple data model and every key is unique and customers may input queries values according to the keys. We can cite *Dynamo* [40] used in the Amazon e-commerce Platform or *Voldemort* [41] used by LinkedIn;

Column-oriented Databases which store and process data according to columns other than rows. We can cite Google's *BigTable* [42] which is the principal inspiration of all column-oriented databases or *Cassandra* [43], which is a distributed storage system to manage the huge amount of structured data distributed among multiple commercial servers;

Document Databases in these databases, we can store more complex data forms. We can cite the open-source *MongoDB* [44], which stores documents as Binary JSON (BSON) or the Apache *CoucheDB* [45] written in Erlang, allowing to manage the concurrency, real time and distributed in an easy way.

With Big Data storage, we have Volume, Variety, Velocity and Veracity challenges: the Volume is intrinsic, the Variety is made with the different types of data structuration, the Velocity with the fast data changes, and the Veracity due to the possible redundancy of servers, which need to have coherent values.

2.3.6 Data Analysis

To analyze Big Data databases, we will need new methodologies and technologies to quickly extract key information from massive data. We call to mind the main technologies that can be used in this context of Big Data:

Hashing is a method that essentially transform data into shorter fixed-length numerical values or index values;

Bloom Filter consists of a series of Hash function;

Index is always an effective method, but has a disadvantage with the additional cost for storing index files, which should be maintained dynamically when data is updated;

Triel also called trie tree, a variant of Hash Tree, is mainly applied to rapid retrieval and word frequency statistics.

Traditional parallel models may not be adequate to support such large-scale parallel programs, and so we can use some other models:

MapReduce is a computing model with only two functions: Map and Reduce. The Map function processes input key-value pairs and generates intermediate key-value pairs. Then, the Reduce function will take the intermediate key to compress the value set into a smaller set.

Dryad is a directed acyclic graph, in which vertexes represent programs and edges the data channels. Dryad execute operations on the vertexes in clusters and transmits data via data channels.

All-Pairs is a system designed for biometrics, bio-informatics and data mining applications. It focuses on comparing element pairs in two datasets by a given function.

Pregel from Google, which facilitates the processing of large-sized graph like analysis of network graphs and social networking services.

All these analyses techniques are also used for another purposes:

Data mining: the extraction of the hidden predictive information from large databases. It scours databases for hidden patterns, finding predictive information that experts may miss, as it goes beyond their expectation [46].

Deep Learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts [47].

Machine Learning is used to identify objects in images, transcribe speech into text, match news items, posts or products with users' interests, and select relevant results of search [48].

Decision-Making is used to help users to make the smarter purchasing decision than human can do, when it is too complex, due to the amount of data [49].

Increasingly, applications using Machine Learning make use of Deep Learning [48], whereas both Deep and Machine Learning will often make use of Data Mining [50]. To use Decision-Making, we will also use the three others analyze techniques.

For the same reasons as for Big Data storage, we have here the Volume, Variety, Velocity and Veracity challenges.

2.3.7 Application of Big Data

Big Data is used in as many domains as types of data. Here, we will see some applications using the technologies of Big Data.

2.3.7.1 Application of Big Data in Enterprises

The application of Big Data in enterprises can enhance their production efficiency and competitiveness in many aspects: in marketing, on sales planning, operation, and supply chain for example. In marketing, enterprises can more accurately predict the consumer behavior and find new business modes. On sales planning, enterprises can optimize their commodity prices. On operation, enterprises can optimize their operation efficiency and satisfaction: optimize avoid excess production capacity, and reduce labor cost. On supply chain, they can mitigate the gap between supply and demand, control budgets and improve services by conduct inventory and logistic optimization.

2.3.7.2 Application of IoT Based Big Data

The application of IoT based Big Data can come from various domains due to the variety of objects, for example, truck can be equipped with sensors to be supervised, and to manage employees, optimize delivery routes, and prevent engine failures. In smart cities, IoT and Big Data can be used for smart parking, crosswalk and energy usage [51], smart metering of temperature, traffic intensity [52], smart transportation, smart energy and water management [53].

2.3.7.3 Application of Online Social Network-Oriented Big Data

Big Data of online social network sites (SNS) mainly comes from instant messages, online social, micro blog, and shared space, which represents various

user activities, and can be used in many domains: relationship between users (Facebook, LinkedIn), "newsletter" (Twitter), as a "TV show" (YouTube), or in transportation (Waze). Classic applications of Big Data from online SNS mainly mine and analyze content information and structural information to acquire values [54]:

Content-based Applications language and text are the two most important forms of presentation in SNS. Through the analysis of language and text, user preference, emotion, interest, and demand, may revealed [55];

Structured-based Applications in SNS, users are represented as nodes while social relation, interest, and hobbies, aggregate relations among users into a clustered structure [56].

Application of Big Data from online SNS may help to better understand user's behavior in the following three aspects:

1. *Early Warning* to rapidly cope with crises if any by detecting abnormalities in the usage of electronic devices and services;

2. *Real-time Monitoring* to provide accurate information for the formulation of policies and plans by monitoring the current behavior, emotion, and preference of users;

3. *Real-time Feedback* acquire groups' feedback against some social activities based on real-time monitoring.

2.3.7.4 Collective Intelligence

With the rapid development of wireless communication and sensor technologies, crowd sensing is becoming a key issue of mobile computing. Crowd sensing in the form of Crowdsourcing has been successfully applied to geotagged photograph, positioning and navigation, urban road traffic sensing, market forecast, opinion mining, and other labor-intensive applications. In the Big Data era, Spatial Crowdsoursing is a hot topic. A user may request the service and resources related to a specific location. Then the mobile users who are willing to participate in the task will move to the specified location to acquire related data. Finally, the acquired data will be sent to the service requester.

2.3.7.5 Smart Grid

Smart Grid is the next generation power grid constituted by traditional energy networks integrated with computation, communications and control for optimized generation, supply, and consumption of electric energy. It brings about the following challenges on exploiting Big Data.

Grid planning by analyzing data in the Smart Grid, the regions can be identified that have excessive high electrical load or high-power outage frequencies. Such analytical results may contribute to grid upgrading, transformation, and maintenance;

Interaction between power generation and power consumption an ideal power grid shall balance power generation and consumption. However, the traditional power grid is constructed based on one-directional approach of transmission-transformation-distribution-consumption, thus, leading to electric energy redundancy and waste. Therefore, smart electric meters are developed to improve power supply efficiency;

The access of intermittent renewable energy there are many new energy resources, such as wind and solar, that can be connected to power grids. However, since the power generation capacities of new energy resources are closely related to climate conditions that feature randomness and intermittency, it is challenging to connect them to power grids.

In this chapter, we have seen that Vehicular Ad-hoc Networks (VANET) are more and more used, and how they work in Europe, with the European Telecommunications Standards Institute (ETSI) standardization. We have seen that VANET will not escape to the Big Data technologies needs. After that, we have seen an introduction to the Big Data technologies, and the different steps where Big Data is usable. In the next chapter, we will see why Big Data technologies can be applied with VANET.

Chapter 3

Applying Big Data Technologies with VANET

We have seen in Chapter 2 how VANET and Big Data work. Technologies and evolution need some adaptions of the Big Data and VANET standards. We will see them in Section 3.1, where we will see the concepts of Big Data from the Section 2.3 adapted to VANET, and then, some European projects adopting VANET with Big Data technologies in Section 3.2.

3.1 Are VANET Data Big Data?

Now that we know exactly what is Big Data and how it works, we can see if VANET are compatible with it. We will see how VANET can generate data in Section 3.1.1, how we can acquire them in Section 3.1.2, how to preprocess them in Section 3.1.3, the transport them in Section 3.1.4, store them in Section 3.1.5, and finally, the analysis of them will not be so different from "classical" Big Data, because when the storage problem is solved, we can use these data like with others.

3.1.1 Data Generaton in VANET

As seen in Section 2.2, in Europe, the ETSI standardization permits to ITS-S to generate message data, and we can have some data either in the payload or in the header.

For each kind of Extended Header, we will be able to get some good data:

Beacon/SHB, we have the position of the sender if we are near to him;

GeoUnicast, we have the position of the sender even if we are not near to him, and we can see how many hops were needed to reach the destination ITS-S;

GeoAnycast/GeoBroadcast, we have the position of the sender even if we are not near to him, and we can see how many hops was needed to

reach the destination area, and with the GeoBroadcast, we can see the impact of the topology inside the destination area;

LS Request/LS Reply, we can see how the vision of the topology by an ITS-S involve.

And with the Facilities layer, we can have some other data at a higher level:

CAM, we can have a better vision of an ITS-S situation, and how the others ITS-S react with it;

DENM, we can see the behavior of the ITS-S receiving the event, and with the GeoBroadcast forwarding algorithm, and the KAF, we can see the impact of the topology.

Besides message data, in VANET, we have another type of data: sensor data. Sensor data is a big source of data, because it covers from GPS sensor to exterior lights status or cruise controller etc. Some of these data are present in the message data, but for the others, we need to put them into log files, and so the definition of how to get these data is implementation-dependent.

3.1.2 VANET Data Acquisition

In VANET, to get data, we have many options: we can get log files in offline mode, or get the network data *via* network sniffing tools or with the ITS-R. The online mode can also be used to get log files, but it is necessary to define the transfer protocol from the ITS-V to the ITS-R, which then forward them to the ITS central platform, or directly from ITS-V to the ITS central platform through the cellular network. The log files will be mainly used for sensor data, since they are not all in message data, and message data will be capture because of LibPcap tools or zero-memory tools (Section 2.3.2).

3.1.3 VANET Preprocessing

When we want to collect the data, we need a preprocessing to make sure not to get wrong or redundant information. We will see here how we can preprocess messages, sensor data and how the routing protocol may “implicitly” preprocess data.

3.1.3.1 Message Preprocessing

According to CAM and DENM structures, there are two ways that we can eliminate redundancy and make aggregations for data. First, we can make an aggregation of CAM data by creating a lighter CAM that contains only average data (speed, heading, acceleration, length, etc.) in a given observation interval and transmit only these aggregated data to the Big Data platforms. Second, we can also select the predefined useful data in the DENM, because some information (e.g. height and position of the longitudinal career left or right) are here to improve the impact reduction in pre- and post-crash use cases, but are *a priori* not useful for other statistics.

3.1.3.2 Sensor Preprocessing

In order to prevent Internet of Things (IoT), where many sensors are deployed, from sending useless data or too much amount of data too often, the preprocessing can be set to send data only when significant changes occur. For example, a temperature sensor does not need to communicate engine temperature every second, but only when it is hot enough to become dangerous or to prove that it is still alive. If sensor does not send other data than "alive data", it means that data has not evolved enough to be notifiable. Some types of thresholds can be set for this.

3.1.3.3 Implicit Preprocessing

Fortunately, some preprocessing is implicitly made by many duplicate packet detection (DPD), inside the GeoNetworking routing (Section 2.2.1.2, and [10]), and with the KAF mechanism (Section 2.2.3.2, and [13]). In VANET communication protocol, if a message is sent many times by multi-hop broadcasting or by sending many times the same event when new neighbors appears, the protocol discards these messages, and reduce the redundancy.

The preprocessing is necessary when we have a huge volume of data. For the moment, there is not enough OBU, RSU and sensors deployed; however, it will be necessary in very near future.

3.1.4 VANET Communication

Now that we have our data generated and preprocessed, we need to communicate them to data centers. For a better understanding, we will firstly describe the VANET architecture, then dissemination mechanisms to reach data centers, and finally security mechanisms will be discussed.

3.1.4.1 VANET Communication Architecture

Figure 3.1 presents the communication in VANET, there exist two types of communications commonly called V2X (Vehicle to X):

1. Vehicle to Vehicle (V2V), when data are sent from a vehicle to another to cooperate or to warn of an event;

2. Vehicle to Infrastructure (V2I/I2V), when data are sent from a vehicle to an RSU (and vice-versa) in order to share informations about events.

As seen in Section 2.3.4, there are three layers in a Big Data network architecture:

1. *Access Network*. It is the user side, where devices communicate with the network infrastructure.

2. *Internet Backbone*. It is where data is transported to the data center. This means the communications from RSU to data center (and vice versa).

3. *Inter- and intra- data center network*. It is the network formed by the data centers. Intra-data center when data center are inside the same building or farms and inter-data center when data center are situated in different locations.

In VANET, the Access Network is the vehicle itself, but it also can be a smartphone application. If it is the vehicle, the communication is made in V2V as in V2I. The V2I permits sending data to the ITS-C, since V2V is more used in the vehicular context, but it can be used with a multi-hop usage to access to the ITS-R. The Internet Backbone is composed of all the ITS-R, which can communicate between them, and the ITS-C. Finally, the Inter- and Intra-DCN is composed by the different ITS-C and DC.

From a VANET point of view, the specific arrangement will be done at the access network. There are two main possibilities: use V2I communication to collect data or use cellular network. However, there is another alternative using a hybridization between these two technologies: vehicle uses V2I near an RSU and switches to the cellular network when RSU becomes too far, to guarantee QoS.

From a Big Data point of view, many challenges are here: we have the Volume of data to transport, and the Velocity of the data and network to manage. And so, we need to define how we can transport all these data from a volatile network topology.

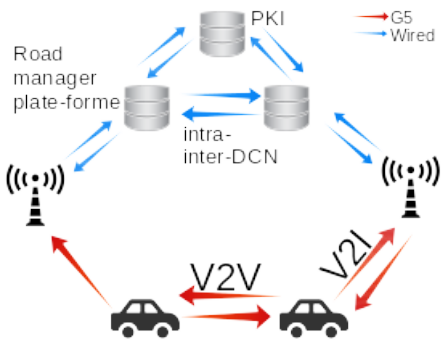


Figure 3.1 – VANET communication

3.1.4.2 Security and Privacy

When we use cooperative messages in VANET, all data are free of encryption, because if we encrypt these messages, there is no cooperation between vendors/manufacturers. So, clear data is mandatory, nevertheless one should not be able to follow a vehicle by tracking its data, and we need to make sure that sender is a real sender. To do that, ETSI messages can be secured with a security header and trailer. Security header contains all informations about the certificate used to sign the message while security trailer contains the signature itself. The certificate will not only be used to make a signature but also to make pseudonym that will change frequently. With this mechanism, privacy can be guaranteed. It needs to be mentioned that all these securities can triple the size of the message. That mechanism is the reason for the apparition of the PKI on Figure 3.1, but that PKI need to perform very well, because when we will have many ITS-V, they will need to get new certificates, and the PKI will also need to be capable to treat revocation lists and other security stuffs. That problematic increases the question of the Veracity, Variety, Volume and Velocity of VANET: Veracity, because we need coherent PKIs, Variety because there are many types of certificates that we can use [57], Volume, because of the number of vehicles, which sign each message, and finally, the Velocity by the use of pseudonyms, which evolve over time.

3.1.5 VANET Data Storage

As we have seen in Section 2.3.5, there are three types of databases storage: structured, semi- and unstructured data, and the storage problematic is not the same for each one of them. Here we will see which type of structuration we have, and in consequence, how to store them in a Big Data architecture.

3.1.5.1 Type of VANET Data

We recall here that CAM and DENM are structured messages defined by a standard, so they match structured data type. However, sensor data are implementation-dependent and so are semi-structured data because they are raw data that have an implicit structure to be able to retrieve data, but they can also be unstructured data (from video camera for example). Consequently, for messages such as CAM and DENM, we can use classical relational databases systems (RDBMS), however we need NoSQL databases for sensor data. Fortunately, NoSQL means Not Only SQL, and then we can use a NoSQL database to store both structured, semi-, and unstructured data [58]. Nevertheless, in the future VANET, we will need all these technologies because VANET will also include unstructured data like video, news, storm warning, etc.

CAM and DENM integrate Volume, Velocity and Variety problematic: we have a huge amount of various data, which evolve every time: for example, a moving ITS-S will send up to 10 CAM per second with different values each time, and two of them have a *HighFrequencyContainer* (see Section 2.2.3.1).

In the near future, we will need to be able to store efficiently other things than messages: videos, audio, and, why not, the airplane black box. Because we talk about vehicles, but airplanes are also vehicular, and they are also regularized by VANET laws.

3.1.5.2 Current Data Storage in VANET

In VANET, there two levels of storage:

1. ITS-S,
2. ITS-C (ITS Central platform)

At the ITS-S level, ETSI defined a standard called Local Dynamic Map (LDM) [59], which is used to retrieve events that occur near the ITS-S and to discover the neighborhood. This LDM is implementation-dependent; however, this standard only specifies requests between LDM and ETSI layers. Therefore, LDM mainly stores DENM and CAM. To access to the data, LDM supports filters in request like equals, not equals, greater than or equal to, like, not like etc.

At the ITS-C level, the storage method is implementation-dependent. The method will be made in function of data type we need to store. If we only store sensors' data, then semi-structured data will be used, but if we store only messages, then we will use RDBMS. Another incoming problem is the certificates and revocation list storage for the security layer, which need

to be very efficient to manage them: for example, if a station is revoked, all the ITS-S need to know it, or, if an ITS-S needs to update its pseudonyms list, the infrastructure must be able to do it quickly, not matter how many ITS-S is requesting it at the same time.

3.1.6 The VANET's Integration Into the 4V

In Section 2.3, we have seen that Big Data comes generally with the 5V: *Variety*, *Velocity*, *Veracity*, *Volume* and *Value* [7]. Here, we have seen that VANET are compliant with four of them:

Variety Different ITS manufacturers implement standard in different ways (e.g. some manufacturers have access to CAN bus and some others not).

Velocity With a frequency of 1 to 10 messages per second for each vehicle, and for each message, some data will change from one message to another like position, timestamp, speed etc.

Veracity All data need to be coherent with the situation to have an efficient cooperative ITS.

Volume With all data generated from messages and sensors and the number of vehicles, data volume increases fast. In [5, 6], they predict that connected vehicle market will grow from 5 million units to 35 % of the vehicle marketed in 2022 along with revenues of around 113 billion euros.

For the moment, Veracity and Volume are not a big concern because there are not yet enough connected vehicles. On the contrary, Variety and Velocity already raise new issues. In the near future, when more vehicles will be connected, then the huge volume of data will require Big Data algorithm (e.g., 60 bytes per vehicle per 100 ms). Veracity will be essential as well, because we cannot have an efficient cooperative ITS with incorrect data. Of course, if the car indicates a location of Berlin instead of Paris, the data will be absurd, and will be detected easily. However, when the car is near an access road, it is more difficult to know if it is on the access road or still on the highway.

Deep-learning approaches will be useful to determine situations and provide recommendations. We can determine two types of data processing: off-line and on-line. Deep-learning are more often off-line processing that need an important volume of data to be processed a priori. Use cases definition will be trained and validated off-line, and then we will use prediction

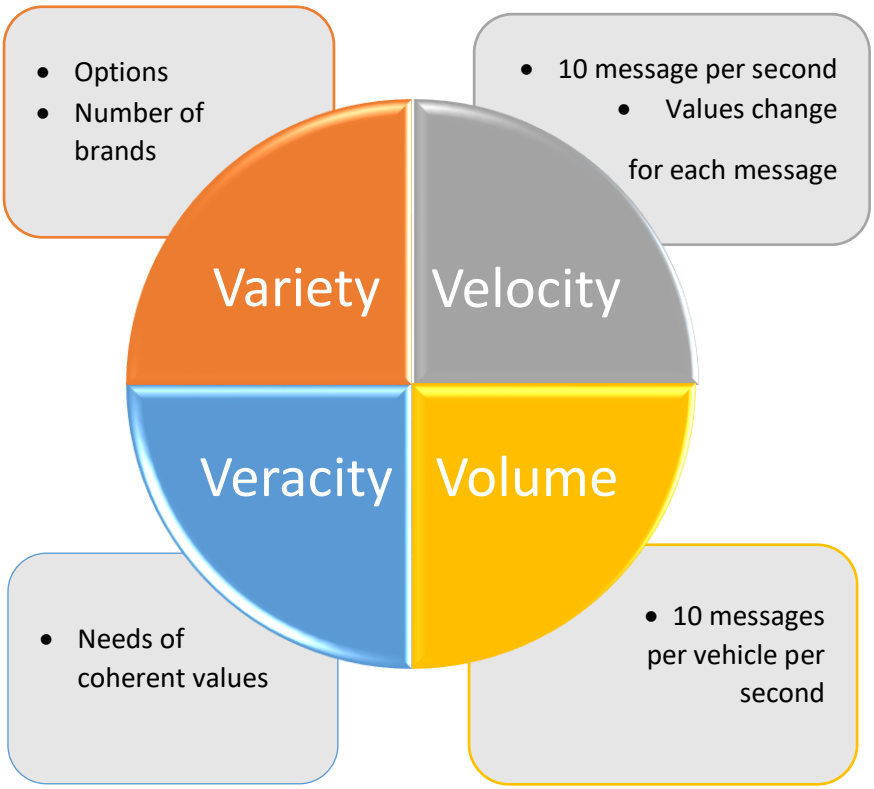


Figure 3.2 – VANET incorporation into the 4V

or recommendation on-line (e.g. for emergency braking). Moreover, we also need traceability, because if there is a problem, we can go back to look at the traces. We know that this situation is going to happen soon, so we have to consider it and begin to prepare solutions.

3.2 European Projects Using Big Data with ITS

In Europe, many projects are using Big Data in intelligent transports. This section summarizes four of the representative projects as follows.

1. TransformingTransport [60] is an EU-funded project that represents a strong consortium of 47 leading transport, logistics and information technology stakeholders in Europe. With a budget of €18.7 million and the participation of 47 organizations from 9 countries, the Horizon 2020

Project name	Country	Budget	Abstract
Transforming Transport [60]	Spain & Portugal	€18.7 million	Deployment of Big Data on ITS to cover 7 domains from ITS.
LeMO [61]	Norway	€1.5 million	Study and analyses Big Data in the European transport domain.
AutoMat [62]	Germany	€5.7 million	Standardize a Common Vehicle Information Model (CVIM).
Big Data Europe [63]	Belgium	€5 million	Implementation of an open source Big Data platform.

Table 3.1 – European projects using Big Data with ITS

Big Data Value Lighthouse project is working on finding a more efficient and more sustainable transport paradigm. It aims to show concrete, measurable and verifiable evidence of data value that can be achieved in mobility and logistics. This project aims to deploy Big Data for ITS in Spain and Portugal and cover 7 domains from ITS and each domain has its own platform: Highways, Vehicles, Rail, Ports, Airports, Urban Mobility, and Supply Chains.

2. LeMO [61] for Leveraging Big Data to Manage Transport Operations is an H2020 project that will explore the implications of the use of big data to enhance the economic sustainability and competitiveness of European transport sector. The project will study and analyses Big Data in the European transport domain in particular with respect to five transport dimensions: mode, sector, technology, policy and evaluation. LeMO will accomplish this by conducting a series of case studies,

in order to provide recommendations on the prerequisites of effective Big Data implementation in the transport field.

3. The core intention of the AutoMat [62] project is to establish a novel and open ecosystem in the form of a cross-border Vehicle Big Data Marketplace that leverages currently unused information gathered from connected vehicles. The interface to the marketplace forms one single point of data access for service providers and intends to supersede proprietary vehicle data interfaces. AutoMat develops and leverages the novel Common Vehicle Information Model (CVIM) data format, which harmonizes information and provides generic and brand-independent datasets.

4. Big Data Europe [63] aims, firstly, to collect requirements for the information and communication technologies (ICT) infrastructure needed by data-intensive science practitioners tackling a wide range of societal challenges; covering all aspects of publishing and consuming semantically interoperable, large-scale, multi-lingual data assets and knowledge, and, secondly, design and implement an architecture for an infrastructure that meets requirements, minimizes the disruption to current workflows, and maximizes the opportunities to take advantage of the latest European RTD developments, including multilingual data harvesting, data analytics, and data visualization. This project has been experimented in Thessaloníki with taxi drivers to make prediction on traffic flow [64].

In this chapter, we have seen that Big Data can be applied with VANET, because VANET raise the 4V issues:

1. Variety, by the options from standards, and the number of manufacturers;

2. Velocity, due to the number of exchanged messages, with different values per messages;

3. Volume, due to the increasing number of vehicles multiplying the message generation;

4. Veracity, due to the C-ITS situation.

We have also seen, that in Europe, many projects use Big Data technologies to manage ITS.

In the next chapter, we will see how we got our VANET data with the development of an ETSI ITS stack, and by validate ITS implementations from the Scoop@f project.

Chapter 4

Getting VANET Data

From now, we have seen how work VANET and Big Data in Chapter 2, if they can be applied together in Chapter 3, now we will see a testing architecture, which permit us to get emulated and real data from VANET. This chapter is presented as follows: Section 4.1 presents the ETSI ITS that we implemented, Section 4.2 how work laboratory tests, and Section 4.3 how work the road tests. In the two last sections, we will also present the tests scenario and methodologies to analyze them.

4.1 Development of an ETSI ITS stack

In order to get VANET data, we implemented the ETSI ITS standardization, but before explain how we implemented it, we need to introduce the main technology used: Qt.

4.1.1 An Open-Source Framework: Qt

Qt, pronounced **/kju:t/** because creators (Haavard Nord and Eirik Chambe-Eng) found that Q is cute on Emacs editor, is a cross-platform framework written in C++ for desktop, embedded and mobile, including Linux, OS X, Windows, VxWorks, QNX, Android, iOS, BlackBerry, Sailfish OS and other platforms. A preprocessor, the MOC (Meta-Object Compiler), is used to extend the C++ language with features like signals and slots. Before the compilation step, the MOC parses the source files written in Qt-extended C++ and generates standard compliant C++ sources from them. Thus, the framework itself and applications/libraries using it can be compiled by any standard compliant C++ compiler like Clang, GCC, ICC, MinGW and MSVC [65].

Development of Qt was started in 1990 by the Norwegian programmers Eirik Chambe-Eng and Haavard Nord. Their company, Trolltech, that sold Qt licenses and provided support, went through several acquisitions over the years. Today former Trolltech is named The Qt Company and is a wholly owned subsidiary of Digia Plc., Finland. Although the Qt Company is the main driver behind Qt, Qt is now developed by a bigger alliance: The Qt

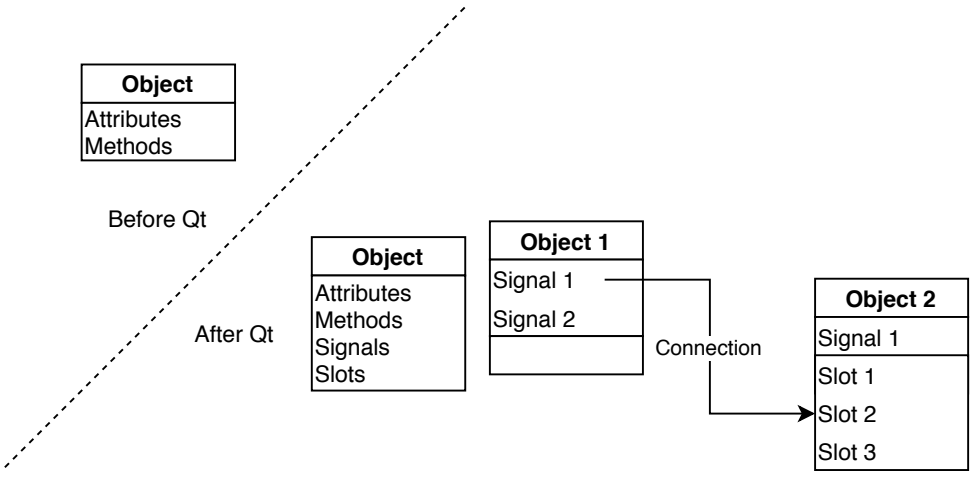
Project. It consists of many companies and individuals around the globe and follows a meritocratic governance model. Nowadays, KDE is almost entirely built with the Qt framework.

Qt is available under various licenses: The Qt Company sells commercial licenses, but Qt is also available as free software under several versions of the GPL and the LGPL.

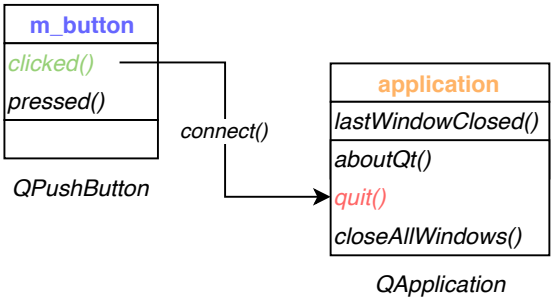
Although, any build system can be used with Qt, Qt brings its own qmake, which take a .pro file, which structures the project and transform it on a specific platform Makefile.

Qt comes with its own Integrated Development Environment (IDE), named QtCreator and also an internationalization and localization system and widgets for GUI development.

The motivations to use this framework were the facility to use its rich library, the open source license and most of all the signals/slots mechanisms. Since Qt was designed to GUI development, an event mechanism was built: signals/slot. When we launch a program, Qt objects can listen for some signals, which are connected to their slots. And so, when a signal is emitted, every slot connected to it will be called as described in Figure 4.1. When we use QObjects (object with Qt properties), we can add signals and slots to the object (Figure 4.1a), and we can connect signals to slots (Figure 4.1b). Note that a signal can be connected to many slots, and a slot can be connected by many signals (from and to many objects, even itself). In Figure 4.1c, when **m_button** emit the *clicked()* signal, **application** call the slot *quit()*.



(a) Objects and QObjects differences (b) Signals and slots connection



(c) Connection between a signal and a slot: when **m_button** emit the *clicked()* signal, **application** call the slot *quit()*

Figure 4.1 – Signals and slots mechanisms

4.1.2 ETSI ITS implementation

We will divide this part into two sections: the first one talks about the physical architecture (how are composed the directories and how are the C structures made and encoded) while the second part talks about the software architecture (how different parts interact and communicate). All the documentation is available on `http://scoop.univ-reims.fr/ITS_documentation`.

4.1.2.1 Physical Architecture

As described in Figure 4.2, we have a global project *ITS*, with several sub-projects: *Application*, *CAN*, *Facilities*, *GeoNet*, *GPS*, *Hybrid*, *LibITS*, *Management*, *Network*, *Security* and *UpperTester*. Some of them have other sub-projects: *Facilities*, *Network* and *LibITS*. The latter has another sub-project with sub-projects inside: *LibASN*. This division in many sub-projects permits to have a better view of the project in the editor and facilitates the versioning, because as SVN branches, sub-projects are sub-directories.

Application is the layer which interacts with the HMI and manages the triggering conditions of DENM: if the vehicle state needs to send an automatic DENM, the Application will trigger it;

CAN from now there is only a fake CAN bus provider, since we do not have a real CAN bus on our PC, which emulate a CAN bus;

Facilities is the module which launch the CA and the DEN if needed;

GeoNet is composed by the main file, which execute all needed modules;

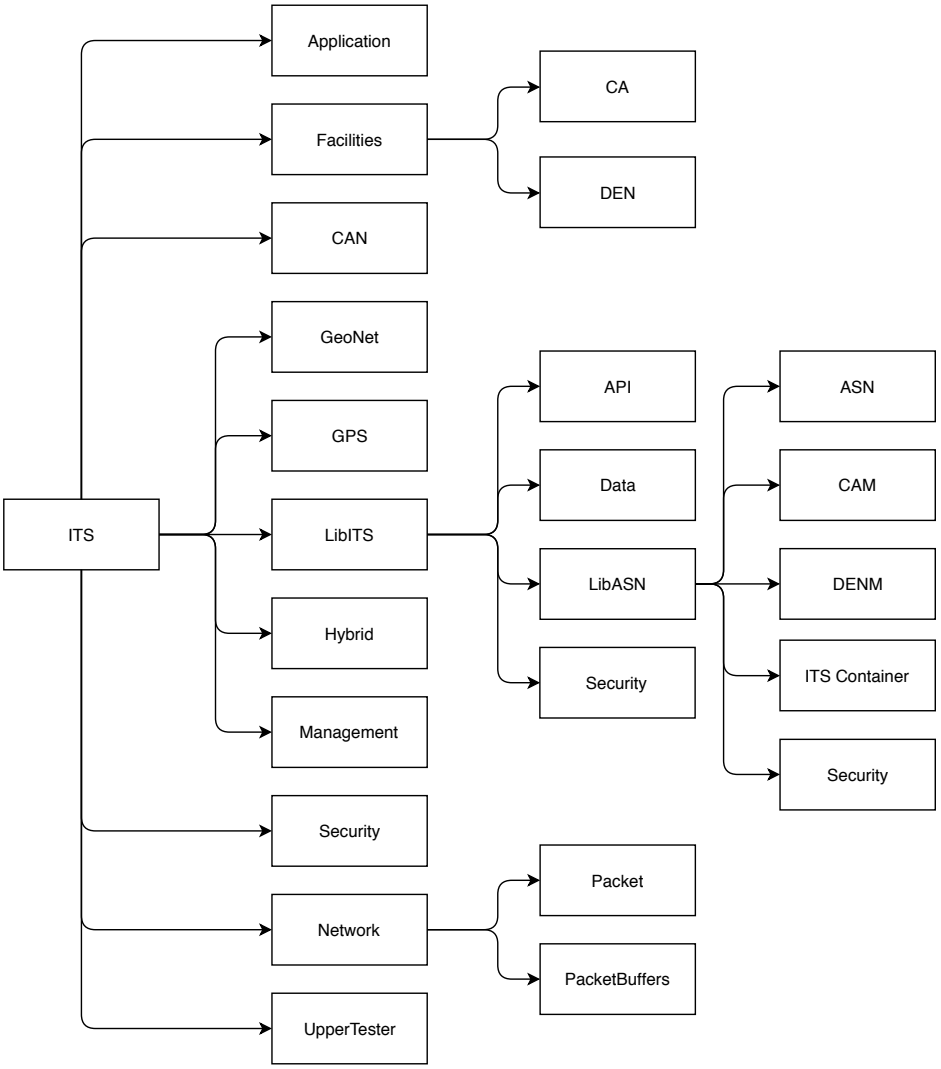


Figure 4.2 – ETSI ITS implementation architecture

GPS is the GPS module. It is composed of a real GPS module and a fake GPS provider for lab tests to simulate position or if we do not have a real GPS sensor;

Hybrid is the configuration module for hybridization between ITS-G5 and TCP or SSL sockets;

LibITS is where we will find all the API used by everyone;

Management is used to get the state of every module;

Network manages all the datagrams creations, the Beaconing when necessary and the sending to the ITS-G5 interface and hybrid interface;

Security manages the security profiles and configuration like certificate files, pseudonyms rounding etc.;

UpperTester is the modules used for lab tests (we will talk about it in Section 4.2.1.1.2, after the definition of an UpperTester).

For the Facilities sub-project, we have:

CA which is composed by the CA Basic Service to generates CAM and CA Reception Management to receive CAM and notify the application layer;

DEN which is composed by the DEN Basic Service to execute the DEN Triggering Service for triggering and update (also terminate) of DENM, DEN Reception Management to receive DENM and notify the application layer, and the DEN KAF Management if KAF is needed.

For the Network sub-project, we have:

Packet is the packet description: GBC, Beacon, SHB etc.;

PacketBuffer is the different buffer needed by the GeoNetworking protocol.

And for the LibITS sub-project, we have:

API with a generic API and specifics APIs in function of the module;

Data with serialization of some structures used between different layers of the projects;

LibASN is the library made by the open source asn1c [66] tool;

Security is the security layer codec to translate the buffer into a C structure and conversely.

The LibASN is divided into the different kind of structures (one by ASN file + the basic types and codecs) to not have a big directory with all inside:

ASN is the ASN types and different codecs;

CAM is the CAM structures;

DENM is the DENM structures;

ITS Container is the Common Data Dictionary structures used by CAM and DENM;

Security is the security layer structures.

To encode and decode CAM and DENM payload, we need to use the ASN.1 UPER, and so we use the asn1c tool, which converts ASN.1 files into C structures with different codecs: XER, BER, DER and UPER. Since this tool converts ASN.1 files into C structures, we decided to create an ASN.1 file of the security header and trailer, that's why there is a security sub-project inside the LibASN, so we can manipulate security layers structures and if the standard changes, we change the ASN.1 file, not the structures themselves.

4.1.2.2 Software Architecture

To implement the ETSI ITS stack, we decided to be as close as possible to the standards, and so we have the Figure 4.3 with:

Application layer communicates with the HMI in Bluetooth and with the Facilities layer in UDP socket;

Facilities layer communicates with the GeoNet layer by using a GeoNet public function to send packet;

GeoNet layer encapsulates Facilities payload in correct BTP / GeoNet header and send it with raw socket through the ITS-G5 interface or in TCP socket if the hybridization is set, and notify the Facilities with UDP socket;

Management layer communicates data from different sensors (CAN / GPS) through Qt signals to the others layers.

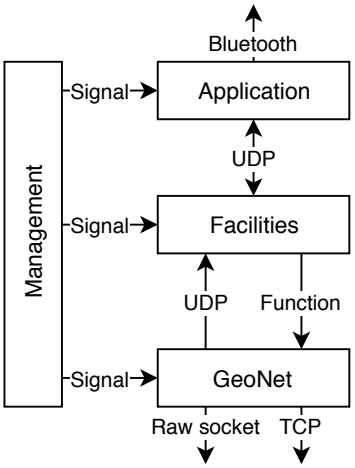


Figure 4.3 – Software architecture of the ETSI ITS implementation

In all the following sub-sections, methods read() and readDebug() have the same behaviors: read() reads the configuration associated to the module and set the module, readDebug() prints the values. To read configuration files, we use the `confuse` library [67]. In Appendix F, all the classes diagrams are available.

- **Manager**

  For better understanding of the working of the different layers we will first see how we implemented the Management layer. The Manager is the Management layer class with all the possible sensors and modules parameters. These sensors and modules are pointers. The Manager puts all set pointers (i.e. used sensors and modules) into a HashMap with key string pointing to void pointer. With this implementation, modules and sensors get the Manager pointer in their constructor, get the pointer to the sensor or module parameter needed and if value change in them, they automatically get the new value. The Manager has a signal *midToChange* connected to its slot *changeMID*. This signal is used by the GeoNet layer when a duplicate address is detected to virtually change the MAC address.

  We use sensors and application in the Manager. Actually, real GPS and CAN bus providers and application are classes, which inherit Fake sensors and application. By doing this, if the sensor or application is used, we put the real sensor / application into the Manager's HashMap,

the Fake one otherwise. Parameters are used to initialize different modules and the GnAddr is the GeoNetworking Address of the ITS-S.

- **Sensors**

  For GPS and CAN bus, the implementation is quite similar. They inherit of a Fake CAN and GPS provider used if we do not have / need the real sensor. They both have a Parameter with all values: heading, position speed and timestamp for GPS, and acceleration control (gas pedal, brake pedal cruise control etc.), curvature, curvature calculation mode, drive direction, exterior lights state, external temperature, lateral acceleration, light bar and siren state, longitudinal acceleration, number of occupants, positioning solution type (GNSS, dead reckoning, both), position of occupants, speed (used if we do not have GPS), steering wheel angle, vertical acceleration and yaw rate for CAN bus. All these values are the ones used by CAM and DENM. If we need more values, we need to add them. Fake classes have functions to change values used by the UpperTester (see Section 4.2.1.1.2). Since we only have a real GPS sensor, only GPS has a real class GPSProvider with a function to get the GPS position every 100 ms (minimum frequency of CAM generation). When a value from a sensor is changed, a *positionChanged()* or *canChanged()* signal is emitted.

  Now that we have seen the left side with the Manager and sensors, we will detail from top to bottom (Application to GeoNet layer).

- **Application**

  As for sensors, there are two types of Application layer: the real one (the ApplicationManagement) and the fake one for the UpperTester (see Section 4.2.1.1.2). The difference with sensors, is that the real Application does not inherit from the Fake one. They do not have the same job. The Fake one set different parameters and the real one triggers events (automatic or manual). The FakeApplicationProvider is composed by a ParameterApplication which have the different type of static values that can be changed on time by an external application; and the ApplicationManagement have the connections to the CAN bus and GPS sensor, the UpperTester if there are tests with CAM (notification of CAM reception) or DENM (generation and notification of DENM reception), and all the necessary for a good Bluetooth communication with the HMI. Functions from the FakeApplicationProvider will be used by the UpperTester to set some values: station type, vehicle role, embarkation status, priority (on road: with traffic light, cross

etc.), type of dangerous goods transported and the station ID. The ApplicationManagement works as follow: if there is the application needed for triggering event, it connects the CAN and GPS signals to its slot situationChanged(), if there is Bluetooth, it turns the server on to communicate with the HMI, initializes UDP sockets with the Facilities layer and if there is an UpperTester, sets the UDP socket to communicate with the UpperTester. When a client is connected to the Bluetooth server, the server adds it to its client list. When it receives a message from the HMI (or the UpperTester), it sends it to the Facilities, and when it receives a message from the Facilities, it sends the message to the clients and to the UpperTester if needed.

- **Facilities**

  The Facilities layer is composed by three sub-layers: Facilities, CA and DEN. The Facilities sub-layer executes the CA and DEN if needed. The CA is composed by the CA Basic Service to generate CAM and the CA Reception Management to receive CAM. The DEN is composed by the DEN Basic Service, the DEN Triggering Management, the DEN Reception Management and the DEN KAF Management.

  The DEN Basic Service set the different modules and forward requests to the appropriate DEN module and returns the result to the Application. The DEN Triggering Management manages the triggering, updating and termination of DENM; the DEN Reception Management the DENM reception and executes the KAF if activated; and the DEN KAF Management the KAF procedures. Besides the three modules, the DEN Basic Service is composed by the Application parameters, DENM parameters, a UDP socket to communicate with the Application layer, and a function AppDENM_terminate to call the good module to terminate the DENM (DEN Triggering Management if it is my event, Reception Management otherwise). The DEN Triggering Management has some parameters to set DENM and a vector of denMessageTableSource to store triggered DENM. The DEN Reception Management has also some parameters and a vector of denMessageTableReception to store received DENM and a UDP socket to notify the Application. Both DEN Triggering and Reception Management have a sequence number to be coherent with the identification of DENM. The DEN KAF Management has parameters and a vector of denMessageTableForwarding to store DENM to repeat. These vectors have the same working template: they set the different timers and start

them. The timers are connected to slots from threads, which execute the functions described in Section 2.2.3.2.

It is easier for the CA: the CA Basic Service wait for a timer or a signal reception that position or CAN bus changed. It then looks if it can generate a CAM, generates and send it; and the CA Reception Management, receives the CAM, decode it and notify the Application.

The DEN Reception and CA Reception Management listen to a UDP socket on the Localhost and the port corresponding to the destination port from the BTP standard (see Section 2.2.2). With this mechanism, we do not need to specify an ad-hoc port, and if there is no Reception Management, the GeoNetworking layer will not be disturbed. If there is a destination port info in the BTP layer, it is managed by the Reception Management.

- **GeoNetworking**

  The GeoNetworking layer has only two working classes: GeoNet and ReceivePackets. That GeoNet class is responsible for the Beacon generation and sending (if no source position vector is sent), and has a function used by it and by the Facilities layer to send data. The GeoNet class manages also the processing of received packet by the ReceivePackets class. To send packets, GeoNet create a raw packet to send in the interface from configuration file. It permits to not have an ITS-G5 interface to send packet: we can use all kind of interface (Wi-Fi, Ethernet, Localhost etc.). If the hybridization is turned on, a TCP socket is created to send the same packet as for ITS-G5 interface (except for the Ethernet layer). By doing this, when we receive packet through the ITS-G5 interface or hybridization, it is the same function to call to process it, and so the same behavior. To know how to send data, the sending function needs a *gnDataRequest_t* structure:

```
typedef struct gnDataRequest_s
{
    CommonHeader::e_upperProtocolEntity upperProtocolEntity;
    CommonHeader::e_headerType packetTransportType;
    quint8 packetTransportSubType;
    quint16 destinationPort;
    quint16 destinationPortInfo;
    destinationAddress_t destinationAddress;
    e_gnCommunicationProfile gnCommunicationProfile;
    LifeTime* maximumPacketLifeTime;
    quint16* repetitionInterval;
    quint16* maximumRepetitionTime;
    quint8* maximumHopLimit;
    TrafficClass trafficClass;
    QByteArray payload;
}gnDataRequest_t;
```

with *CommonHeader::e_upperProtocolEntity* an enum with possibles values {ANY, BTP A, BTP B, IPv6}; *CommonHeader::e_headerType*

and *packetTransportSubType* the values of the *HeaderType* and *HeaderSubType* from the *CommonHeader*; *destinationAddress_t* a union between a GeoNetworking address or a destination area, *e_gnCommunicationProfile* an enum with values {Unspecified, ITS G5A}. Each "person" who wants to send data needs to fill in a gnDataRequest_t structure.

To be able to use the different routing algorithms, the GeoNet class is composed by different packet buffer and each kind of packet has a class to be put in HashMaps (we need comparison functions, and only classes can do it easily). By making classes of each packet type, we can fill them faster and easier than with structures.

The ReceivePackets class has only the configuration parameter to know which interface to listen, and which MAC address to not listen (our MAC address) and a signal connected to the GeoNet packet processing slot. In terms of algorithm, using the pcap library [68], we look up for ITS-G5 interface, open it on live mode, handle it and listen to it with the filter Ethernet with a protocol number 0x8947 and the Ethernet source different of the ITS MAC address. When the MAC address changes for different reason, the ReceivePackets class change the filter and listen again.

For the security layer, we have a SecurityManager class which knows the different profiles and stores the received payload and received certificates. With this, it can adapt the profiles to send and can retrieve if a sender has the right SSP to send packets. To encode or decode the security layer of a packet, the GeoNet use the SecurityManager's functions.

4.1.3 Executing the ITS stack

As mentioned before, configuration files are managed with the `confuse` library. All the configuration files are placed in the */etc/its/config* directory. We have 8 files:

1. appli.conf
2. btp.conf
3. can.conf
4. denm.conf
5. gps.conf

6. security.conf
7. system.conf
8. upperTester.conf

All these files permits to set the ITS default configuration and all parameters need for laboratory testing.: appli.conf for the ParameterApplication attributes, btp.conf for the BTP ports number, can.conf for the ParameterCAN, denm.conf for the KAF and values of request and result identifiers, gps.conf for the static position and the address of the GPS deamon, security.conf for the localization of the private key and certificate, system.conf for the networking configuration and static values like vehicle length, and the upperTester.conf for the ports used by the UpperTester and the destination area to use for DENM triggering.

Once the configuration is made, to execute the ITS stack, we call the `geonet` program:

```
Usage: geonet <abcfCDgHnuh>
-a --application: Activate the ApplicationManager
-b --bluetooth: Activate the Bluetooth
-c --can: Activate the CANProvider
-f --facilities: Activate the facilities layer
-C --cam: Activate CAM only
-D --denm: Activate DENM only
-g --gps: Activate the GPSProvider
-H --hybrid: Activate the hybridization
-n --geonet: Activate the ITS-G5 interface
-u --upperTester: Activate the UpperTester
-h --help: Print this help and quit
```

4.2 Laboratory Testing

After the implementation of a standardization, we should conduct some tests. To do so, we can test it in laboratory with an emulated environment, or in a real situation. The laboratory tests offer a less cost testing, in a controlled environment, but since the environment is emulated, it cannot reflect all situations, that's why in a second time, when the laboratory tests pass, we can test the implementation on real environment. We will first see how to test on laboratory the conformance to a standard (Section 4.2.1), then functional tests (i.e. Scoop@f conformance tests) in Section 4.2.2, and then how to test an ITS implementation on road (closed or not) in Section 4.3.

4.2.1 Conformance Testing

On ITS-G5 laboratory tests, each component will be tested in line with the process presented in Figure 4.4 to see if it is in conformance with the standardization, we talk about conformance tests. On one hand, we have the tester composed of two elements: a computer equipped with testing engine, and an ITS-G5 gateway customized to broadcast every 802.11p message received by the wireless interface, on the other interface. This allows the computer to analyze the G5 flow in order to execute the tests. The tester is linked to the System Under Test (SUT, i.e. the implementation to test) by an Ethernet connection. Thanks to these two links (G5 gateway and Ethernet link) we can execute automatically a set of tests. Note that the ITS-G5 is required only if the SUT use this type of interface. Indeed, the implementation can send messages through the classical Wi-Fi or the Ethernet connection, since we have an Ethernet header with the Header Type changed (as a reminder, with a value of 0x8947). Thanks to this, an implementation can be tested on the same computer as the tester unit by using the Localhost (that's how we first tested our implementation). Of course, the ITS-G5 is useful only for the ETSI ITS-G5 tests, if we test a coffee maker, we do not use that gateway. To be tested, the SUT needs to implement a piece of code called UpperTester [69] (which we will talk more about in Section 4.2.1.1.2).

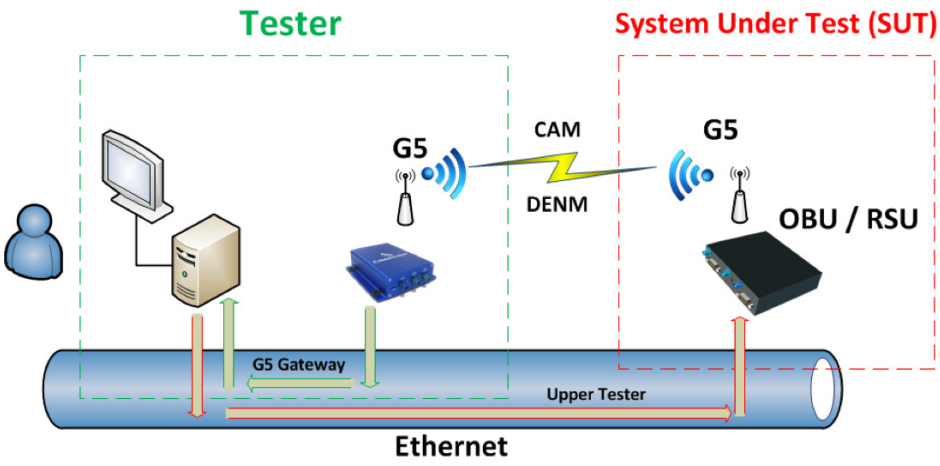


Figure 4.4 – Conformance tests architecture

4.2.1.1 Testing Framework

A testing framework is mainly composed of 5 parts:

1. Test Runtime: it is the software which executes the bench of test, it has to be time synchronized with the SUT;

2. Abstract Protocol Tester (APT): it is an implementation which simulates the layer we intend to test;

3. Component to test: it implements the UpperTester in order to be stimulated by the Test Adapter using common primitives, which will trigger some behaviors on the component;

4. Test Adapter: it is independent of the software, written in any programming language, it provides the transport and translation of messages between the SUT and the APT;

5. Codecs: it defines the rules that the Test Adapter will use to translate and to encapsulate the TTCN-3 (the formal testing language) structures into SUT messages and vice versa.

And the test execution is achieved as follows:

1. APT sends message either to UpperTester or the gateway;

2. Test adapter translates the message in a binary one and add all required protocol headers;

3. The SUT receives the message and reacts (or sometimes the expected behavior is to not react), using one of the two links;

4. The test adapter translates the message into a TTCN-3 structure by using codecs;

5. The APT examines the reaction and provides a verdict, which can be one of the follows:

   success All works as intended;

   inconclusive We do not know if there is problem. For example, if we do not receive an intended message, we do not know if it is because the implementation does not work or if the connection with it is broken. Note that the inconclusive result is different from a fail result;

   fail The resulting behavior is not the one expected.

4.2.1.1.1 TTCN-3 and Test Adapter

The TTCN-3 (for Testing and Test Control Notation Version 3 [70]) is a testing language used to be independent of the tester unit by offering a hardware abstraction. The tester implements the tests in this language, and then compile it in the same language as the Test Adapter (if possible). The Test Adapter (TA), implements the different layers used: the functions to use (get timestamp for example), the primitives to send to the UpperTester (change the speed value for example), and all the codecs (encoding in ASN.1 UPER, add the ETSI headers etc.). The AT and the pre-compiled TTCN-3 tests are built to make an executable program, which we will configure to make some tests, as if they were simple functions. The program will be executed by the Test Runtime. There are a lot of Test Runtimes: TTWorkBench [71] (TTWB), Titan [72], or Elvior [73], to mention just a few.

TTWorkBench is a proprietary tool developed by Spirent as an Eclipse plugin with an AT wrote in Java language. This tool is composed by 2 views: the developing view and the execution view. In the first one, we can write the TTCN-3 tests and the AT, and compile them to produce tests campaigns. In the execution view, we can configure tests and execute them partially or totally.

Elvior is a proprietary tool with an AT, which can be in C, C++, C# or Java language, and is with its own Graphical User Interface (GUI) [74].

Titan is a free and open-source project developed by Eclipse as an Eclipse plugin to write TTCN-3 code and execute tests. The AT is written in C++, and the tests can be executed or built in command line (CLI).

Since the ETSI plugtests are made with the TTWB framework [69], we used it with the Scoop@f partners, to be in the same conditions as for the plugtests, but to test our implementation, we used both TTWB and Titan, with a reimplemented AT. By developing our ITS-G5 stack and the AT for Titan, it permits us to detect some bugs from the tests used in the ETSI plugtests and notify them, but also some bugs with TTWB and Titan behaviors.

4.2.1.1.2 UpperTester

The UpperTester is almost the most important thing when we make laboratory test. It is the software, which will communicate with the tester to emulate the environment, and notify it from all events (reception of message, or result of a command from the tester). The UpperTester communicates with

the tester using UDP. In the ETSI ITS-G5 tests, there are 44 primitives, but in Scoop@f project, we only use the 27 firsts above the line:

3 common primitives,

14 CAM primitives,

4 DENM primitives,

6 GeoNetworking primitives,

---

3 IPv6OverGeoNetworking primitives,

3 BTP primitives,

3 MAP/SPAT primitives,

4 IVI primitives,

4 SRE/SSE primitives (traffic light control [75]).

To recognize each primitive, we use the first byte, which define which primitive is in the message. These message type bytes are classified: 0x0x for the common primitives, 0x1x for DENM primitives and so on (except for the three lasts, which are all in the range 0xAx).

4.2.1.1.3 UpperTester Implementation

The UpperTester is an emulation of received signals of the tested ITS-S, and so, **it should not change the behavior of the ITS-S by itself**, but make the ITS-S change its behavior. That's why in our implementation, the UpperTester use the functions of Fake GPS, CAN and Applications, and that's why these Fake modules have attributes corresponding to the UpperTester primitives. To encode primitives, the UpperTester has a table of 256 possible primitives (the primitive identifier is encoded in one byte). These primitives are function pointer to the primitive itself. Using a table of function pointer is more efficient than a vector or an HashMap, because we call the m_req[request[0]] (assuming that our table is called m_req, and the payload request), we are in $O(1)$ and not in $O(\log(256))$ with a vector or an HashMap. If the request number does not exist, we respond to the tester with a failure response. When the UpperTester needs to send a GeoNetworking packet, it calls the GeoNet send function with a filled gnDataRequest_t as if it is the Facilities layer. If it is a DENM test, it creates an AppDENM request as if it is the Application layer. If it has to change a parameter, it uses the parameter's function dedicated to.

4.2.2 Functional Testing

To test the functional part of an ITS-S in the Scoop@f project, we have to test the if the triggering conditions are respected (Section 4.2.2.1), the log generation that will be used in road tests (Section 4.2.2.2), the interface between ITS-R and central platform (ITS-C) in Section 4.2.2.3, and some interoperability tests (offline tests in Section 4.2.2.4, and online tests in Section 4.2.2.5).

4.2.2.1 Triggering Conditions

On an ITS-V, automatic DENM messages can be triggered by respecting some conditions of the vehicle and their environment. Then a careful attention should be given to this event generation. Series of test cases have been derived from the specification of these triggering conditions. In order to achieve these test cases, we either need to have a connection to a CAN Bus emulator or to implement an adapted UpperTester on each component to be able to receive primitives emulating car environment events. For the Scoop@f we needed to extend the UpperTester primitives to emulate the triggering conditions like setting the strength braking to test the emergency braking for example. To do that, we use the unused message type bytes, and define the data format (see Figure 4.5, and Table 4.1 for the strength braking example).

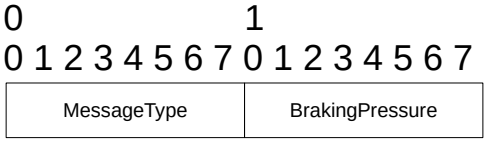


Figure 4.5 – An extended UpperTester primitive

Name	Length	Value
MessageType	1 byte	0xB6
BrakingPressure	1 byte	[0-252] braking pressure 253 saturation 254 sensor test running 255 unavailable

Table 4.1 – Strength Braking primitive's values

4.2.2.2 Log Generation

Each component should generate all log records related to pertinent events. The events are mainly those about emission and reception of messages (for any layer). A common specification of log records should be defined and each vendor should insert a generation module in the developed component.

Two types of logs are considered:

- TLog (Technical Log) that are generated when an event occurs (mainly emission or reception of messages). TLogs will be used essentially for road test;
- ULog (User Log) that are generated periodically with vehicle data. ULogs will be used essentially for a Scoop@f supervision purpose.

The access to a log file server is also considered in the specification. In order to use these log files, the generation is tested by simulating the server and by triggering the SUT for TLogs or just listening for ULogs. The laboratory test of the generation and emission of these logs is crucial, since if it does not work on road, it is very difficult to determine where the problem is, and it is more expansive to test.

4.2.2.3 ITS-C and ITS-R Interfaces

Usually, in Europe, most of road operators handle the DATEX messages to send (or receive) requests from the ITS-C (or to the ITS-C). In the Scoop@f project, ITS-R can receive or emit DATEX messages. These DATEX are used to send or notify of DENM, and to aggregate CAM with an average DATEX CAM. The translation on any RSU of received DATEX message into an equivalent DENM (or sent DENM into an equivalent DATEX message) has to be checked by simulating the two aspects: send a DATEX message to the RSU and listening the derived DENM; send a DENM to the RSU and listening DATEX messages sent to the ITS-C. To test these aspects, we had to develop a DATEX server, which send DATEX to create DENM, and a simulator of ITS-V, which sends CAM and DENM to test if the ITS-R send the good DATEX to the server.

4.2.2.4 On Laboratory Offline Interoperability Tests

To ensure the interoperability between the equipment (OBU and RSU), we need first to test the communication between them. Since they are located in different sites, we will start by testing the interoperability between them

(using off-line principle). To test on laboratory the interoperability, we exchange some PCAP log files, and we replay them with different equipment, by modifying timestamps and locations.

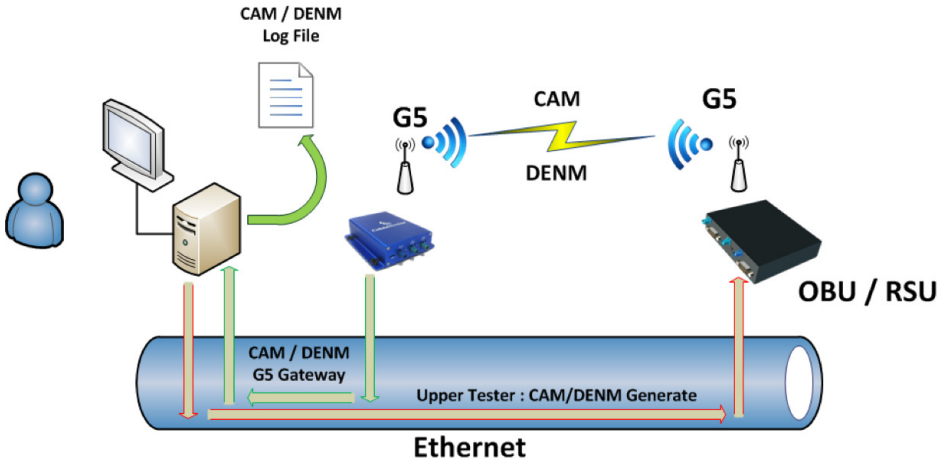


Figure 4.6 – Log generation architecture

In Figure 4.6, the SUT must be stimulated in order to start sending CAM and DENM messages. The messages sent will be captured by the Ethernet-G5 gateway. Then, these messages are transmitted to the PC via the Ethernet link, where the Wireshark tool should be running. This tool is used then to save each message in a PCAP format.

For each equipment, we intend to generate these types of messages:

1. CAM: For each equipment, a log file with a minimum of 10 CAM messages has to be generated.

2. DENM: For DENM messages, 12 messages must be saved each in a separate file.

The architecture for tests execution is almost the same as for the log file generation. However, the data flow will be reversed. Figure 4.7 describes this architecture.

A CAM or a DENM message must be extracted from the PCAP log file. Then, it has to be broadcasted by the Ethernet-G5 gateway. The message reception is verified using the OBU/RSU HMI or using the Upper Tester indication.

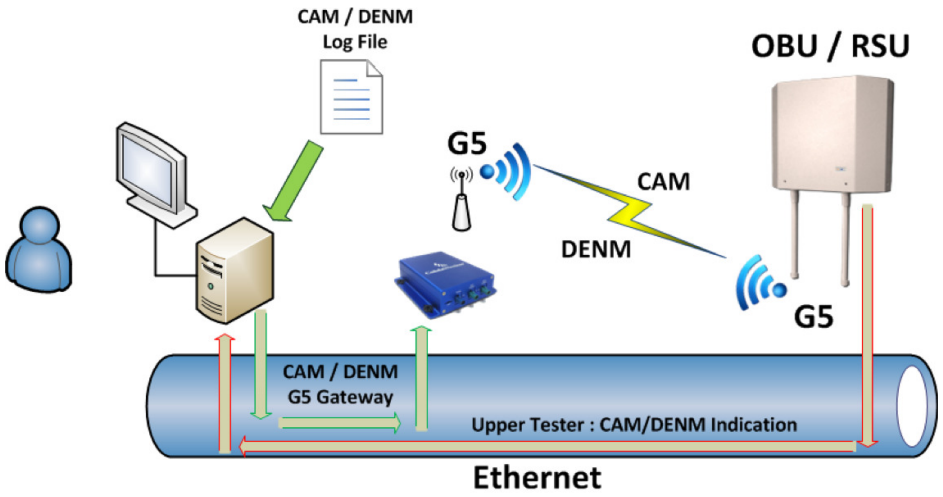


Figure 4.7 – Test execution architecture

4.2.2.5 On Laboratory Online Interoperability Tests

Figure 4.8 shows the architecture of the on-line interoperability. In this architecture, the SUT is composed of two components (OBU or RSU). One of the components is stimulated via the Upper Tester (UT) link in order to send a CAM or a DENM message. When the other equipment receives this message, it sends a CAM or a DENM indication to the PC via the Ethernet link. This indication includes all the data received in the message. Then, the PC compares the data of the message sent and the one received. After that, it decides if the test succeeds or fails. This is repeated for all DENM use-cases.

4.3 Road Testing

When we want to make experimentation on road, we have two options: online or offline tests. In both cases, we need some log systems, because instead of laboratory tests, vehicles are real, and we can not simulate the vehicle behavior: each value can not be exactly set. Values from these logs need then to be parsed with an error rate acceptable, which depends on the standards and deliverable interpretation. When we use the offline method, we make several tests, get the logs, and then we compare the values more or less automatically, and we set verdicts. With online method, the ITS-S needs to transfer its logs to a test unit, which will compare the values. In both

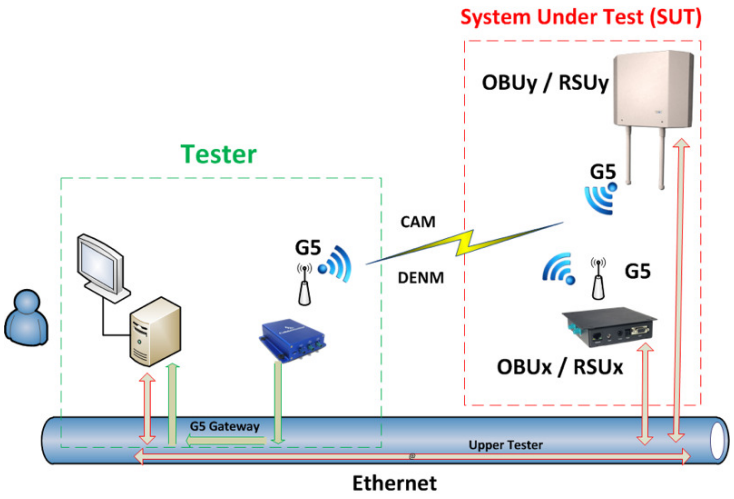


Figure 4.8 – Interoperability tests architecture

cases, we need a very precise and well-formed logs file to be sure to not miss something, and with online tests, we also need a good way to get the files. All these mechanisms (log generation and transfer) need to be tested on laboratory, in order not to go to the road with faulty systems.

4.3.1 Road Test in Scoop@f Project

In the Scoop@f project, both online and offline tests can be done. For the log file format, several ASN.1 definitions were made: two (technical logs called TLogs for the ITS-S state, and user log called ULogs for the Scoop@f application utilization supervision) for each type of ITS-S (customer vehicle, RSU and road operator vehicle). By doing these ASN.1 separations, each type of ITS-S does not have value which does not concern them, for example, RSU have no value about its speed. We decided to create ASN.1 definitions to ensure the implementation independence and not be affected by the hardware representation of numbers, and to be able to encode the logs file with ASN.1 UPER as for CAM and DENM, since all the manufacturers should be able to use this encoding for CAM and DENM, and rules are the same with all kind of ASN.1 structures. The motivation to encode with the ASN.1 UPER is that this is the most efficient way to encode data without less and with the small size, because we use the exact number of bits needed to encode the values.

To send these logs to the test unit, the extension possibility of the ASN.1 was used. In the CAM definition, the RSU container can be extended to add

some infrastructure services. In the Scoop@f project, that capability was used for the security procedures and for the file transfer. For the security, the RSU transfers the requests from the ITS-V to the PKI server for certificates updates and some other stuff; for the log files, ITS-V send the files to the RSU, which forwards them to the central unit. The extension, called CAM-I for CAM-Infrastructure, is made as in Appendix G.1. Three new containers are added:

1. Service Advertisment Container for the service details mike communication and type of service;

2. Position Enhancement Container to involve the positioning solution of ITS-V;

3. Environment and Context Container for the environment information.

The Scoop@f system has some disadvantages due to industrial constraints. If an RSU wants to make some security services, it will send the CAM-I with an Advertised Service ID at the security service value, and if it wants to forward log files, it set the Advertised Service ID with the log file transfer service number. But if the RSU wants to do both, it needs to send two different CAM-I. In Scoop@f, to send the two different CAM-I, they send five CAM-I of the first service and then five other CAM-I of the second service per second, which is in contradiction with the standard (one CAM per second for the RSU). Moreover, the BTP port used to send CAM-I is the same as the CAM, which is not semantically correct. When we use BTP-B, we have a destination port info which is used for this purpose.

When the log files are collected, we need to decode the UPER, but with all the different ASN.1 files and no system to identify the type of log file, and some types has the same name but not the same values and range, if we want to make a tool to decode them, we have to internally rename them, and select manually the log type.

To remedy this situation, we propose a new way to send the Advertisment Services and to define the logs.

4.3.2 Evolution of the Log System Proposition

One of the two Scoop@f problems with the log system and transfer is the CAM-I. We can not use one CAM-I for multiple services. To solve this, we propose to change the CAM-I (see Appendix G.2). The Service Advertisement Container is now composed by a list of Service Advertisment Container and RSU addresses (IP and MAC):

```
ServiceAdvertismentContainer ::= SEQUENCE {
  rsuMacAddress RsuMacAddress,
  rsuIpAddress RsuIpAddress,
  ads ServiceAdvertisments
}

ServiceAdvertisments ::= SEQUENCE (SIZE(0..7)) OF ServiceAdvertisment
```

Since the RSU's addresses will not change from a Service to another, they can be defined globally for each Services. By making a list of 0 to 7 Services, we can have one Service per channel (one Control Channel (CCH) and six Service Channels (SCH)). To save more bits, the three containers of the container are optional, and the IP address is modified:

```
RsuIpAddress ::= CHOICE {
  rsuipv4Andv6Address Ipv4Andv6,
  rsuiPv4Address IPv4Address,
  rsuiPv6Address IPv6Address
}

Ipv4Andv6 ::= SEQUENCE {
  rsuiPv4Address IPv4Address,
  rsuiPv6Address IPv6Address
}
```

In Scoop@f project

```
RsuIpAddress ::= SEQUENCE {
  rsuIPv4Address IPv4Address OPTIONAL,
  rsuIPv6Address IPv6Address OPTIONAL
}
```

The proposed version

With the proposed solution, the IP Address is more coherent and take less space to encode. We can save some other bits with the Advertised Service ID, by passing from 8 bits to 3, by reducing the range to [0..7], and 5 other bits with the size of the Service Access Capabilities from 8 bits to 3. The last reduction was made with the number of Satellite locally available: we were in 8 bits with 255 possible satellites, and now we have only 30. We chose this number because with the actual GPS satellites, we can not have more than 12 satellites at the same time, and the Galileo system will have 30 satellites, 6 of which are spare parts.

For the BTP port issue, we just need to set the destination port info to an unused value (e.g. 1).

For the TLogs and ULogs problem, instead of making 6 different ASN.1 definitions, we propose to make only one definition, which define all kind of logs, and which we can be extended to other project logs (see Appendix G.3, where we only show for the TLog optimization, but the ULog can be added in the same way): we have LogFile composed by a LogHeader and some Logs (a collection time and the data. Putting the collection time directly here permit to be sure that it will be in the first position, and not have to search for it):

```
TLogFile ::= SEQUENCE {
  header TLogHeader,
  logs   TLogs
}

TLogs ::= SEQUENCE OF TLog

TLog ::= SEQUENCE {
  collectionTime TimestampIts,
  log            TLogType
}
```

With this definition, the TLogHeader is always with the same length, and so, by decoding it, we can get the source with the Actor structure:

```
Actor ::= SEQUENCE {
  projectID ProjectID,
  actorID   ActorID
}

ProjectID ::= INTEGER {
  unavailable (0),
  scoopf      (1)
} (0..16)

ActorID ::=INTEGER (0..16)

Scoop-ActorID ::= INTEGER {
  unavailable (0),
  dirif       (1),
  dira        (2),
  diro        (3),
  sanef       (4),
  ld38        (5),
  cd22        (6),
  cd35        (7)
  stBrieucAgg (8),
  rBZH        (9),
  renault     (10),
  psa         (11)
} (0..16)
```

The first 4 bits are used to select the project (Scoop@f, InterCor, etc.) and the 4 last to specify the source of the ITS-S, and so, we can have 16 projects with 16 sources inside: 256 different sources. We could also specify the type of log (TLog or ULog), but we potentially divide by two the number of sources. When we have decoded the Header, we can adapt the codec to select the good ASN.1 definition of the logs. Another point to see is the CAM/CAM-I or DENM log. In Scoop@f, we select some data and put them in the TLog, with some possible error when translating them into a human readable system, and we have two TLogs per message: on sending and reception, which double the possible errors. We propose to put directly the CAM/CAM-I in a CAM-I TLog and DENM in a DENM TLog, with a boolean to know if we are sending or receiving the message. By doing this, we can not make mistake by omitting data, and if we need more data, we have them since we have the entire message. For all TLog divided in two TLog depending on the sending or reception method, we add the boolean and aggregate the TLogs. Many other optimizations can be done in the data values definition, but in Figure 4.9, we can see different size of the biggest TLogs. We can see that our solution (the dark blue bars) are less or equal to the Scoop@f one and that they can be very finer when CAM-I is used, because instead of generating up to 7 CAM-I, we generate only one CAM-I. But even with one service, our CAM-I is lighter than the Scoop@f CAM-I.

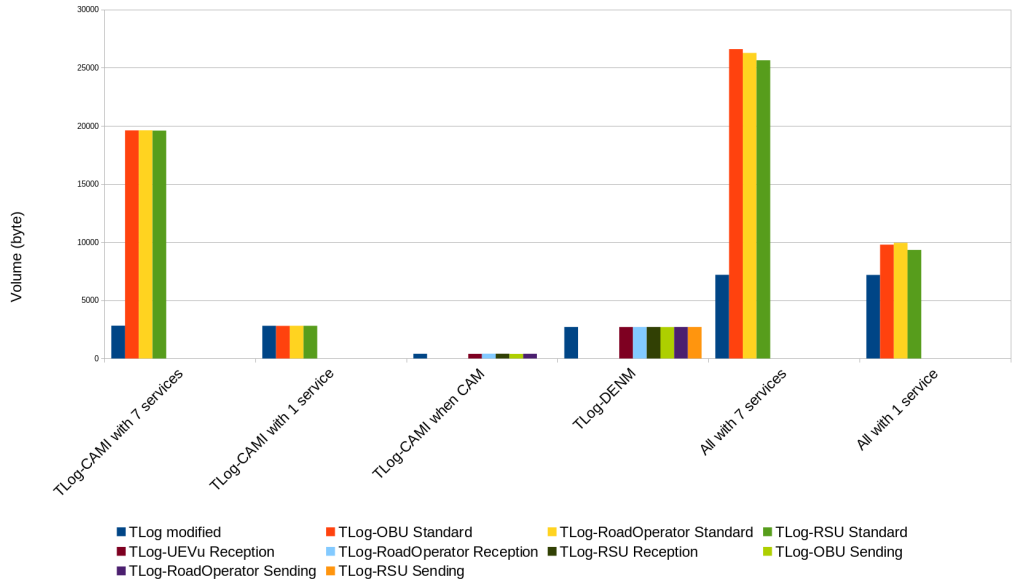


Figure 4.9 – Comparison between the Scoop@f TLogs and proposed TLogs sizes

In this chapter, we have seen how we implemented the ETSI ITS stack, and how we validate ITS implementations in the Scoop@f project in order to get VANET values to analyze. The validation of ITS is made in two main steps: the extraction of properties that need to be tested from standards, the tests of these properties. The tests are first done on laboratory with emulated environments, in order to manage the behavior more easily than in real situations. When the implementation successfully pass the laboratory tests, it is tested on road with a real environment. To do that, we need to implement log systems to get values of the tested properties.

In the next chapter, we will see our Big Data analysis methodology using VANET data from the Scoop@f project.

Chapter 5

Driver's Behavior Extraction

We have seen how VANET generate data (Chapter 2), how to use them with Big Data (Chapter 3), and we got some data from the Scoop@f project (Chapter 4). We will now see a new methodology that we propose to extract some exemplars, and why we use it with VANET data. Here, our exemplars are driver's behavior, i.e. how they manage different dangerous situations. If we can do that, it will raises privacy issues: the vehicle can change its pseudonym, but the driver can not change his behavior on the road, and he is still traceable. Since our methodology uses k Nearest Neighbors principles, we will first see a review of different methodologies using the k Nearest Neighbors principle in Section 5.1, then the proposed methodology based on local density in Section 5.2, its complexity and an implementation of the algorithm in Section 5.3, and finally, a use case from an InterCor experimentation in Section 5.4.

5.1 Overview of Techniques using the k Nearest Neighbors

Techniques using the Nearest Neighbors for data classification and sampling are often used since decades [76]. Here we will see an overview of some of them: the first two k Nearest Neighbors algorithms, which are used by most of the others algorithms, and the methods, which use the principle of local density, since we use it in our methodology.

5.1.1 k Nearest Neighbors [77]

The k Nearest Neighbors (kNN) technique is the simplest way to categorize a data. It uses a training set $\Omega = \{x_1, x_2, \ldots, x_n\}$, with x_i data from different classes. We define a way to calculate the distance between the data: if the distance between two data is small, it means that they are similar. When we want to classify a new incoming data, we calculate the distance between it and all the data from the training set. We order them, and we take the k nearest data. We count how many times classes are represented by these

data, and the one which is the most represented is the class of our incoming data. Of course, if there is a tie, we need to choose the class with different techniques: randomly, the first class to appear, etc.

We can illustrate the kNN methodology with a simple example: imagine that we have two classes, with the appropriate training set, the DC Comics™ games and the Marvel™ games. If I take a game (illustrated with question mark in Figure 5.1), is it a DC Comics™ game or a Marvel™ game? That distance differ from a use-case to another. According to the Figure 5.1, the nearest neighbor is DC Comics™. So, with a 1-NN technique, the game is a Marvel™ game. But if we take the 3 or 5-NN, the game is a DC Comics™ one.

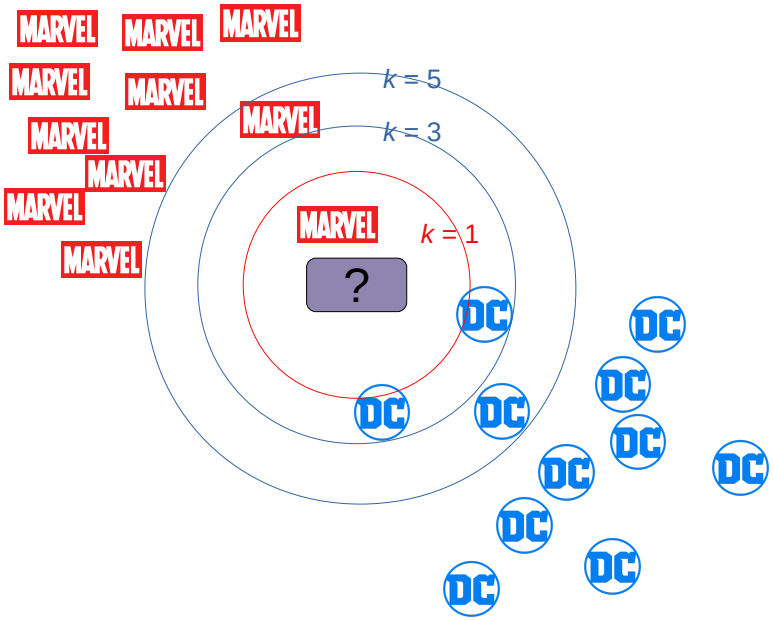


Figure 5.1 – Utilization of the kNN technique to determine the game category

As we can see, this technique is strongly impacted by the choice of the k value. Moreover, more we increase the k value, more the probability of error increases; for example, if we choose a k value of the number of data in the training set, the incoming data will be categorize in the most represented class of the dataset, which is not necessarily the good one.

5.1.2 Weighted k Nearest Neighbor [78]

The Weighted k Nearest Neighbor (WkNN) technique is based on the kNN mentioned above, but add a distance-weighted function. That weight is then

calculated between the incoming data and all the members of the training set. We then use the kNN technique with the distance-weighted computed to categorize the new data. In the original paper, the author proposed three functions to compute weights between the incoming data and the i-th data of the training set:

$$w_i = k - i + 1 \tag{5.1}$$

$$w_i = \frac{1}{d_i}, d_i \neq 0 \tag{5.2}$$

$$w_i = \begin{cases} \frac{d_k - d_i}{d_k - d_1}, & d_k \neq d_1 \\ 1, & d_k = d_1 \end{cases} \tag{5.3}$$

Where:

- w_i is the distance-weight between the unclassified data and the classified i-th data from the training set,
- k is the number of the wanted nearest neighbors,
- i is the classified i-th data of the training set,
- d_i is the Euclidean distance between the unclassified data and the classified i-th data of the training set,
- d_k is the Euclidean distance between the unclassified data and the classified k-th data of the training set.

Equations (5.1) and (5.2) are easier to compute, but they give a more probability of error: "[Equation (5.1)] has a clear drawback of not being able to adjust the distribution of weights, depending on closeness of the neighbors to the unknown sample", and "For [Equation (5.2)]..., the relationship between the distance d_i and the corresponding weight w_i is such that the weights w_i take very large values for distances d_i close to zero, and thus reduce the corresponding classification algorithm in many cases to a simple nearest-neighbor rule", i.e. the kNN mentioned before.

As we can see, the result will be mainly influenced by the distance-weight calculation technique, improve the efficiency of the kNN methodology with the third presented way to compute the distance-weight. That methodology is used in many other methodologies, like the Modified kNN [79], which improve accuracy of WkNN, or can be used for ordinal classification [80].

5.1.3 k Nearest Neighbors Model-Based [81]

The k Nearest Neighbors Model-Based (kNNModel) models the training dataset using local density, and when a new data is inserted to the dataset, the data is compared with the model entries, and is classified with the most similar model entry class. The modeling is made as follows:

1. Select a similarity measure (an other possible name for the distance) and create a similarity matrix from the given training dataset

2. Set to "ungrouped" the tag of all data tuples

3. For each "ungrouped" data tuple, find its largest local neighborhood, which covers the largest number of neighbors with the same category

4. Find the data tuple x_i with a largest global neighborhood N_i among all the local neighborhoods, create a representative $\langle Cls(x_i), Sim(x_i), Num(x_i), Rep(x_i)\rangle$ into M to represent all the data tuples covered by N_i, and then set to "grouped" the tag of all the data tuples covered by N_i

5. Repeat steps 3 and 4 until all the data tuples in the training dataset have been set to "grouped"

6. Model M consists of all the representatives collected from the above learning process

The representative $\langle Cls(x_i), Sim(x_i), Num(x_i), Rep(x_i)\rangle$ respectively represents the class label of x_i, the lowest similarity to x_i among the data tuples covered by N_i, the number of data tuples covered by N_i, and a representation of x_i itself. To clear up potential ambiguities in step 4, if some neighborhoods have the same maximal number of neighbors, the chosen one is the one with minimal value of $Sim(x_i)$ (i.e. the one with the highest density).

To classify the new data, we use the following algorithm:

1. For a new data tuple d_t to be classified, calculate its similarity to all representatives in the model M

2. If d_t is covered only by one representative $\langle Cls(x_i), Sim(x_i), Num(x_i), Rep(x_i)\rangle$, i.e. the distance of d_t to d_j is smaller than $Sim(d_j)$, d_t is classified as the category of d_j

3. If d_t is covered by at least two representatives with different category, classify d_t as the category of the representative with largest $Num(d_j)$

4. If there is no representative in the model M covers d_t, classify d_t as the category of a representative, which boundary is closest to d_t

That algorithm has the advantage that the k value is selected implicitly by the model construction process, but has the disadvantage to have bad accuracy on marginal data, which fall outside the regions of representatives. Model-Based algorithms was originally used for text categorization, but is also used as a defense against profile injection attacks of collaborative recommender systems [82].

5.1.4 Locally Nearest Neighbors [83]

To explain the Locally Nearest Neighbors, we need to explain the Nearest Feature Line (NFL). The NFL was originally made for Face Recognition, and adds a new distance: the Feature Line (FL) distance. For each couple of points, we "draw" a line between them, and we project the unclassified point to this line. Since the projection is made perpendicularly, we use the dot product to calculate the distance:

$$\begin{gathered}(p-x)\dot{(}x_2-x_1) = [x_2+\mu(x_2-x_1)-x]\dot{(}x_2-x_1) = 0 \\ \mu = \frac{(x-x_1)\dot{(}x_2-x_1)}{(x_2-x_1)\dot{(}x_2-x_1)}\end{gathered} \tag{5.4}$$

When $\mu = 0$, the projection $p = x_1$; when $\mu = 1$, $p = x_2$; when $0 < \mu < 1$, p is an interpolating point between x_1 and x_2; when $\mu > 1$, p is a forward extrapolating point on the x_2 side; when $\mu < 0$, p is a backward extrapolating point on the x_1 side.

The NFL distance is calculated by taking the minimum FL.

$$d(x, \overline{x_{i^*}^{c^*} x_{j^*}^{c^*}}) = \min_{1 \leqslant c \leqslant C} \min_{1 \leqslant i < j \leqslant N_c} d(x, \overline{x_i^c x_j^c}) \tag{5.5}$$

where:

- i^*, j^* are the best matched patterns of the class
- c^* is the best matched class
- c is the x_i and x_j classes
- C is the number of classes
- N_c is the number of pattern feature points to represent the class c

This methodology is a good methodology for face recognition purpose, but performs also on image, texture, and audio classification and retrieval.

The Locally Nearest Neighbors focuses only on the local NN prototypes of the query point instead of all the patterns in each class.

5.2 A New k Nearest Neighbors Methodology Using the Local Density

All these previous methodologies are used when we have a training set and a new data to classify. Our methodology is new because we use the local density of a data to extract samples, and does not need training set to work. A set of exemplars, also called samples, is a classical way for storing and representing the cognitive structures [84]. The exemplars are real data extracted from a large dataset unlike the prototypes that are artificial data such as the statistics. Thus, the selection of few exemplars that represent the whole dataset is one of the first step when exploring a dataset. For instance, the selection of exemplars is central to several clustering methods [85]. The selection of exemplars is a case-oriented process, which is also called sampling [86]. The goal is to extract a small subset of representative data from the dataset, the driver's behavior in our case.

5.2.1 Sampling Method

Let Ω be a dataset with n data defined by:

$$\Omega = \{X_1, X_2, X_3, ...X_n\}$$

The goal of sampling is to select a subset of Ω. We call this subset Σ:

$$\Sigma = \{Y_1, Y_2, Y_3, ...Y_p\} \text{ with } Y_j \in \Omega$$

When sampling, p is much smaller than n ($p << n$) and Y_j (with $1 \leqslant j \leqslant p$) is a representative or exemplar of Ω.

Our method is based on an estimation of the local density in a neighborhood of each data. The first exemplar we select is the one with the highest local density. Then the nearest neighbors of this exemplar are removed from Ω. We obtain the following exemplars by iterating the process until the set of remaining data is empty.

Since our method is based on the local density of the neighborhood of each data, we decided to call it Fuldon, which stands for Fuldon Uses Local Density Of Neighborhood.

5.2.1.1 Local Density

Here, we only consider multidimensional quantitative data. Thus, X_i with $1 \leqslant i \leqslant n$ is a vector defined by:

$$X_i = (v_1(i), v_2(i), ...v_p(i))$$

where v_1, v_2,... v_p are the p variables that are the features of data. In this context each data lies a p-dimensional data space.

Classically the density is defined using a unit hypervolume. For instance, the hypersphere of radius α can define the unit hypervolume. In the data space, the local density at X is then equal to the number of data of Ω lying inside the unit hypersphere centered in X. Unfortunately, the definition of density comes up against the curse of dimensionality [87]. When the dimension of the data space increases, the volume of the available data becomes sparse and the classical definition of density has no meaning. For circumventing this drawback, we define the density for each variable using the approach of parallel coordinates [88] (see Figure 5.2). So, we have p densities each defined in a one-dimensional space. The sum of these densities gives us a density-based index we use in the whole data space.

Let us define the density computed in the one-dimensional space of the variable v_j (with $1 \leqslant j \leqslant p$). The dataset Ω is projected in this space, and we obtain n values with:

$$\Omega_j = \{v_j(1), v_j(2), v_j(3), ...v_j(n)\}$$

These values are in the range $[min_j, max_j]$ where $min_j = \min_{1 \leqslant i \leqslant n} (v_j(i))$, and $max_j = \max_{1 \leqslant i \leqslant n} (v_j(i))$. Let us define the unit interval that we use to compute the density at each value x. Let k be an integer between 1 and n. If we expected a local density equal to k, then the length α_j we propose for the unit interval is equal to $\alpha_j = \frac{max_j - min_j}{n} \times k$. Thus, the local density at x is equal to the number of elements of Ω_j that are in the unit interval $[x - \alpha_j/2,\ x + \alpha_j/2]$. The local density at X_i for the variable v_j is then defined by:

$$density_j(X_i) = \#\{\ [v_j(i) - \alpha_j/2,\ v_j(i) + \alpha_j/2]\ \cap\ \Omega_j\ \}$$

Finally, the local density at X_i for all the variables is defined by:

$$density(X_i) = \sum_{1 \leqslant j \leqslant p} density_j(X_i)$$

We select the data which has the highest local density. This data is the first exemplar of Ω:

$$Y_1 = \arg\max_{X_i \in \Omega} \quad density(X_i)$$

5.2.1.2 Nearest Neighbors

The previous procedure permits us to select only one exemplar. We obtain the following exemplars by reducing the dataset and iterating this procedure. The dataset is reduced by removing Y_1 and its nearest neighbors.

Let us describe our definition of the nearest neighbors of a data X in a dataset Ω. The neighbors of X_i for the variable v_j are the data of Ω that are in the unit interval centered in X_i. This neighborhood N_j is defined by:

$$N_j(X_i) = \{X_k \in \Omega \text{ with } v_j(k) \in [v_j(i) - \alpha_j/2,\ v_j(i) + \alpha_j/2]\}$$

The nearest neighbors of X_i for all the variables should be in the neighborhoods for each variable. Thus, the nearest neighbors of X_i are in the neighborhood N defined by:

$$N(X_i) = \bigcap_{1 \leqslant j \leqslant p} N_j(X_i)$$

To select the second exemplar Y_2 we exclude the first one Y_1 and its nearest neighbors $N(Y_1)$. We apply the procedure defined in the previous section within a reduced dataset $\Omega \setminus N(Y_1)$. Then Y_2 is the data with the highest local density within the reduced dataset.

We iterate the procedure until the reduced dataset is empty. The exemplars we obtain gives us the samples of Ω.

5.2.1.3 Number of Exemplars

We set our method of sampling using the parameter k where k is an expected local density at each data. The value of k lies between 1 and n when the dataset has n data. Let us consider a toy example with 200 simulated data ($n = 200$) with 5 variables ($p = 5$). Figure 5.2 uses parallel coordinates [88] to display the profiles of these data with 200 dashed broken lines. The exemplars are selected using the parameter value $k = 100$. We obtain 7 exemplars (bold broken lines in Figure 5.2).

The number of selected exemplars decreases when the parameter value k increases. Figure 5.3 shows that the number of selected exemplars decreases from 200 to 1 when the density parameter k increases. This property of our method permits us to adapt a strategy to select the number of samples that we extract from the dataset. If we want a specific number of samples selected from the initial dataset, then we can adjust the parameter k to obtain the expected number of exemplars.

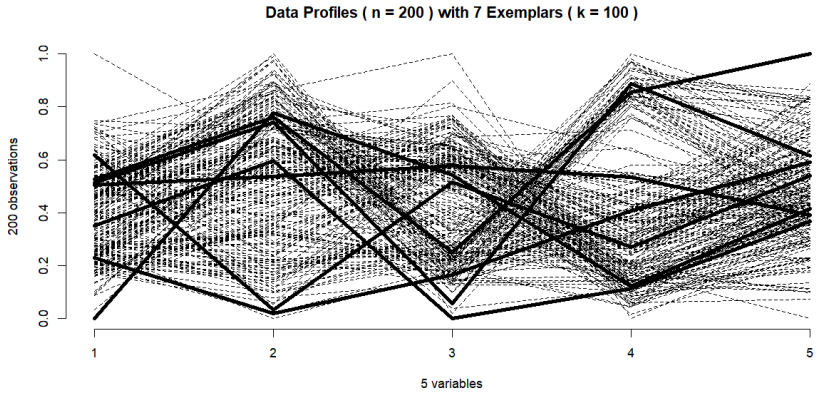


Figure 5.2 – Profiles of 200 simulated data with 5 variables (dashed lines), with the election of 7 exemplars with a parameter value $k = 100$ (bold lines)

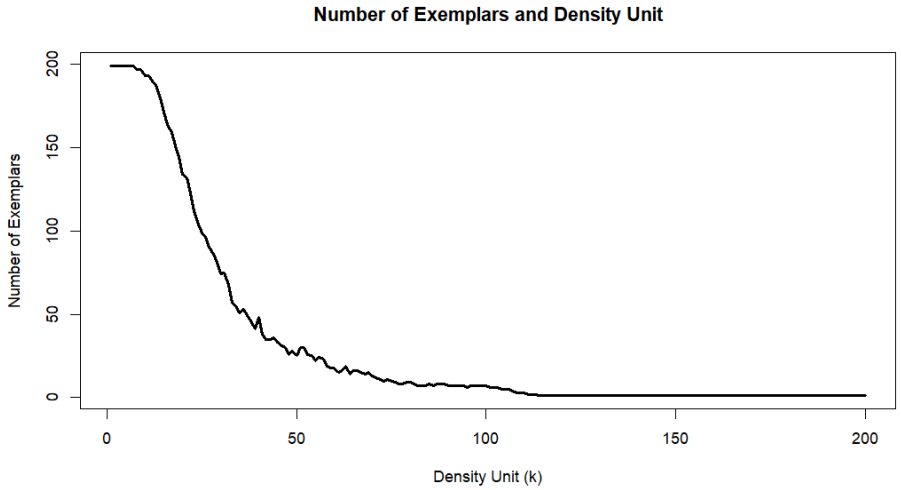


Figure 5.3 – Number of selected exemplars decreases from 200 to 1 when the density parameter k increases from 1 to 200

5.2.2 Assessment of Sampling

The exploratory analysis of a dataset is complex for many reasons. The dataset is often divided into classes but the distribution of these classes is unknown. Moreover, the number of these classes is also unknown. To better understand data, the use of a complementary exploratory trial on a smaller dataset is often necessary. The selection of a reduced number of samples should then represent all the classes of the dataset. For this reason, we will

Number of classes	Distribution in dataset ($n = 200$)	Number of selected exemplars	Exemplars distribution between classes
4	(42, 51, 65, 42)	25	(4, 8, 8, 5)
4	(42, 51, 65, 42)	18	(5, 6, 4, 3)
4	(42, 51, 65, 42)	13	(3, 3, 5, 2)
4	(42, 51, 65, 42)	9	(2, 3, 3, 1)
4	(42, 51, 65, 42)	7	(2, 2, 2, 1)
4	(7, 103, 62, 28)	10	(1, 3, 4, 2)
5	(40, 47, 55, 9, 49)	10	(2, 3, 2, 2, 1)
6	(6, 12, 76, 80, 24, 2)	9	(2, 1, 2, 1, 2, 1)
7	(23, 16, 51, 46, 1, 36, 27)	8	(1, 1, 1, 2, 0, 1, 2)
8	(37, 9, 3, 19, 48, 12, 45, 27)	9	(3, 0, 0, 1, 1, 1, 3, 1)

Table 5.1 – Distribution between classes within a dataset ($n = 200$) and within the selected exemplars

evaluate our sampling method under controlled conditions when the distribution of the classes is known. But of course, the method remains designed for applications in exploratory analysis when the classes are unknown. This method is particularly useful when classes have large overlapping and when the classes have very different numbers of data. In such cases the classical methods of clustering very often fail.

Let us consider a dataset with known distribution of classes for assessing our sampling method. We verify that the distribution of the selected exemplars between classes remains comparable with the distribution of data within the initial dataset. Table 5.1 gives the results we obtain with some simulations.

In the first five rows of the table, we use the dataset displayed in Figure 5.2. This dataset is simulated using four classes with a large overlapping. The 200 data are randomly distributed between these classes. (42, 51, 65, 42) is the distribution between the four classes. The number of selected exemplars decreases when the parameter k increases. We obtain 25, 18, 13, 9 and 7 exemplars using respectively 50, 60, 70, 80 and 100 as values of k. In these five simulations, the four classes are effectively represented by the exemplars. But when k increases, the number of selected exemplars becomes too small for representing each class.

In the five last rows of Table 5.1 we simulate five datasets with respectively 4, 5, 6, 7 and 8 classes. The number of data in each class is randomly selected and it could be very different from one class to another one. The datasets

Name of dataset	n	p	Distribution in dataset	Number of exemplars	Distribution of exemplars
Iris	150	4	(50, 50, 50)	8	(3, 3, 2)
Wine	178	13	(59, 71, 48)	9	(2, 4, 3)
Glass	214	9	(70, 76, 17, 13, 9, 29)	19	(1, 6, 1, 5, 1, 5)
Haberman	306	3	(225, 81)	10	(7, 3)
Ecoli	336	7	(143, 77, 2, 2, 35, 20, 5, 52)	23	(3,9,0,0,3,3,2,3)

Table 5.2 – Distributions between classes with a real dataset and with selected exemplars (n = number of data, p = number of variables)

have 200 data and the parameter k is equal to 80 when selecting exemplars. When the number of classes increases, the number of exemplars becomes too small for representing each class (see the two last rows of the table). But these classes are represented if the number of selected exemplars increases (i.e. if we decrease the value of the parameter k).

Let us study the sampling with real datasets. We consider some datasets of UCI repository (see in [89]). Table 5.2 displays the selection of exemplars using our blind method (i.e. when the classes are unknown) on the classical dataset called “Iris”, “Wine”, “Glass”, “Haberman” and “Ecoli”.

These datasets have respectively 3, 3, 5, 2 and 8 classes. Our sampling method gives generally an exemplar in each class. Obviously, the method fails if the number of classes is high relative to the number of selected exemplars. Moreover, the method often fails if the number of elements within one class is very low. For instance, in the last line of Table 5.2, two classes have only 2 elements and these classes are not represented by the exemplars. But these classes can be represented by an exemplar if we increase the number of exemplars we select.

5.3 Complexity of the kNN Using Local Density

The algorithm is pretty simple: we have a loop to compute the different samples (see Appendix H for the pseudo-code of all the algorithms). Now we need to get the most “difficult” part: the sample selection. To do that, we first compute the Euclidean distance of data, for each variable, then we compute the neighborhood of each data for each variable, the density for each data and each variable, the sum of densities for each data, we check the denser to have the selected sample, the neighborhood for each variable of the sample data, and finally, the intersection of that neighborhood.

The samples algorithm is a succession of appending function (the ap-

pending of the selected sample to the list of selected samples), sample selection, removing the sample and its neighborhood from the dataset, and the normalization of the remaining data set. We can compute the complexity for each of them. We assume that the appending function is in $O(1)$. The purge of selected samples and neighborhood is in $O(N)$, and normalize is in $O(N \times p) = O(N^2)$.
The sample selection is a succession of Euclidean distance computation, neighborhood computation, densities of this neighborhood computation, sum of these densities, selection of the denser neighborhood, the neighborhood of each variable computation, the intersection the neighborhood. The Euclidean distance computation is in $O(p \times N \times N) = O(N^3)$.
The neighborhood computation is in $O(p \times N \times N) = O(N^3)$.
The densities computation is in $O(p \times N \times N) = O(N^3)$.
The sum of densities is in $O(N \times p) = O(N^2)$.
The denser neighborhood selection is in $O(n)$.
The neighborhood for each variables of the denser neighborhood is in $O(N \times p) = O(N^2)$.
The intersection computation is in $O(N \times p) = O(N^2)$.
And so, Samples is in

$$\begin{aligned} & O(N^3) + O(N^3) + O(N^3) + O(N^2) + O(N) + O(N^2) + O(N^2) \\ = \; & 3O(N^3) + 2O(N^2) + O(N) \\ = \; & O(N^3) \end{aligned}$$

Samples is so in

$$O(N) + O(N^2) + O(N^3) = O(N^3) \tag{5.6}$$

With N the number of observations in the dataset, and p the dimension of these data.

5.3.1 Algorithm Implementation

The algorithm was first implemented in R with the Shiny package for the visualization in a parallel coordinates and nodes network view. R is the most used language for Big Data analysis purposes [7], and uses variables as vector. The choice of this language was evident for our purpose, because we uses matrix and vector of matrix. With R, we can easily use them. For example, the purge of a selected exemplar and its neighborhood is literally made by substract the vector the exemplar and neighborhood from the matrix:

```
numbers <- all[-remove] # numbers gets all the
                        # data numbers "-" what
```

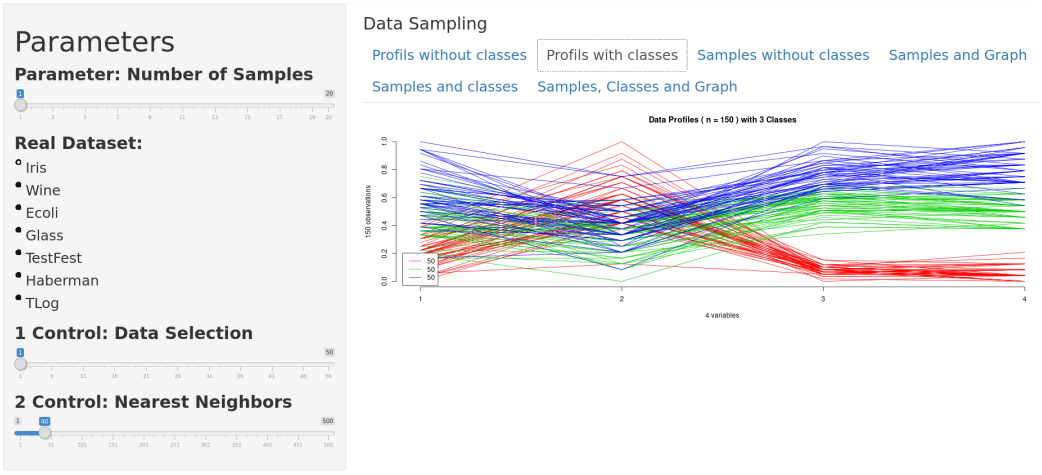
```
                          # we need to remove
rest <- data[number,]     # the rest of the data
                          # are the remaining data,
                          # i.e. the numbers data
```

With the Shiny package, we can make an online demonstrator. The interface is a web page, and the server will execute the R scripts and print result on the web page. In the Figure 5.4a we have the data presented in parallel coordinates, where each line is a data with its variables values; the Figure 5.4b is the same curves, but with a colorization for each class; the Figure 5.4c presents the selected samples, and the Figure 5.4e the samples and their corresponding classes with their distributions; finally, the Figure 5.4d presents the samples with their neighborhood in a nodes network view, where central nodes are the samples while Figure 5.4f adds the classes distributions. The R/Shiny application offers a good visualization, but has some bad time executions as we can see in Figure 5.5, with also 4 seconds to execute the algorithm on the small Iris dataset when $k = 1$. In these curves, we have the different k values in abscissa and time value in ordinate. To be sure that it is not the algorithm but the implementation and the Shiny package, which delayed the execution time of an algorithm with a complexity of only $O(N^3)$, the algorithm was also implemented in C++ language, and the time execution is much better as we can see in Figure 5.6. The time division mean is about 400 as we can see in Table 5.3.

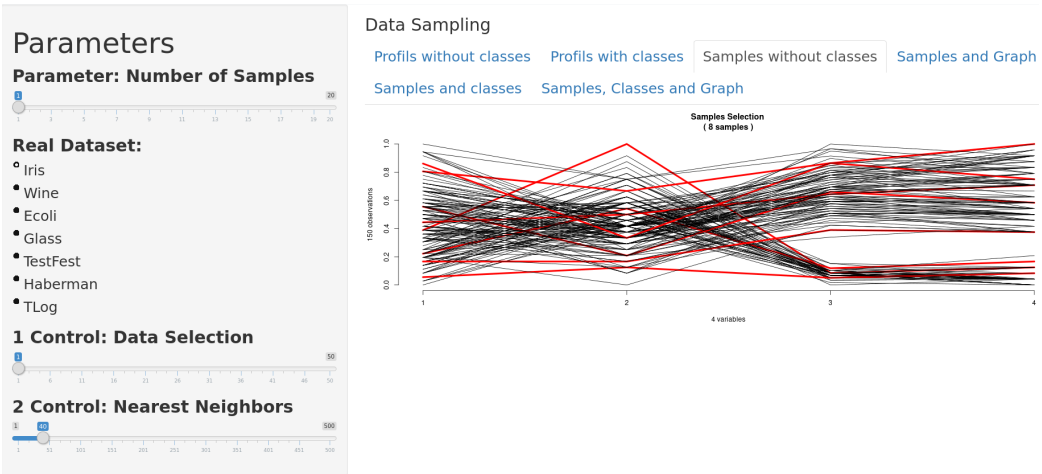
Dataset name	**Number of data**	**Number of variables**	**Time division**
Iris	150	4	105.021
Wine	178	13	553.092
Ecoli	336	7	190.671
Haberman	214	9	472.266
Glass	306	3	695.775
TestFest	58	95	308.382
TLog	3201	17	424.351

Table 5.3 – Time division of the execution when using the C++ version

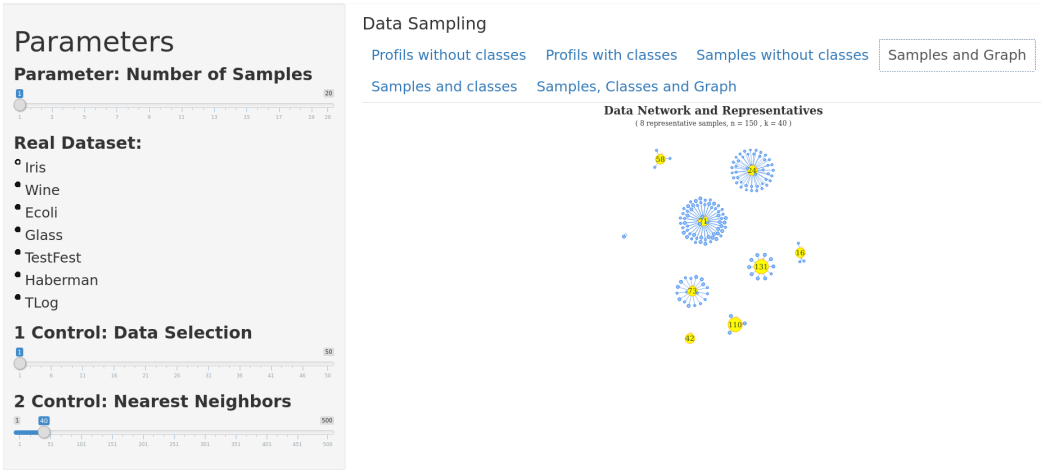
(a) Data presentation



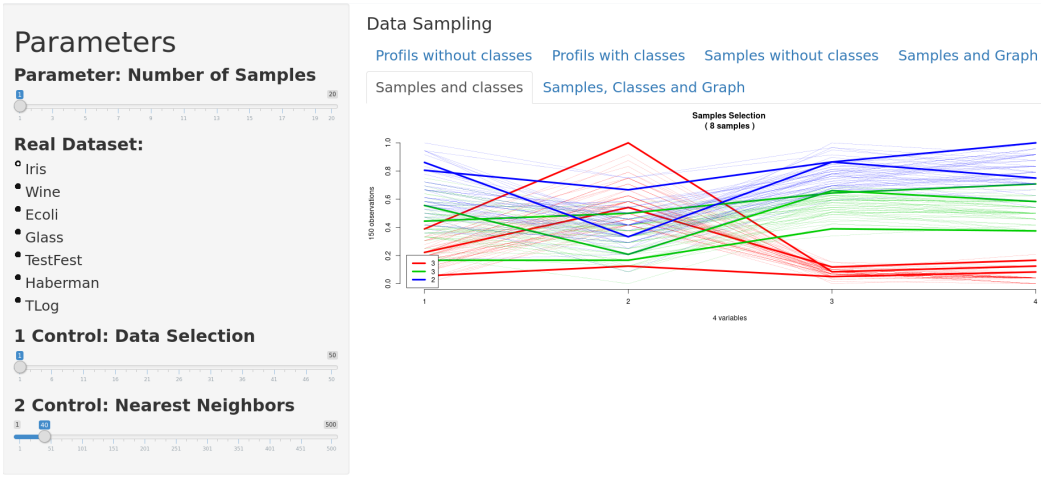
(b) Data presentation with classes



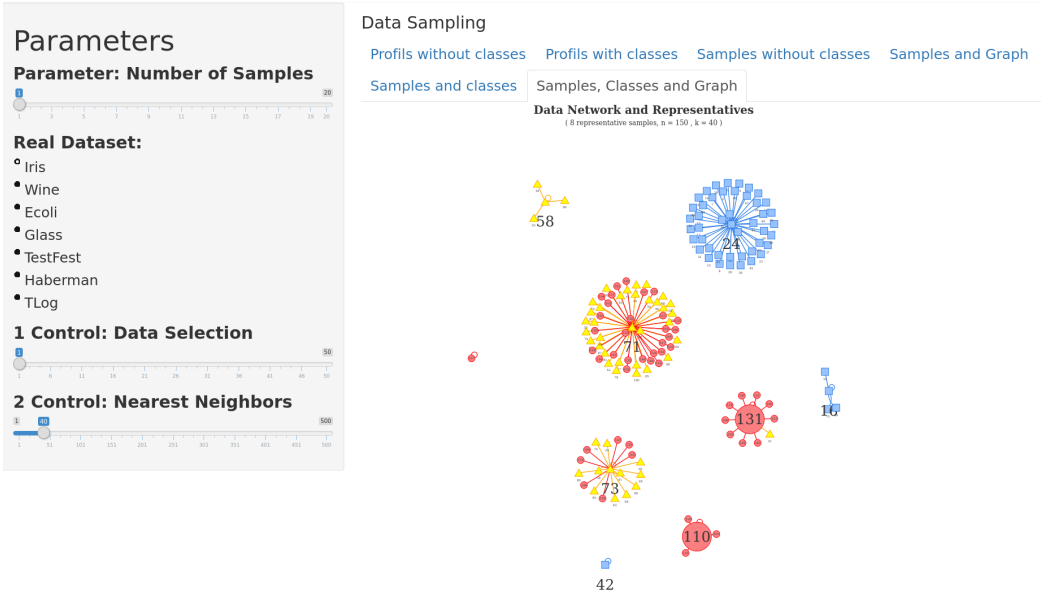
(c) Samples without classes



(d) Samples without classes in a nodes network view

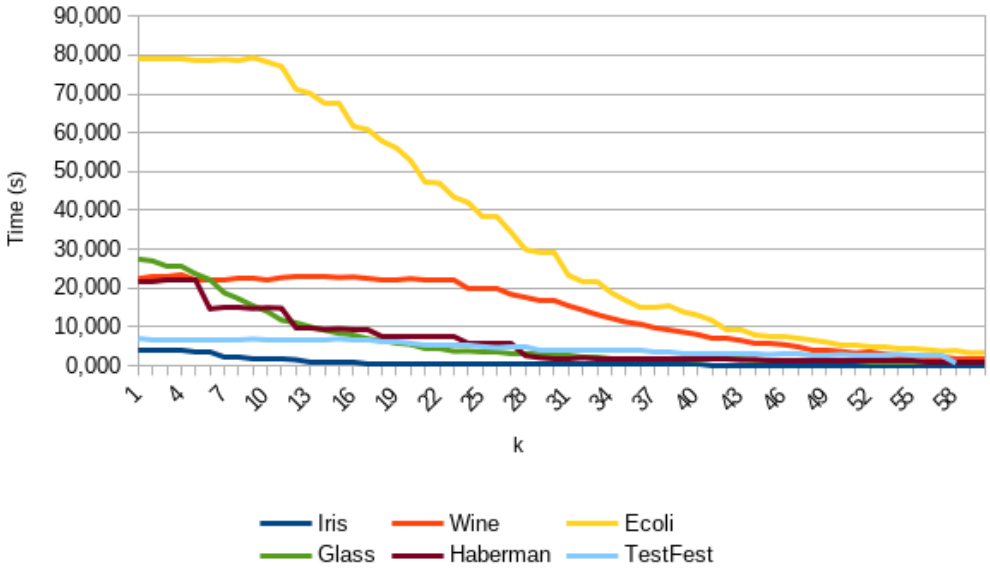


(e) Samples with classes

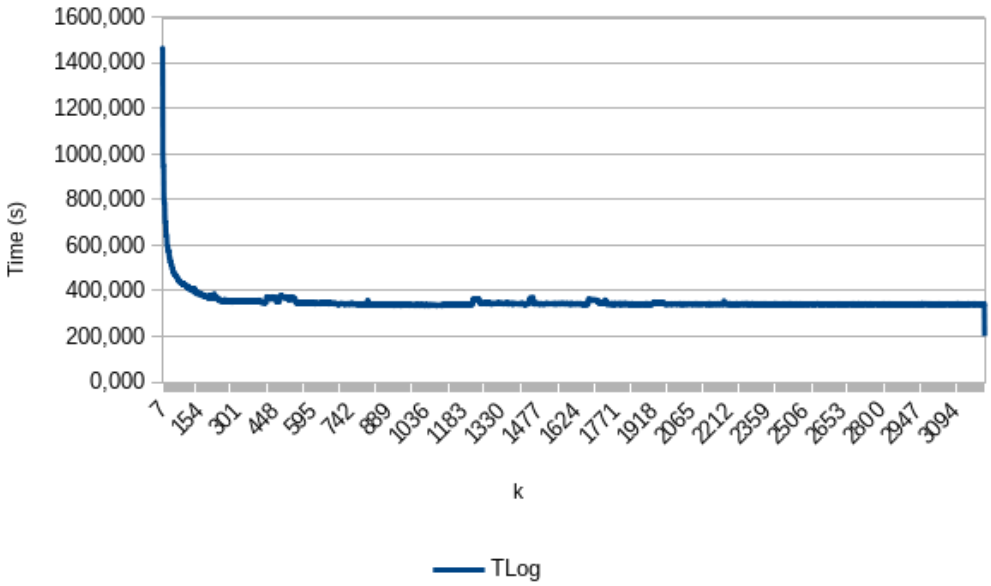


(f) Samples with classes in a nodes network view

Figure 5.4 – R/Shiny implementation of the algorithm

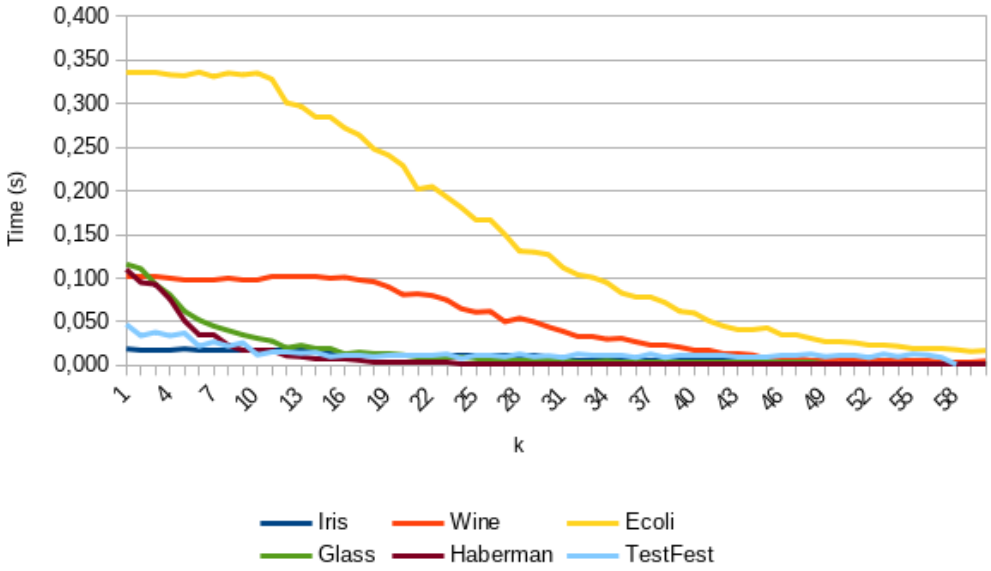


(a) Time execution with R/Shiny on small datasets (60 firsts k values)

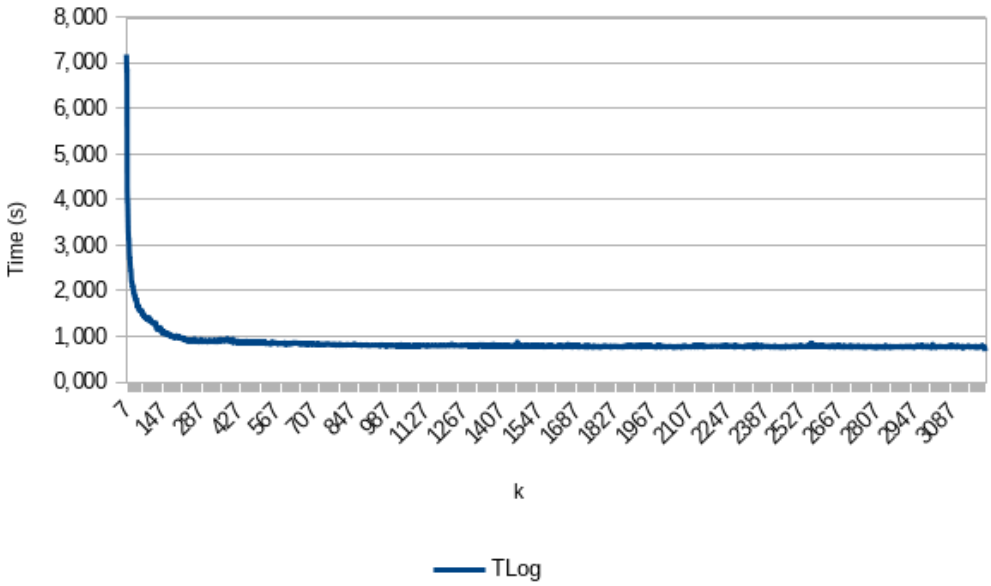


(b) Time execution with R/Shiny on large datasets

Figure 5.5 – Time execution with R/Shiny



(a) Time execution with C++ on small datasets (60 firsts k values)

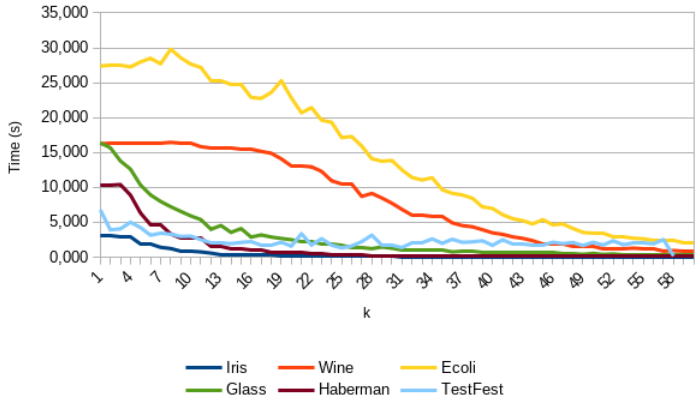


(b) Time execution with C++ on large datasets

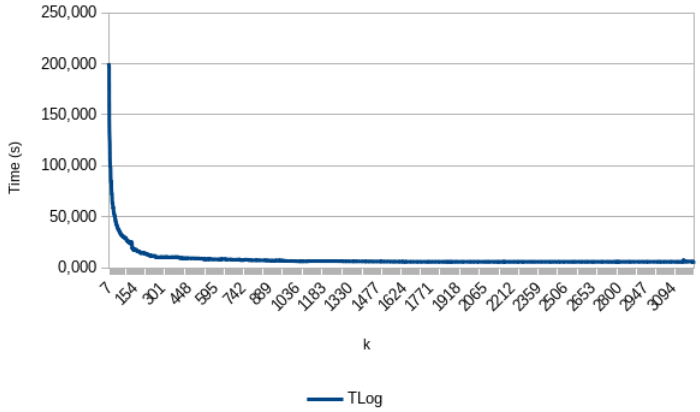
Figure 5.6 – Time execution with C++

5.3.2 Algorithm Parallelization

The proposed methodology is very easily parallelizable to reduce the execution time. Effectively, the algorithm is mainly composed by loops (*apply* function in R programming), and there is no concurrent access to data, and so, each one of steps can be parallelized. But if we proceed this way, we get the curves from Figures 5.7a and 5.7b, which are slower than the sequential ones. This is explained by the fact that we need to allocate the threads, and it is very expensive in terms of time and computation. A better solution is to parallelize the sample selection by distributing only the firsts nested loops, which give the curves from Figures 5.8a and 5.8b with 8 threads. With these 8 threads, we divided by two the execution time against the sequential one.

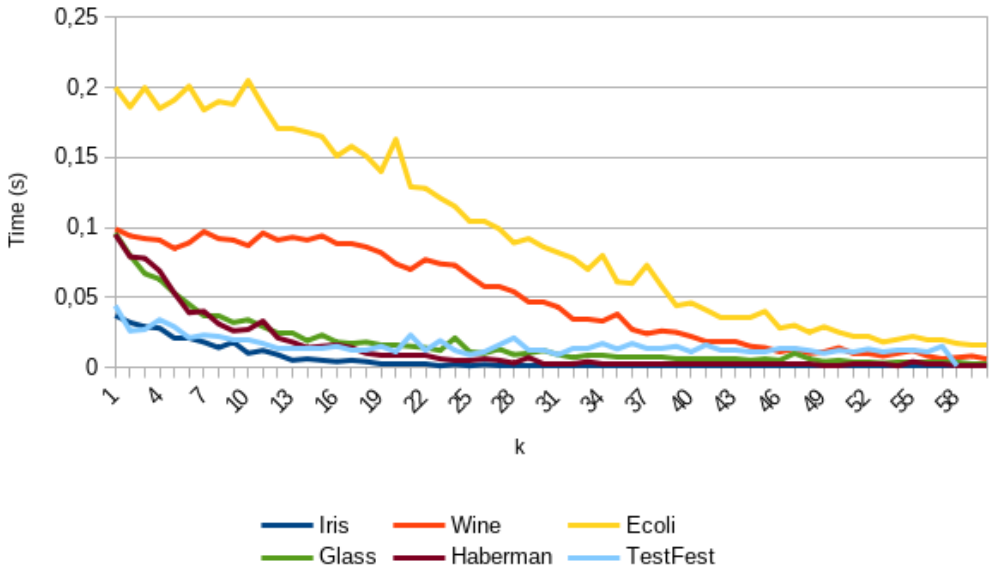


(a) Time execution with parallelization of all steps on small datasets (60 firsts k values)

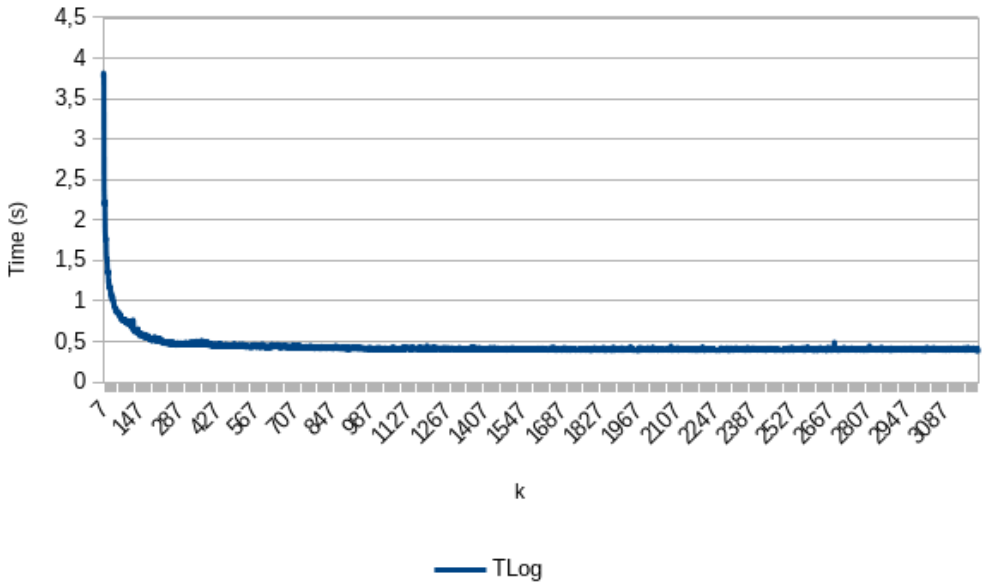


(b) Time execution with parallelization of all steps on large datasets

Figure 5.7 – Time execution with parallelization of all steps



(a) Time execution with parallelization of sample algorithm on small datasets (60 firsts k values)



(b) Time execution with parallelization of sample algorithm on large datasets

Figure 5.8 – Time execution with parallelization of sample algorithm steps

5.4 VANET From Real World Experimentation

5.4.1 On a Roadway Experimentation

In the Scoop@f [14] (Cooperative System at France) project, ITS are experimented in the real life. To do that, connected vehicles drive on roadway and communicate with the infrastructure or other vehicles via the ITS-G5. Messages used in Scoop@f are CAM and DENM. We first used our methodology with TLogs from a road test with a slippery road. These logs contain 3 201 data of 6 variables (the brake pedal usage: active or not, the steering wheel angle, the strength braking and 3 for the exterior lights). The exterior lights are defined by the left and right turn signal (warning is the combination of both), and daytime light.

Here we want to describe characteristics from the experimentation and trying to modeling vehicle behavior on this type of route. By using the 3 201 data we obtain the Figure 5.9. We then used our methodology with the $k \in \{40, 80, 100, 140, 180, 200\}$ that give us the Table 5.4. With $k = 40$, 142 samples are extracted, 102 with $k = 80$, 92 with $k = 100$, 58 with $k = 140$, 46 with $k = 180$, 48 with $k = 200$ and 18 with $k = 500$. With our methodology, a big preprocessing is made, dividing by $\{22, 31, 34, 55, 69, 6, 178\}$ the number of data to process.

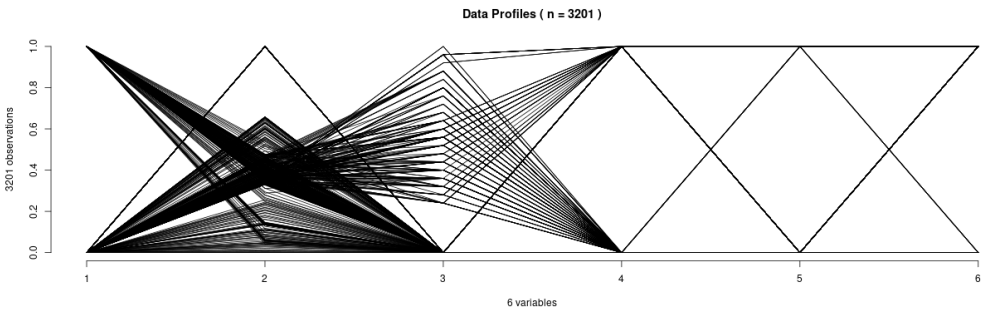


Figure 5.9 – Profiles of 3 201 data with 6 variables from roadway experimentation.

In order to illustrate the impact of variation of k to the number of representative samples, Figure 5.10 presents the obtained profiles while varying k. It can be seen that it is very difficult to distinguish exemplars among data. To provide a clearer view, Figures 5.11a and 5.11b present two ex-

k	Number of exemplars	Division
40	142	22
80	102	31
100	92	34
140	58	55
180	46	69
200	48	66
500	18	178

Table 5.4 – Selection of exemplars with different values of the parameter k from roadway experimentation.

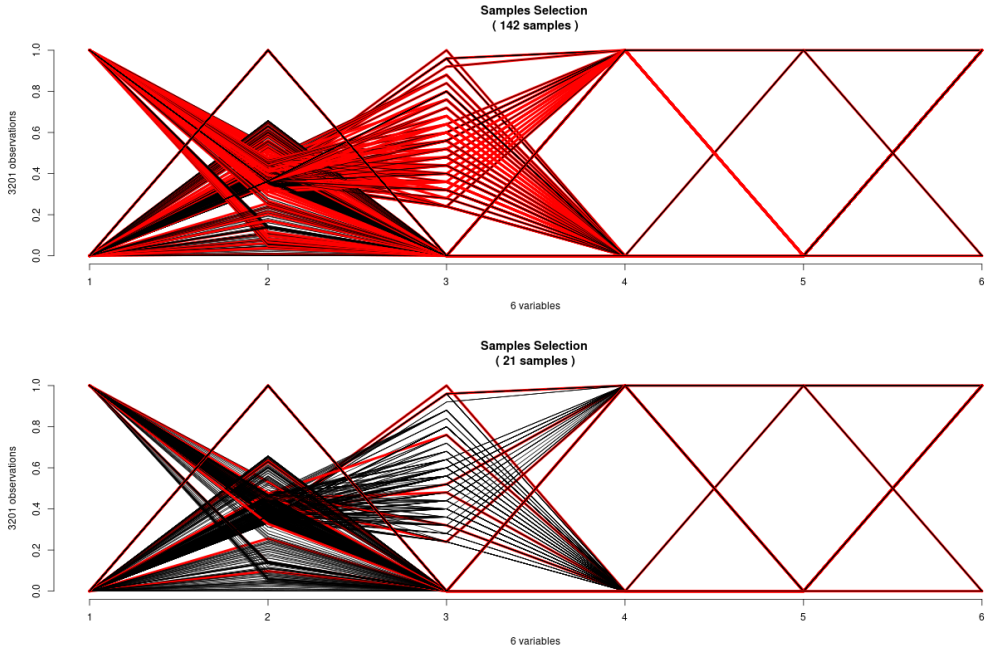
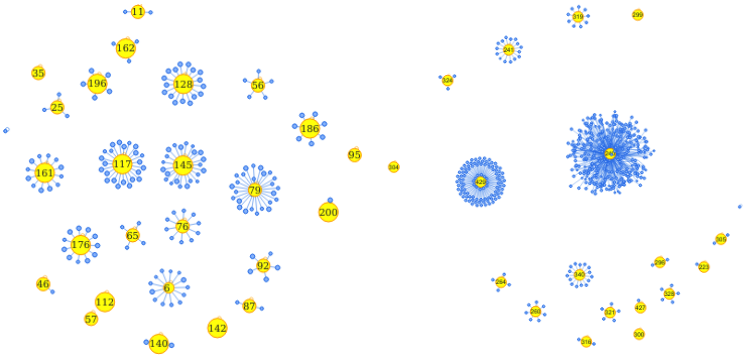


Figure 5.10 – Profiles of 3,201 data with 6 variables from roadway experimentation with varying values of k.

treme cases where one obtained with the smallest values (k=40) and another with the greatest value (k=500). In fact, these figures present the exemplars (yellow nodes) obtained with the algorithm and their connected neighbors (blue nodes). It can be seen that the number of exemplars can be limited depending on the need of use case. When the value of k increases, the number of selected exemplars decreases. It has to be noticed here that the most



(a) Data Network with k=40 (b) Data Network with k=500

Figure 5.11 – Data Network of extreme cases

important is to find the compromised tuning between the desired number of exemplars and the desired processing time and costs.

5.4.2 The InterCor TestFest Event

From the 23rd to the 26th of April 2018, the University of Reims Champagne-Ardenne (URCA), in France, organized the second InterCor [90] TestFest event. During this week, 18 European brands tested their connected vehicles on a 30 km track inside the city of Reims and on roadway around the city (see Figure 5.12). Unlike the first event at Dordrecht in Netherlands, the security layer was added. We took advantage of this event to collect real VANET data, and try to extract driver's behavior. To do that, we provided 9 tablets to 9 drivers from the InterCor project. These tablets saved periodically the vehicular position using the GPS.

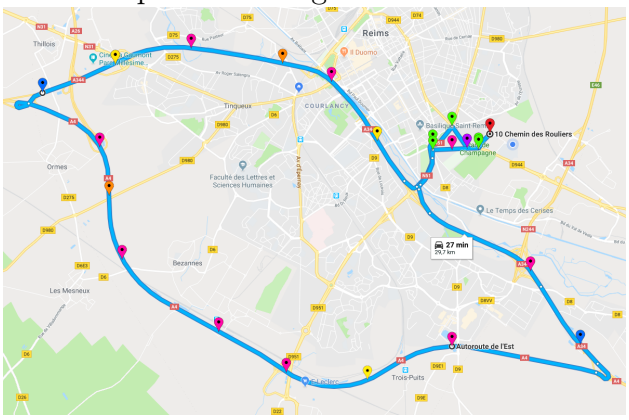


Figure 5.12 – TestFest track

1 begin of the track
5 traffic lights
2 RSUs
3 points between RSUs
9 DENM position
2 tolling zones
1 yield sign
1 end of the track

At the end of the event, we got back the tablets, and preprocessed the data, by selecting 24 strategic points on the track: the dangerous positions where a deceleration is needed. By doing this, we want to see if drivers react in the same way with dangerous situation. For each point, we took the timestamp from the beginning of the track, the vehicle speed, acceleration, and heading from the North. Since the first point is the start of track, its timestamp is set to 0, we did not use it, and we had $24 \times 4 - 1 = 95$ variables per record. With these 9 drivers, we got 58 observations, which correspond to 58 records with the distribution of records per driver as following (7, 8, 5, 2, 4, 9, 7, 3 ,13).

We applied our methodology with these data. In Figure 5.13, we present the resulting sample selection and their neighborhood for a k value of 48. Each number corresponds to a driver, and each driver has its own color and shape, so we can see that there is only a few samples which gather other drivers (i.e. nodes connected with an other color and shape). In the Table 5.5, we can see different values of k. When $k \geqslant 19$ (33 %), samples gather too many data, and so drivers are mixed together; and with $k \geqslant 48$ (82 %) some drivers are not represented (0 value in the sampling distribution).

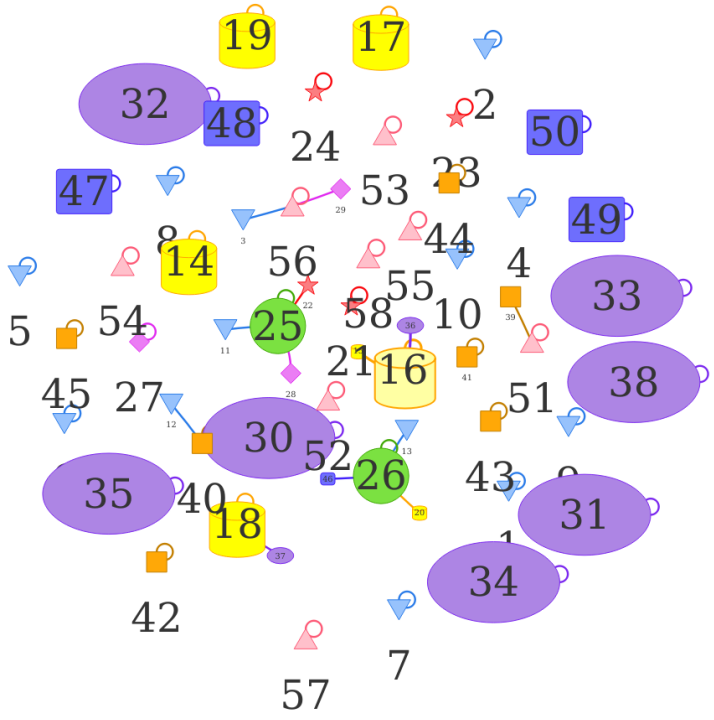


Figure 5.13 – Exemplar neighborhood with a k value of 48

k	**Number of exemplars**	**Sampling distribution**	**Number of samples representing an other class**
7 (12 %)	57	(6, 8, 5, 2, 4, 9, 7, 3, 13)	0
10 (17 %)			
14 (24 %)		(7, 7, 5, 2, 4, 9, 7, 3, 13)	
19 (33 %)	53	(6, 7, 5, 2, 4, 9, 7, 2, 11)	4
29 (50 %)	44	(6, 7, 4, 2, 3, 7, 5, 1, 9)	7
38 (66 %)	40	(6, 5, 4, 2, 3, 6, 5, 1, 8)	
48 (82 %)	34	(7, 4, 3, 0, 3, 5, 4, 2, 6)	

Table 5.5 – Different values of k with Intercor TestFest data

In this chapter, we have seen how we can use a Big Data analysis methodology with VANET data taken from the Scoop@f project, and the second InterCor TestFest event. That methodology use the local density of variables values to extract samples. These samples are used with our data as driver's behavior. We have seen that these behaviors are distinct from a driver to another, and raise issues on privacy, because even if the vehicle change its pseudonym along the journey, the driver does not change his behavior, and so he is still traceable.

Chapter 6

Conclusions and Perspectives

6.1 Conclusions

In Chapter 2, we have seen an overview of the VANET technologies, firstly in a large manner, and then, more specifically the ETSI standard.

VANET are the base of Cooperative Intelligent Transport System (C-ITS), using V2X communications: Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I). In Europe, we use the European Telecommunications Standards Institute (ETSI) standardization, which is based on the IEEE 802.11p (an IEEE 802.11a amendment).

After that, we have seen a Big Data overview, with techniques used in different steps and some applications of Big Data. Big Data was first defined by the 3V: Volume, Velocity, and Variety. Two other V was added after: Value and Veracity. Volume concerns the data scale; Velocity that data collection and analysis must be rapidly and timely conducted; Variety is the heterogeneity of data; Value is the low density of a huge value; and Veracity is the reliability of the values.

Big Data can be used in many domains: Enterprises; Internet of Things; Online Social Media; Collective Intelligent or Smart grid.

In Chapter 3, we have seen that VANET need the Big Data technologies: actually, VANET challenge answer to the Variety, Velocity, Veracity, and Volume questions. We have the Variety by the number of options and the number of different generated data (sensor, message, depend on type of vehicle); the Velocity intrinsic of the network topology, and the rapidity of data to change, notably with the security problematic and the privacy with the anonymization; the Veracity, which is necessary in a C-ITS environment, and improved by the security layer with the signature, which guarantee the authenticity and the integrity of the message; and finally, the Volume, because of the increasing of the number of VANET actors.

Because of these problematic, many ITS projects in Europe need to use Big Data technologies.

In Chapter 4, we have seen how we implemented the ETSI ITS stack, by using the Qt framework, enabling us to use a complete library simplifying the code, and especially, the signals and slots mechanisms, which permit us

to create fast and easier communications between the different layers. Our stack is as close as possible as the ETSI standard in a logical point of view: we have the HMI, which communicates with the Application layer using a UDP connection (through Bluetooth if needed), the Application communicates with the Facilities by using UDP, which allows us to have a distant Application layer, as the UpperTester, or if we need to separate the communication part from the application part. The Facilities layer can use directly the GeoNet function to send payload, by using a structure to indicate how it wants that the payload is sent. Finally, the GeoNetworking layer manages the Beacon and the Headers generation, and send it to the appropriate interfaces. We have a Management layer, which is accessible by all parts, which provides the state of the unit, and an other layer accessible by all the entities: the Security layer. That last signs and verifies messages, and can be assigned by everyone if needed.

We have also seen how we can validate systems, by using the TTCN-3 language and frameworks to build it and generate tests campaign. The TTCN-3 is the language used to implement the scenarios that we have extracted from standardization and deliverable. Some extracted properties have to be test first in laboratory, to be sure that they work before test them on real environment, because the road tests are more expensive. When we make laboratory tests, we had a piece of code, the UpperTester, on the unit that will emulate the environment. The critical problem of the UpperTester is that it is only to modify the values, not the behavior. For example, if we need to simulate the modification of the unit yaw rate, it has to indicate to the appropriate sensor the modification, and the sensor will check the constraints on the value, not the UpperTester.

For road tests, since we do not have the UpperTester, we can not emulate the environment (and it is not wanted). And so, to know if the unit has the good behavior, they need to log their state. To do that, in the Scoop@f project, we use two types of logs: TLogs for the technical tests, and ULogs for the supervision of the use the Scoop@f application. We have two possibilities to access these logs: in offline mode where we manually access the log files, and in online mode, where mobile units send the files to ITS-R, which forward them to the ITS-C. We have seen that some improvement can be done in the log format and how they are sent to the ITS-C. The actual definition of the logs does not permit to automatically use them, since there is no way to discover, which kind of log we have. Moreover, to send them to the ITS-R, an extension of the CAM (the CAM-I) is made, but since we can have only one service per CAM, we need to violate the frequency of ITS-R (of 1 message per second) to send the different services. So, we proposed a new

version of the CAM-I and of the log files to be more generic and extensible with others log formats.

In Chapter 5, we have seen an overview of the kNN methods. We have seen the basic kNN, which permits classifying new element by using a training set, and some improvements that can be used depending on the use case. We proposed a new methodology, which is based on the local density of each data to extract samples from database. That methodology was assessed with random data, well-known databases, and we have seen that we can extract exemplars that corresponds to the good classes, i.e. for most of them, the neighborhood of the sample is from the same class as the sample itself. Unfortunately, as for most of kNN methods, our methodology has a big dependency on choice of the k value, and has a complexity of $O(N^3)$. We have implemented the algorithm with different technologies: R / Shiny and C++. With the C++ technology, the algorithm runs quickly, but not with R. By parallelize the sample extraction algorithm with 8 threads, we divided by two the time execution. The parallelization is made on the firsts nested loop. That way is quicker than parallelize each step, because of the time to allocate the threads.

We have then used the methodology with real VANET data: firstly, with Scoop@f road tests, and secondly, with the InterCor TestFest. The Scoop@f road tests was the simulation of a slippery roadway, where we try with the TLogs to see if there is differences between different passages. Since we can extract some samples, it means that we have differences with the same driver on this road. In the TestFest, vehicles experimented the ITS-G5 secured communication on city, national roads and roadways. We selected 24 strategic points on the 30 km track. These points are where the vehicle need to decelerate, and where we have events. Each ITS-G5 manufacturer of the TestFest is considered as a class, and when we use the algorithm with these data, we see that class are not mixed: the sample class is the same as its neighborhood.

6.2 Perspectives

In the near future, we will use our new methodology, to extract some driver's behaviors, which is an important point for the security challenge. Actually, if we can extract behaviors, it questions the privacy, since if the pseudonym changes, the driver's behavior remain the same.

There are also many challenging issues raised from VANET. We selected to present here three of the most crucial challenges.

1. **Law Evolution**

Law needs to be adapted with VANET evolution to ensure anonymity and responsibility for VANET usage. Anonymity concerns essentially Europe, because in Europe there is a culture of privacy, which is strongly turned up. Each country has its own definition and ways to ensure privacy, and so they will need to have a consortium to decide these issues together.

2. **Scalability and interoperability**

   With a prediction of 35 % of the vehicle marketing in 2022 approximating around 113 billion euros of revenue ([5, 6]), interoperability will be another challenge, among all providers and with more countries. One of interoperability challenges will be the interoperability concerning security and privacy infrastructure: each country has its own infrastructure and so when a vehicle travels to a foreign country, it needs to adapt communications with the infrastructure. InterCor is an example of project that intends to do that with the interoperability of four European countries: Netherland, Belgium, France and United Kingdom [90].

3. **Agile Approach**

   Since VANET are becoming a Big Data challenge, with the same issues, we need to change the way we look them: we can not stay with a classical view, with static implementation in each layer. We need agile approach, because technology evolve quickly: we have more and more connected vehicles and the autonomous car is coming on the market. All these technologies need to be able to be adapted easily. We can not stay with classical programming, because we could need to make a gap in the different evolution of the equipment. When we use vehicles, this is not allowed, because of security constraints. The adaption should be done easily to be compliant with the security challenges.

Bibliography

[1] John Gantz and David Reinsel. "Extracting value from chaos". In: *IDC iview* 1142.2011 (2011), pp. 1–12.

[2] Yunshu Liu et al. "Traffic big data analysis supporting vehicular network access recommendation". In: *Communications (ICC), 2016 IEEE International Conference on*. IEEE. May 2016, pp. 1–6.

[3] Nan Cheng et al. "Big data driven vehicular networks". In: *IEEE Network* 99 (2018), pp. 1–8.

[4] IEEE. *IEEE Draft Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (November 2011)*. 2011.

[5] *Connected Car Market by Hardware (Semiconductor Components, and Connectivity ICs- Wi-Fi, Bluetooth and Cellular), Application (Telematics, Infotainment, and Combined Telematics & Infotainment), and Geography - Global Forecast to 2022Connected Car Market by Hardware (Semiconductor Components, and Connectivity ICs- Wi-Fi, Bluetooth and Cellular), Application (Telematics, Infotainment, and Combined Telematics & Infotainment), and Geography - Global Forecast to 2022*. Research and Market, 2017.

[6] Christian Radüge Richard Viereckl Jörg Assmann. "The bright future of connected cars". In: *Strategy&* (2014).

[7] Min Chen, Shiwen Mao, and Yunhao Liu. "Big data: a survey". In: *Mobile Networks and Applications* 19.2 (2014), pp. 171–209.

[8] M. Fiore et al. "Vehicular Mobility Simulation for VANETs". In: *Simulation Symposium, Annual(ANSS)*. Vol. 00. Mar. 2007, pp. 301–309. DOI: 10.1109/ANSS.2007.44. URL: doi.ieeecomputersociety.org/10.1109/ANSS.2007.44.

[9] *Intelligent Transport Systems (ITS); European profile standard for the physical and medium access control layer of Intelligent Transport Systems operating in the 5 GHz frequency band*. ETSI ES 202 663 (ETSI Standard). Nov. 2009.

[10] *ETSI EN 302 636-4-1; Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 4: Geographical Addressing and forwarding for point-to-point and point-to-multipoint communications; Sub-part 1: Media-Independent Functionality*. 302 636-4-1 (European Standard). July 2014.

[11] *ETSI EN 302 636-5-1; Intelligent Transport Systems (ITS); Vehicular Communications; GeoNetworking; Part 5: Transport Protocols; Sub-part 1: Basic Transport Protocol*. 302 636-5-1 (European Standard). Aug. 2014.

[12] *ETSI EN 302 637-2; Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service*. 302 637-2 (European Standard). Nov. 2014.

[13] *ETSI EN 302 637-3; Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Application; Part 3: Specifications of Decentralized Environmental Notification Basic Service*. 302 697-3 (European Standard). Nov. 2014.

[14] URL: https://ec.europa.eu/inea/en/connecting-europe-facility/cef-transport/projects-by-country/multi-country/2014-eu-ta-0669-s.

[15] I Present. "Cramming more components onto integrated circuits". In: *Readings in computer architecture* 56 (2000).

[16] Gordon E Moore et al. "Progress in digital integrated electronics". In: *Electron Devices Meeting*. Vol. 21. 1975, pp. 11–13.

[17] Chip Walter. "Kryder's law". In: *Scientific American* 293.2 (2005), pp. 32–33.

[18] URL: http://hadoop.apache.org.

[19] Doug Laney. "3D data management: Controlling data volume, velocity and variety". In: *META group research note* 6.70 (2001), p. 1.

[20] Yuri Demchenko et al. "Addressing big data issues in scientific data infrastructure". In: *Collaboration Technologies and Systems (CTS), 2013 International Conference on*. IEEE. 2013, pp. 48–55.

[21] Yuri Demchenko, Cees De Laat, and Peter Membrey. "Defining architecture components of the Big Data Ecosystem". In: *Collaboration Technologies and Systems (CTS), 2014 International Conference on*. IEEE. 2014, pp. 104–112.

[22] Susan Miele and Rebecca Shockley. "Analytics: The real-world use of big data". In: *Retrieved from IBM Institute for Business Value, Said Business School* (2013).

[23] James Manyika et al. "Big data: The next frontier for innovation, competition, and productivity". In: (2011).

[24] Kenneth Cukier. *Data, data everywhere: A special report on managing information*. Economist Newspaper, 2010.

[25] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[26] URL: https://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research.

[27] URL: https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users.

[28] O Liu et al. "Social Network Analysis Using Big Data". In: *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 2. 2016.

[29] Junghoo Cho and Hector Garcia-Molina. "Parallel crawlers". In: *Proceedings of the 11th international conference on World Wide Web*. ACM. 2002, pp. 124–135.

[30] Maurizio Lenzerini. "Data integration: A theoretical perspective". In: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM. 2002, pp. 233–246.

[31] Jonathan I Maletic and Andrian Marcus. "Data Cleansing: Beyond Integrity Analysis." In: *Iq*. Citeseer. 2000, pp. 200–209.

[32] Xiaomeng Yi et al. "Building a network highway for big data: architecture and challenges". In: *IEEE Network* 28.4 (2014), pp. 5–13.

[33] Y. Xu et al. "mPath: High-Bandwidth Data Transfers with Massively Multipath Source Routing". In: *IEEE Transactions on Parallel and Distributed Systems* 24.10 (Oct. 2013), pp. 2046–2059. ISSN: 1045-9219. DOI: 10.1109/TPDS.2012.298.

[34] Mike P Wittie et al. "Exploiting locality of interest in online social networks". In: *Proceedings of the 6th International COnference*. ACM. 2010, p. 25.

[35] Nikolaos Laoutaris et al. “Inter-datacenter bulk transfers with netstitcher”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 74–85.

[36] Yuan Feng, Baochun Li, and Bo Li. “Jetway: Minimizing costs on inter-datacenter video traffic”. In: *Proceedings of the 20th ACM international conference on Multimedia*. ACM. 2012, pp. 259–268.

[37] Mosharaf Chowdhury et al. “Managing data transfers in computer clusters with orchestra”. In: *ACM SIGCOMM Computer Communication Review*. Vol. 41. 4. ACM. 2011, pp. 98–109.

[38] Xia Zhou et al. “Mirror mirror on the ceiling: flexible wireless links for data centers”. In: *ACM SIGCOMM Computer Communication Review* 42.4 (2012), pp. 443–454.

[39] Martin Kleppmann. *Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems.* ” O’Reilly Media, Inc.”, 2017.

[40] URL: `https://aws.amazon.com/dynamodb`.

[41] URL: `http://www.project-voldemort.com/voldemort`.

[42] URL: `https://cloud.google.com/bigtable`.

[43] URL: `http://cassandra.apache.org`.

[44] URL: `https://www.mongodb.com`.

[45] URL: `http://couchdb.apache.org`.

[46] Arun K Pujari. *Data mining techniques*. Universities press, 2001.

[47] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[48] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), p. 436.

[49] Evangelos Triantaphyllou. “Multi-criteria decision making methods”. In: *Multi-criteria decision making methods: A comparative study*. Springer, 2000, pp. 5–21.

[50] Ian H Witten et al. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[51] Manas Ranjan Mallick. “A Comparative Study Of Wireless Protocols With Li-Fi Technology: A Survey”. In: *Proceedings of 43rd IRF International Conference, 29th May*. 2016, pp. 8–12.

[52] Xingqin Lin, Ansuman Adhikary, and Y-P Eric Wang. “Random Access Preamble Design and Detection for 3GPP Narrowband IoT Systems.” In: *IEEE Wireless Commun. Letters* 5.6 (2016), pp. 640–643.

[53] Sabrina Sicari et al. "Security, privacy and trust in Internet of Things: The road ahead". In: *Computer networks* 76 (2015), pp. 146–164.

[54] John Scott. *Social network analysis*. Sage, 2017.

[55] Alvaro Ortigosa, José M Martín, and Rosa M Carro. "Sentiment analysis in Facebook and its application to e-learning". In: *Computers in human behavior* 31 (2014), pp. 527–541.

[56] Olivier Serrat. "Social network analysis". In: *Knowledge solutions*. Springer, 2017, pp. 39–43.

[57] *Intelligent Transport Systems (ITS); Security; Security header and certificate formats*. ETSI TS 103 097 (Technical Specification). Oct. 2017.

[58] Rolf Sint et al. "Combining unstructured, fully structured and semi-structured information in semantic wikis". In: *Fourth Workshop on Semantic Wikis–The Semantic Wiki Web 6 th European Semantic Web Conference Hersonissos, Crete, Greece, June 2009*. Citeseer. 2009, p. 73.

[59] *ETSI EN 302 895; Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM)*. ETSI EN 302 895 (European Standard (Telecommunications series)). Sept. 2014.

[60] URL: https://transformingtransport.eu.

[61] URL: http://cordis.europa.eu/project/rcn/211234_fr.html.

[62] *AutoMat project*. URL: http://www.automat-project.eu.

[63] URL: big-data-europe.eu.

[64] *Thessaloniki traffic estimation project*. URL: http://trafficthess.imet.gr.

[65] URL: https://www.qt.io.

[66] URL: http://lionet.info/asn1c/documentation.html.

[67] URL: http://www.nongnu.org/confuse/manual.

[68] URL: http://www.tcpdump.org.

[69] *Intelligent Transport Systems (ITS); Architecture of conformance validation framework*. ETSI TR 103 099 (Technical Report). Mar. 2017.

[70] URL: http://www.ttcn-3.org.

[71] URL: https://www.spirent.com/Products/TTworkbench.

[72] URL: https://projects.eclipse.org/projects/tools.titan.

[73] URL: http://www.elvior.com.

[74] URL: https://www.verifysoft.com/en_elvior_testcast.html.

[75] *Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Facilities layer protocols and communication requirements for infrastructure services*. ETSI TS 103 301 (Technical Specification). Nov. 2016.

[76] Nitin Bhatia et al. "Survey of nearest neighbor techniques". In: *arXiv preprint arXiv:1007.0085* (2010).

[77] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[78] Sahibsingh A Dudani. "The distance-weighted k-nearest-neighbor rule". In: *IEEE Transactions on Systems, Man, and Cybernetics* 4 (1976), pp. 325–327.

[79] Hamid Parvin, Hosein Alizadeh, and Behrouz Minaei-Bidgoli. "MKNN: Modified k-nearest neighbor". In: *Proceedings of the World Congress on Engineering and Computer Science*. Vol. 1. Citeseer. 2008.

[80] Klaus Hechenbichler and Klaus Schliep. "Weighted k-nearest-neighbor techniques and ordinal classification". In: (2004).

[81] Gongde Guo et al. "KNN model-based approach in classification". In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2003, pp. 986–996.

[82] Bamshad Mobasher, Robin Burke, and Jeff J Sandvig. "Model-based collaborative filtering as a defense against profile injection attacks". In: *AAAI*. Vol. 6. 2006, p. 1388.

[83] Wenming Zheng, Li Zhao, and Cairong Zou. "Locally nearest neighbor classifiers for pattern classification". In: *Pattern recognition* 37.6 (2004), pp. 1307–1309.

[84] Marcello Frixione and Antonio Lieto. "Prototypes Vs Exemplars in Concept Representation." In: *KEOD*. 2012, pp. 226–232.

[85] Brendan J Frey and Delbert Dueck. "Clustering by passing messages between data points". In: *Science* 315.5814 (2007), pp. 972–976.

[86] William G Cochran. *Sampling techniques*. John Wiley & Sons, 2007.

[87] Michael E Houle et al. "Can shared-neighbor distances defeat the curse of dimensionality?" In: *International Conference on Scientific and Statistical Database Management*. Springer. 2010, pp. 482–500.

[88] Julian Heinrich and Daniel Weiskopf. "State of the Art of Parallel Coordinates." In: *Eurographics (STARs)*. 2013, pp. 95–116.

[89] Kevin Bache and Moshe Lichman. *UCI machine learning repository*. 2013.

[90] *Intercor project*. URL: `http://intercor-project.eu`.

[91] *Intelligent Transport Systems (ITS); Vehicular Communications; Geographical Area Definition*. ETSI EN 302 931 (European Norm). July 2011.

Appendix A

GeoNetworking headers

In this appendix, all the figures and tables are extracted from the ETSI standard [10].

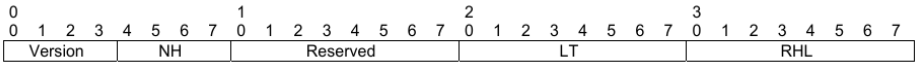


Figure A.1 – *Basic Header* structure

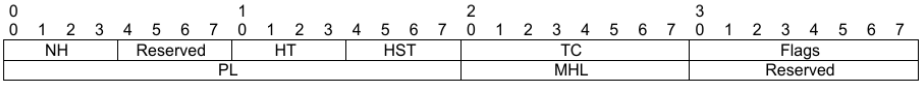


Figure A.2 – *Common Header* structure

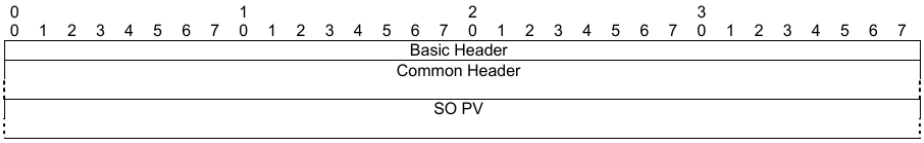


Figure A.3 – *Beacon* structure

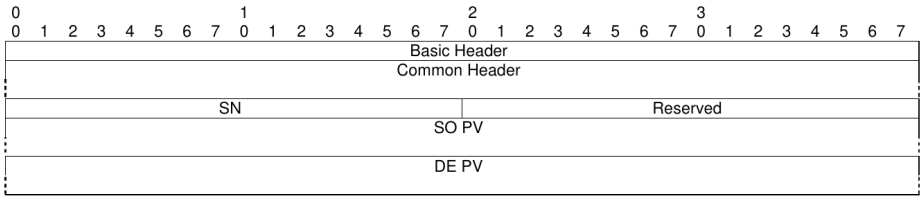


Figure A.4 – *GeoUnicast* structure

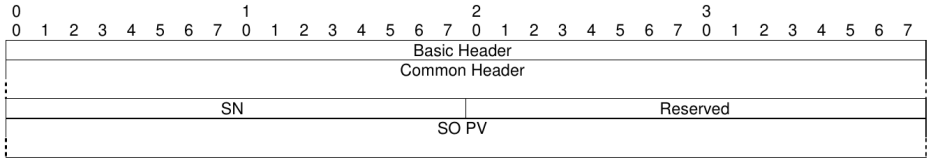


Figure A.5 – *Topologically-Scoped Broadcast* structure

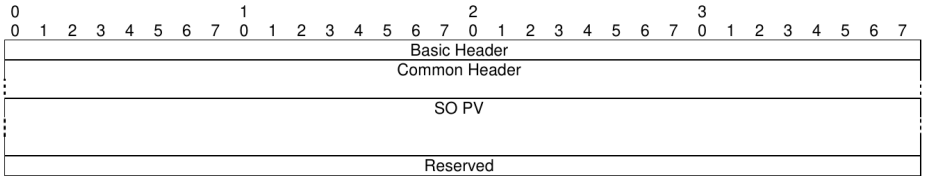


Figure A.6 – *Single-Hop Broadcast* structure

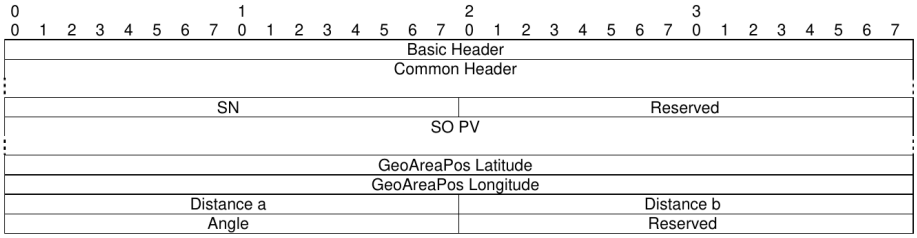


Figure A.7 – *GeoBroadcast* / *GeoAnycast* structure

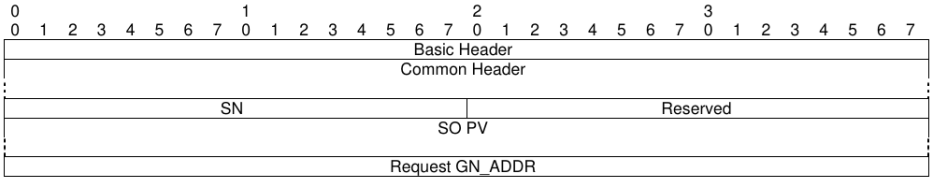


Figure A.8 – *Location Service Request* structure

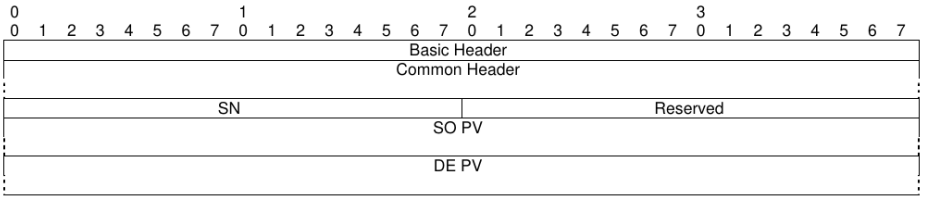


Figure A.9 – *Location Service Reply* structure

Header Type (HT)	**Header Sub-type (HST)**	**Encoding**	**Description**
Any		0	Unspecified
	Unspecified	0	Unspecified
Beacon		1	Beacon
	Unspecified	0	Unspecified
GeoUnicast		2	GeoUnicast
	Unspecified	0	Unspecified
GeoAnycast		3	GeoAnycast
	GEOANYCAST_ CIRCLE	0	Circular area
	GEOANYCAST_ RECT	1	Rectangular area
	GEOANYCAST_ ELIP	2	Ellipsoidal area
GeoBroadcast		4	GeoBroadcast
	GEOBROADCAST_ CIRCLE	0	Circular area
	GEOBROADCAST_ RECT	1	Rectangular area
	GEOBROADCAST_ ELIP	2	Ellipsoidal area
TSB		5	Topologically-scoped broadcast (TSB)
	SINGLE_HOP	0	Single-hop broadcast (SHB)
	MULTI_HOP	1	Multi-hop TSB
LS		6	Location service (LS)
	LS_REQUEST	0	Location service request
	LS_REPLY	1	Location service reply

Table A.1 – Header Type and Sub-type

Appendix B

GeoUnicast Forwarding Algorithms

B.1 Greedy Forwarding

Algorithm 1 Greedy Forwarding pseudo-code

```
-- P is the packet to be forwarded
-- i is the i-th neighbor from ITS location table entry (LocTE)
-- NH is the LocTE identified as next hop, NH.LL_ADDR its link layer address
-- NH_LL_ADDR is the link layer address of the next hop
-- LPV is the local position vector
-- PV_P is the destination position vector in the GeoNetworking packet to be forwarded
-- PV_I is the position vector of the i-th LocTE
-- MFR indicates the progress according to the MFR policy
-- B is the forwarding packet buffer
-- LocT is the location table
-- TC is traffic class request by the application (source operation) or the field in the received Common Header
-- BCAST is the Broadcast LL address

MFR ← DIST(PV_P, LPV)  -- Initialize MFR
for i ∈ LocT do
   if i.IS_NEIGHBOUR then  -- LocTE i is neighbor
      if DIST(PV_P, PV_I) < MFR then
         NH ← i
         MFR ← DIST(PV_P, PV_I)
      end if
   end if
end for
if MFR < DIST(PV_P, LPV) then
   NH_LL_ADDR ← NH.LL_ADDR
else -- Forwarder is at a local optimum
   if LocT.HAS_NO_NEIGHBOURS & TC.SCF_IS_ENABLED then
      ADD P TO B
      NH_LL_ADDR ← 0  -- Indicates that packet is buffered
   else
      NH_LL_ADDR ← BCAST  -- No buffering allowed, fall back to BCAST
   end if
end if
```

B.2 Contention-Based GUC Forwarding Algorithm

$$TO\_CBF\_GUC = \begin{cases} TO\_CBF\_MAX + \dfrac{TO\_CBF\_MIN - TO\_CBF\_MAX}{DIST\_MAX} \times PROG & \text{for } PROG \leqslant DIST\_MAX \\ TO\_CBF\_MIN & \text{for } PROG > DIST\_MAX \end{cases} \quad \text{(B.1)}$$

where:

- TO_CBF_MIN is the minimum duration the packet shall be buffered.
- TO_CBF_MAX is the maximum duration the packet shall be buffered.
- PROG is the forwarding progress of the local GeoAdhoc router towards the destination, i.e. the difference between the sender's distance and GeoAdhoc router's local distance from the destination. The sender position is taken from its LocTE.

Algorithm 2 Contention-Based Forwarding for GUC pseudo-code

```
-- P is the GUC packet to be forwarded
-- LPV is the local position vector
-- PV_P is the destination position vector contained in the GeoNetworking packet
-- PV_SE is the position vector in the LocT with position accuracy indicator PAI_SE
-- B is the forwarding packet buffer
-- TO is the timeout that triggers the rebroadcast of the packet
-- NH_LL_ADDR is the link layer address of the next hop
-- BCAST is the Broadcast LL address

if P ∈ B then  -- Contending
   REMOVE P FROM B
   STOP TIMER
   DISCARD P
   RETURN -1  -- Indicates that packet is discarded
else -- New packet
   if (PV_SE EXISTS) AND (PAI_SE = TRUE) then PROG ← (DIST(PV_P, PV_SE) - DIST(PV_P, LPV)
      if PROG > 0 then  -- Forwarding process
         ADD P TO B
         TO ← Equation (B.1)
         START TIMER(TO)
         RETURN 0  -- Indicates that packet is buffered
      else
         DISCARD P
         RETURN -1  -- Indicates that packet is discarded
      end if
   else
      ADD P TO B
      TO ← TO_CBF_MAX
      RETURN 0  -- Indicates that packet is buffered
   end if
end if

if TIMER(TO) EXPIRES then
   FETCH P FROM B
   NH_LL_ADDR ← BCAST
   RETURN NH_LL_ADDR  -- Indicates that packet could be forwarded
end if
```

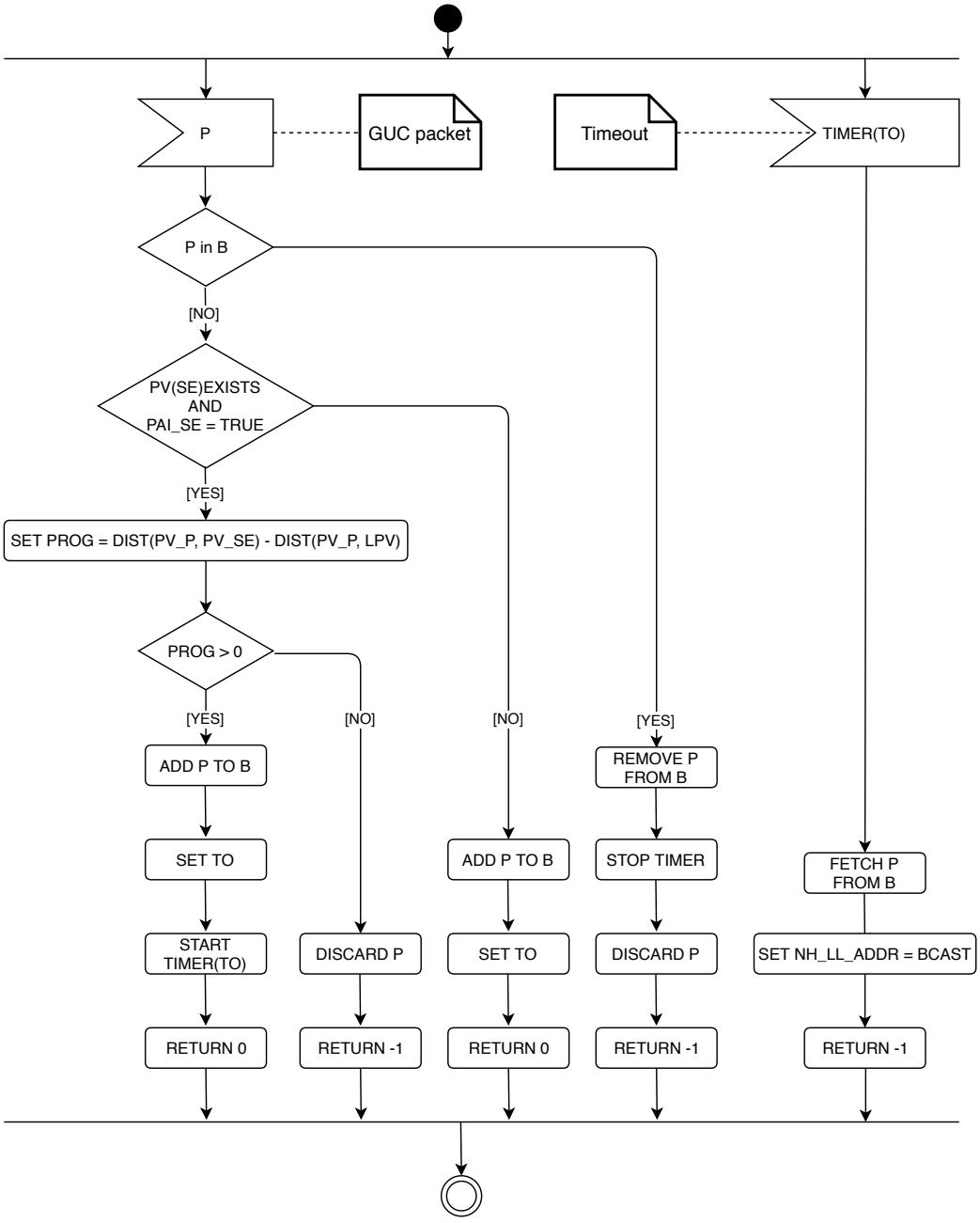


Figure B.1 – Activity diagram for GUC CBF

Appendix C

GeoBroadcast Forwarding Algorithms

C.1 GeoBroadcast functions

The F function returns an integer as follows:

$$F(x,y) = \begin{cases} = 1 \text{ for } x = 0 \text{ and } y = 0 \text{ (at the center point)} \\ > 0 \text{ inside the geographical area} \\ = 0 \text{ at the border of the geographical area} \\ < 0 \text{ outside the geographical area} \end{cases} \tag{C.1}$$

and is calculated as follows [91]:

- for circular area
$$F(x,y) = 1 - \left(\frac{x}{r}\right)^2 - \left(\frac{y}{r}\right)^2 \tag{C.2}$$
- for rectangular area
$$F(x,y) = \min\left(1 - \left(\frac{x}{a}\right)^2, 1 - \left(\frac{x}{a}\right)^2\right) \tag{C.3}$$
- for ellipsoidal area
$$F(x,y) = 1 - \left(\frac{x}{a}\right)^2 - \left(\frac{y}{b}\right)^2 \tag{C.4}$$

where

- x is the latitude of the center point
- y is the longitude of the center point
- r is the radius of the circle
- a is the distance between the center point and the short side of the rectangle or the length of the long semi-axis for ellipsoidal area

- b is the distance between the center point and the long side of the rectangle or the length of the short semi-axis for ellipsoidal area

The G function returns 1 or -1 as follows:

$$G = \begin{cases} +1 & \text{inside or at the border of the sectorial area} \\ -1 & \text{outside the sectorial area} \end{cases} \tag{C.5}$$

and is defined as follows:

$$G = \begin{cases} & (DIST\_R < DIST\_F) \text{ AND} \\ +1 \quad \text{for} & (DIST\_F < DIST\_MAX) \text{ AND} \\ & (\angle FSR \leqslant ANGLE\_TH) \\ -1 \quad \text{otherwise} \end{cases} \tag{C.6}$$

where:

- DIST_R is the distance between the GeoAdhoc router's local position and the sender position
- DIST_F is the distance between the forwarder position and the sender position
- DIST_MAX is the theoretical maximum communication range
- $\angle$FSR is the angle between the positions of the forwarder, the sender and the local GeoAdhoc router
- ANGLE_TH is a threshold value for the angle, which should be in the range $[30, 60]^\circ$ in function of the neighborhood density. For example:
  1. 30° when density is less than 0,025 node/m^2
  2. 45° when density is more than 0,025 node/m^2 and less than 0,05 node/m^2
  3. 60° when density is more than 0,05 node/m^2

C.2 Simple Forwarding Algorithm for GBC

Algorithm 3 Simple Forwarding for GBC pseudo-code

```
-- P is the GBC packet to be forwarded
-- LAT and LONG are latitude of the LPV, respectively
-- PV_SE is the sender position vector in its LocTE with latitude LAT_SE, longitude LONG_SE
-- with LAT_SE and LONG_SE as latitude and longitude
-- and position accuracy indicator PAI_SE
-- A is the center point of the destination area in the GeoNetworking
-- packet to be forwarded
-- NH_LL_ADDR is the link layer address of the next hop
-- BCAST is the Broadcast LL address
-- GREEDY() is the Algorithm 1
-- B is the forwarding packet buffer

Calculate F(LAT, LONG)  -- Equation (C.1)
if F ⩾ 0 then  -- Local GeoAdhoc router is inside or at the border of target area
   NH_LL_ADDR ← BCAST
   NH_LL_ADDR
else -- Local GeoAdhoc router is outside of target area
   if (PV_SE EXISTS) AND (PAI_SE = TRUE) then
      Calculate F(LAT_SE, LONG_SE)  -- Equation (C.1)
      if F < 0 then  -- Sender is outside of target area
         NH_LL_ADDR ← GREEDY(A)
         RETURN NH_LL_ADDR
      else
         DISCARD P  -- The packet as already reached the destination area
         RETURN -1  -- Indicates that packet is discarded
      end if
   else
      NH_LL_ADDR ← BCAST
      RETURN NH_LL_ADDR
   end if
end if
```

C.3 Contention-Based GBC Forwarding Algorithm

$$TO\_CBF\_GBC = \begin{cases} TO\_CBF\_MAX + \dfrac{TO\_CBF\_MIN - TO\_CBF\_MAX}{DIST\_MAX} \times DIST & \text{for } DIST \leqslant DIST\_MAX \\ TO\_CBF\_MIN & \text{for } DIST > DIST\_MAX \end{cases} \quad \text{(C.7)}$$

where:

- TO_CBF_MIN is the minimum duration the packet shall be buffered.
- TO_CBF_MAX is the maximum duration the packet shall be buffered.
- DIST is the distance between the GeoAdhoc router's local position and the sender (i.e. previous forwarder or source) position.
- DIST_MAX is the theorical maximum communication range of the wireless access technology

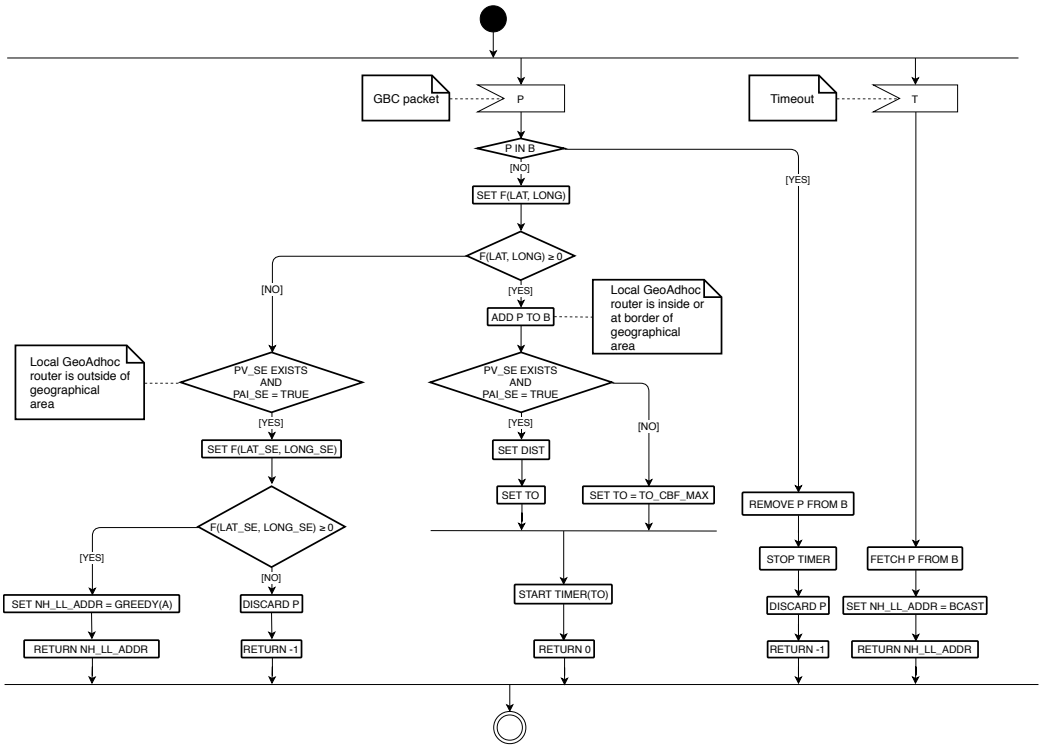


Figure C.1 – Activity diagram for GBC CBF

Algorithm 4 Contention-Based Forwarding for GBC pseudo-code

```
-- P is the GBC packet to be forwarded
-- LAT and LONG are latitude of the LPV, respectively
-- PV_SE is the sender position vector in its LocTE with latitude LAT_SE, longitude LONG_SE
-- with LAT_SE and LONG_SE as latitude and longitude
-- and position accuracy indicator PAI_SE
-- A is the center point of the destination area in the GeoNetworking
-- packet to be forwarded
-- NH_LL_ADDR is the link layer address of the next hop
-- BCAST is the Broadcast LL address
-- GREEDY() is the Algorithm 1
-- B is the forwarding packet buffer

if P IN B then  -- Contending
   REMOVE P FROM B
   STOP TIMER
   DISCARD P
   RETURN -1  -- Indicates that packet is discarded
else -- New packet
   Calculate F(LAT, LONG)  -- Equation (C.1)
   if F ⩾ 0 then  -- Local GeoAdhoc router is inside or at the border of target area
      ADD P TO B
      if (PV_SE EXISTS) AND (PAI_SE = TRUE) then
         DIST ← DIST(PV_SE, LPV)
         TO ← TO_CBF_GBC  -- Equation (C.7)
      else
         TO ← TO_CBF_MAX
      end if
      START TIMER(TO)
      RETURN 0  -- Indicates that packet is buffered
   else -- Local GeoAdhoc router is outside of target area
      if (PV_SE EXISTS) AND (PAI_SE = TRUE) then
         Calculate F(LAT_SE, LONG_SE)  -- Equation (C.1)
         if F < 0 then  -- Sender is outside of target area
            NH_LL_ADDR ← GREEDY(A)
            RETURN NH_LL_ADDR
         else
            DISCARD P  -- The packet as already reached the destination area
            RETURN -1  -- Indicates that packet is discarded
         end if
      else
         NH_LL_ADDR ← BCAST
         RETURN NH_LL_ADDR
      end if
   end if
end if

if TIMER(TO) EXPIRES then
   FETCH P FROM B
   NH_LL_ADDR ← BCAST
   RETURN P, NH_LL_ADDR
end if
```

C.4 Advanced GBC Forwarding Algorithm

Algorithm 5 Advanced Forwarding for GBC pseudo-code

```
-- P is the GBC packet to be forwarded
-- L_LL_ADDR is the LL address of the local GeoAdhoc router
-- NH_LL_ADDR is the link layer address of the next hop
-- DEST_LL_ADDR is the LL destination address carried in P
-- B is the forwarding packet buffer
-- LPV is the local position vector with latitude LAT and longitude LONG
-- PV_SE is the sender position vector in its LocTE with latitude LAT_SE, longitude LONG_SE
-- with LAT_SE and LONG_SE as latitude and longitude
-- and position accuracy indicator PAI_SE
-- TO is the timeout that triggers the rebroadcast of the packet
-- COUNTER is the retransmit counter for the packet P
-- MAX_COUNTER is the retransmit threshold
-- BCAST is the Broadcast LL address
-- GREEDY() is the Algorithm 1
-- INOUT1 indicates whether the local GeoAdhoc router is outside the target area or not
-- INOUT2 indicates whether the local GeoAdhoc router is outside the sectorial contention area or not
-- INOUT3 indicates whether the sender is outside the target area or not
-- A is the center point of the destination area in the GeoNetworking
-- packet to be forwarded

NH_LL_ADDR ← -1  -- Initialize NH_LL_ADDR
INOUT1 ← F(LAT, LONG)  -- Equation (C.1)
if INOUT1 ⩾ 0 then  -- Inside or at the border of target area
   if P IN B then  -- Contending
      if B.P.COUNTER ⩾ MAX_COUNTER then  -- Stop contending
         REMOVE P FROM B
         STOP TIMER
         DISCARD P
         RETURN -1  -- Indicates that packet is discarded
      else
         INOUT2 ← G()  -- Equation (C.6) for sectorial contention area
         if INOUT2 ⩾ 0 then  -- Inside or at the border of sectorial area
            REMOVE P FROM B
            STOP TIMER
            DISCARD P
            RETURN -1  -- Indicates that packet is discarded
         else -- Outside the sectorial area
            P.COUNTER++
            TO ← TO_CBF_GBC  -- Equation (C.7)
            START TIMER(TO)
         end if
      end if
   else -- New packet
      ADD P TO B
      if DEST_LL_ADDR = L_LL_ADDR then  -- Greedy forwarding
         COUNTER ← 1  -- Initialize COUNTER
         NH_LL_ADDR ← GREEDY(A)  -- Greedy() returns LL address of next hop or 0
         TO ← TO_CBF_MAX  -- Equation (C.7)
         START TIMER(TO)
         RETURN NH_LL_ADDR
      else -- CBF
         if (PV_SE EXISTS) AND (PAI_SE = TRUE) then
            DIST ← DIST(PV_SE, LPV)
            TO ← TO_CBF_GBC  -- Equation (C.7)
         else
            TO ← TO_CBF_MAX
         end if
         START TIMER(TO)
         RETURN 0  -- Indicates that packet is buffered
      end if
   end if
else -- Local GeoAdhoc router is outside of target area
   if (PV_SE EXISTS) AND (PAI_SE = TRUE) then
      Calculate F(LAT_SE, LONG_SE)  -- Equation (C.1)
      if F < 0 then  -- Sender is outside of target area
         NH_LL_ADDR ← GREEDY(A)
      else
         DISCARD P  -- The packet as already reached the destination area
         RETURN -1  -- Indicates that packet is discarded
      end if
```

```
    else
      NH_LL_ADDR ← BCAST
      RETURN NH_LL_ADDR
    end if
  end if

  if TIMER(TO) EXPIRES then
    FETCH P FROM B
    NH_LL_ADDR ← BCAST
    RETURN P, NH_LL_ADDR
  end if
```

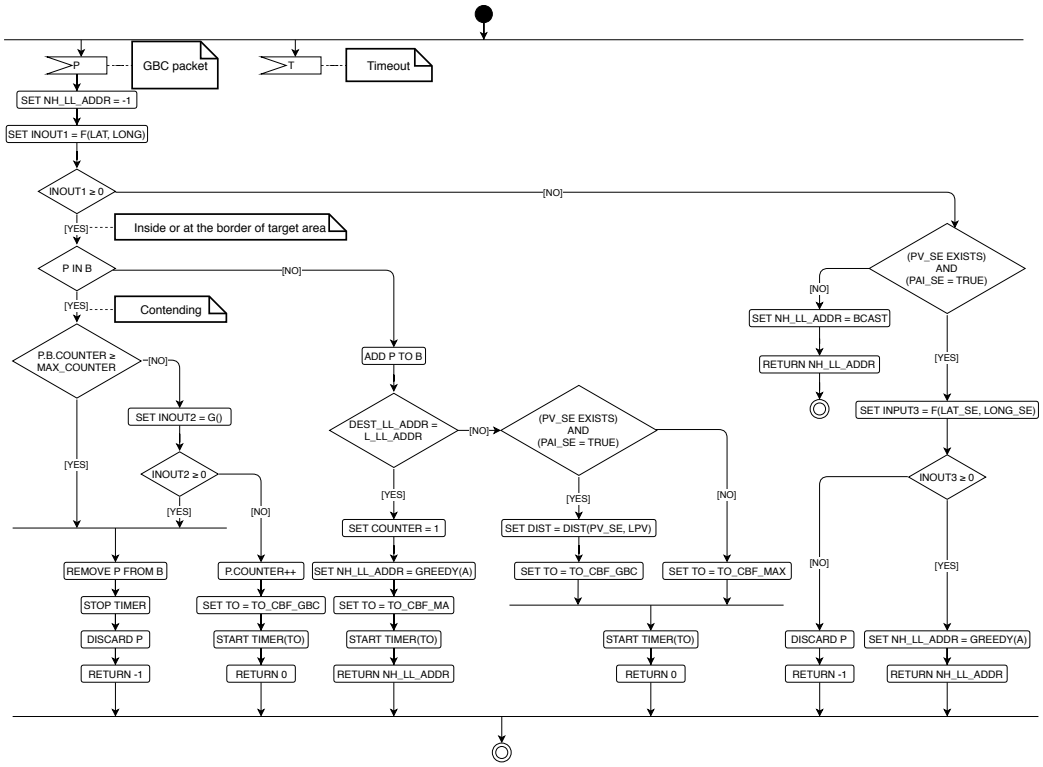


Figure C.2 – Activity diagram for advanced GBC forwarding

Appendix D

Protocol operation of the CA basic service

D.1 CAM sending operation

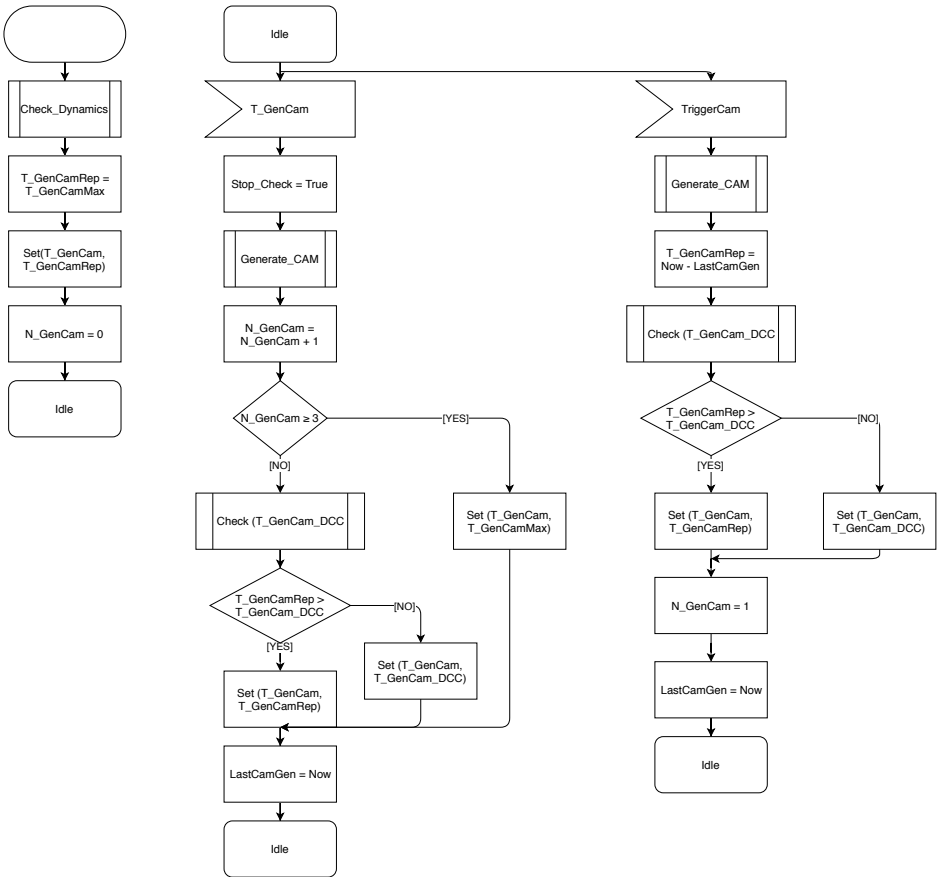


Figure D.1 – Process *Generate_CAM*

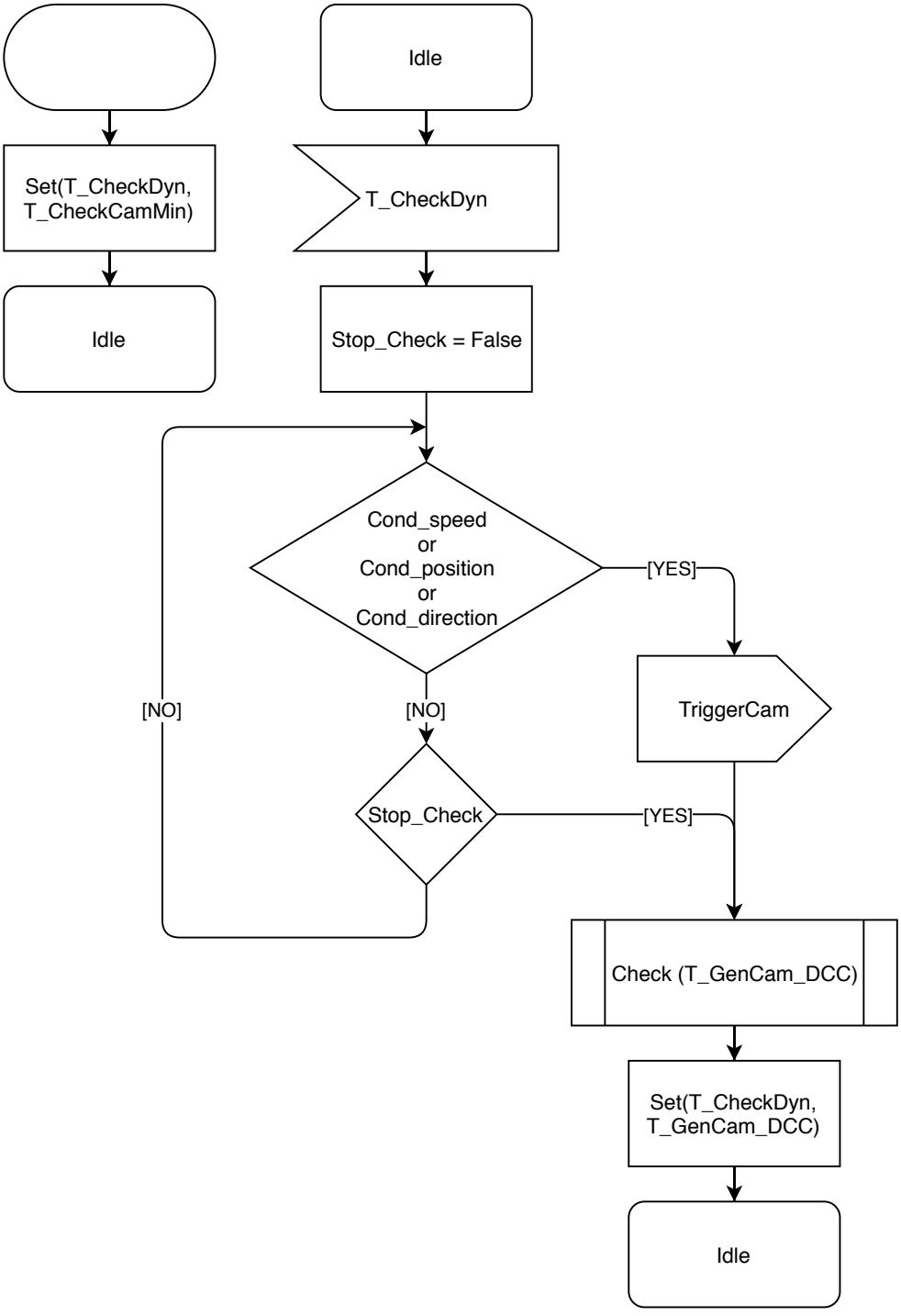


Figure D.2 – Process *Check_Dynamics*

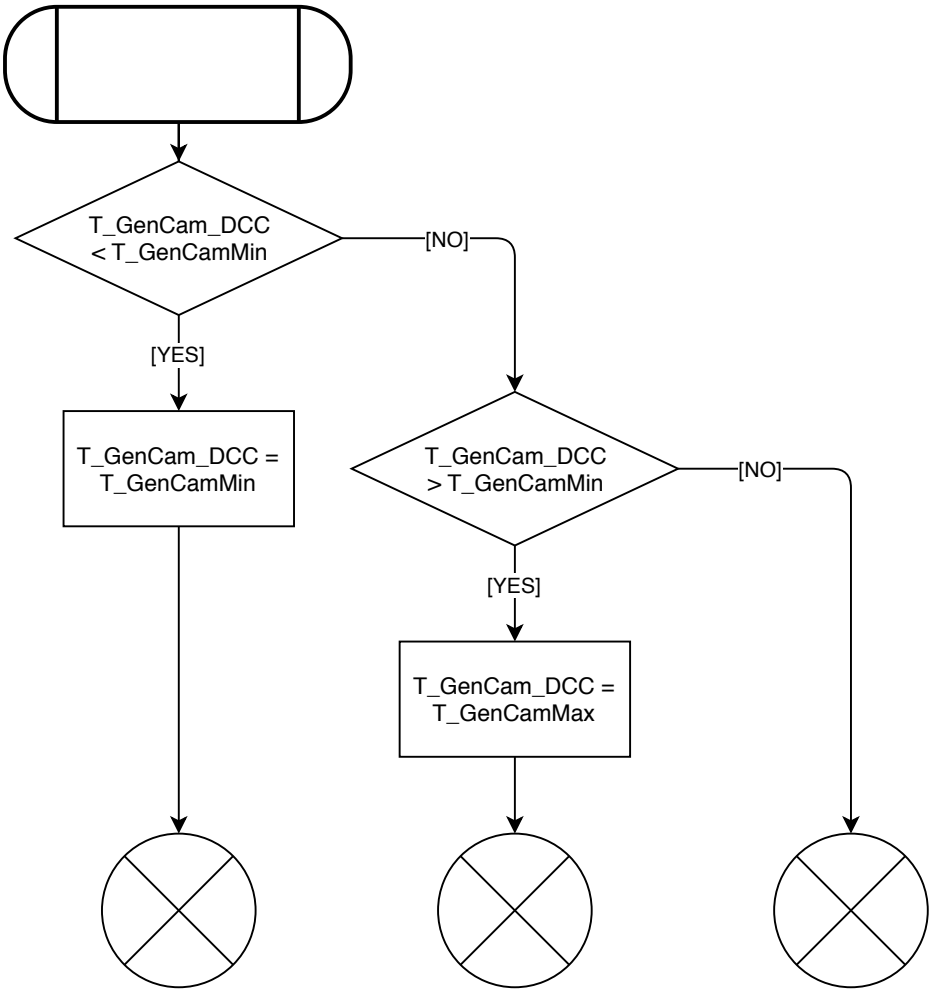


Figure D.3 – Procedure *Check*

Appendix E

Protocol operation of the DEN basic service

E.1 DENM sending operation

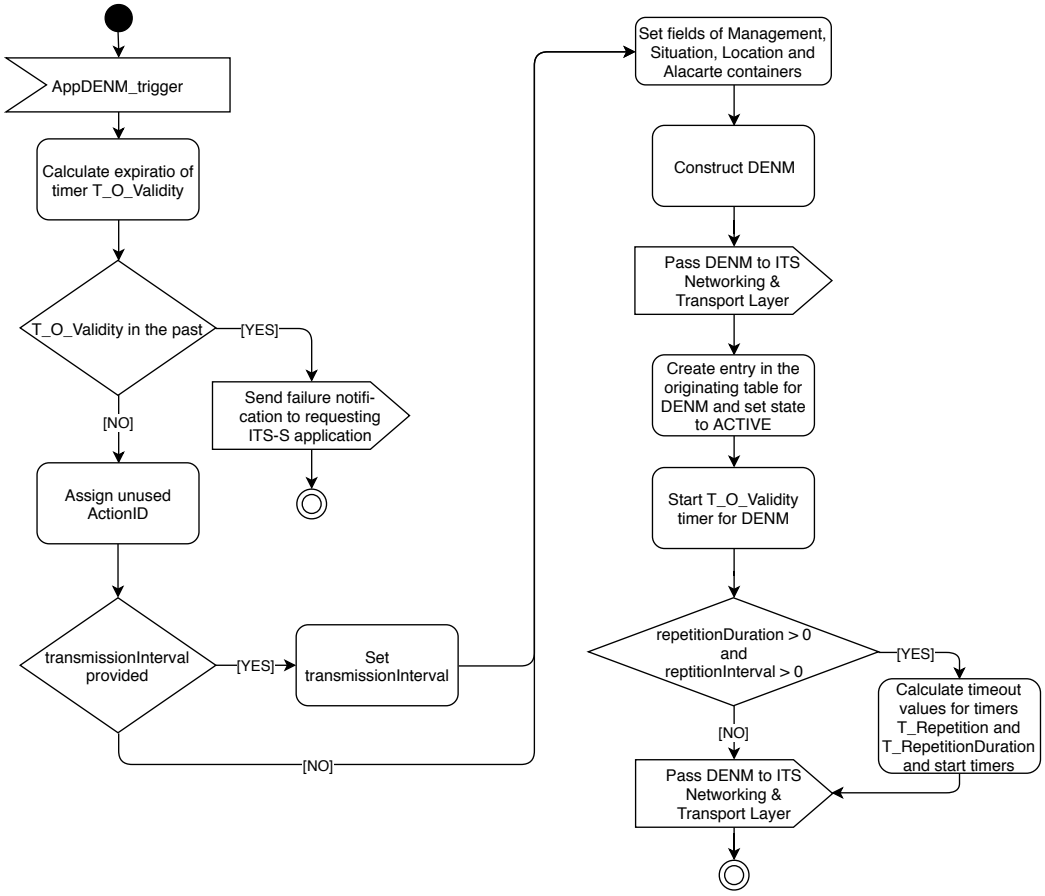


Figure E.1 – *AppDENM_trigger*

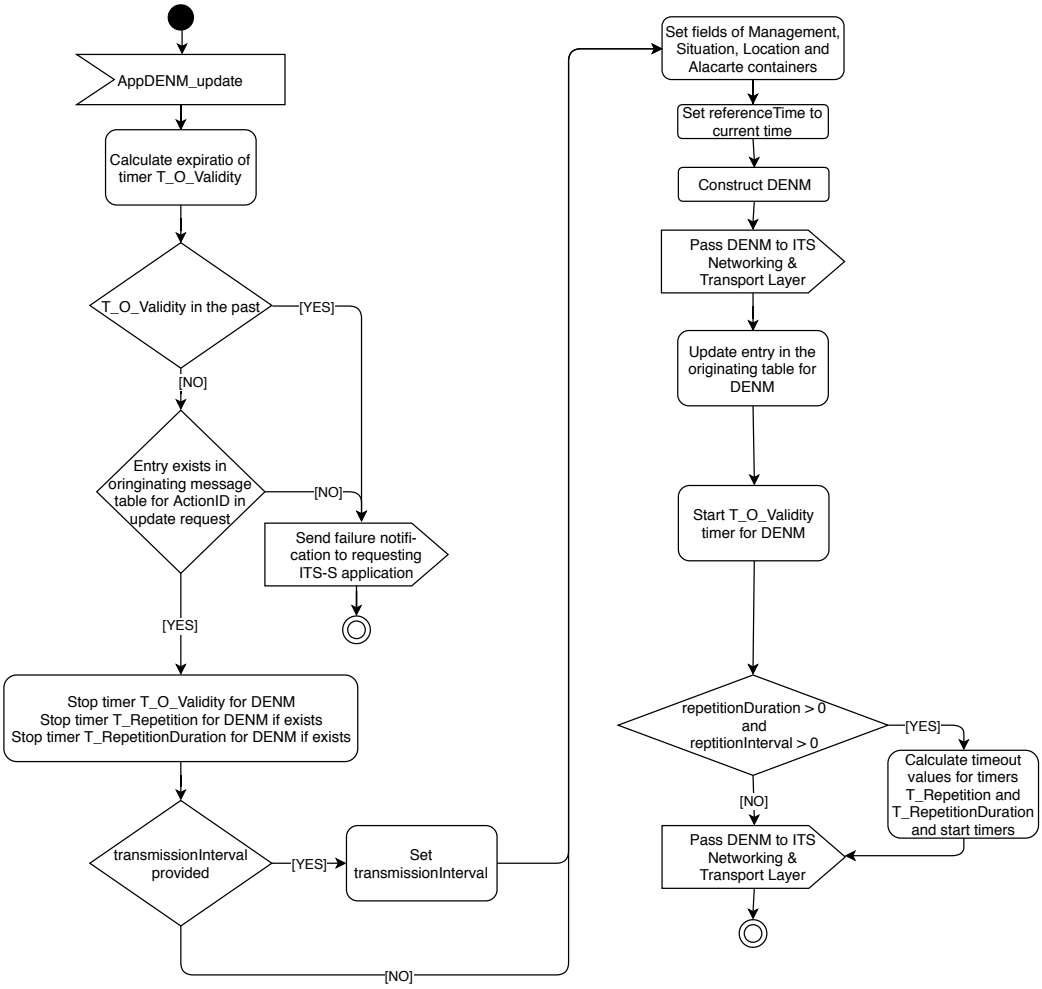


Figure E.2 – *AppDENM_update*

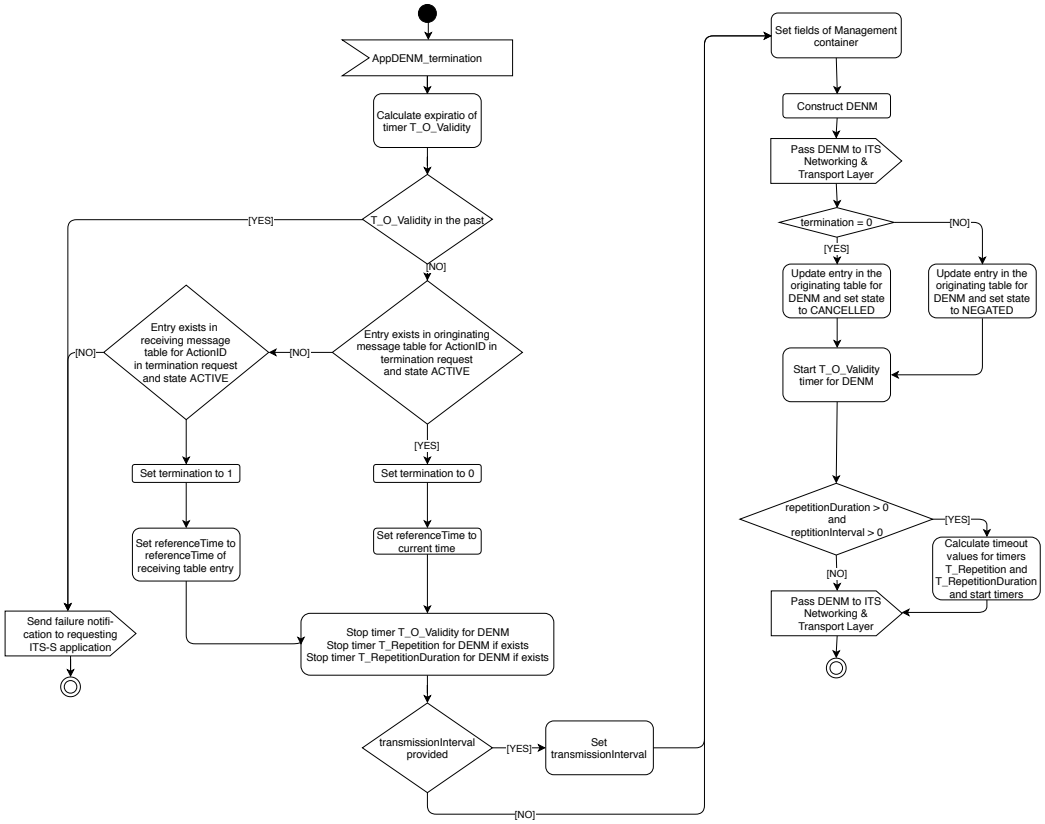


Figure E.3 – *AppDENM_termination*

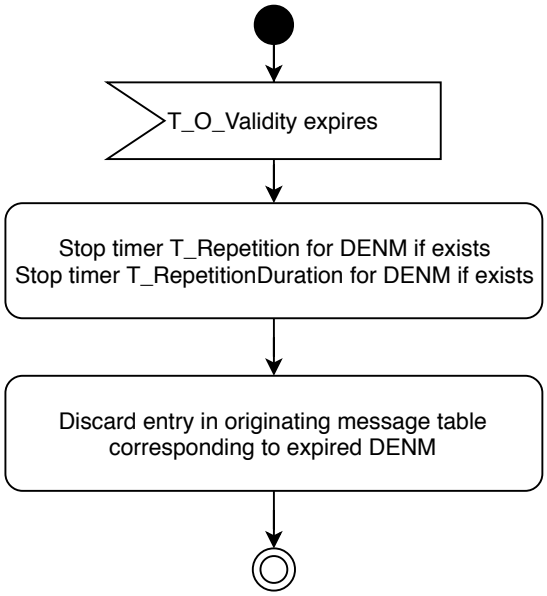


Figure E.4 – *T_O_Validity* expiration

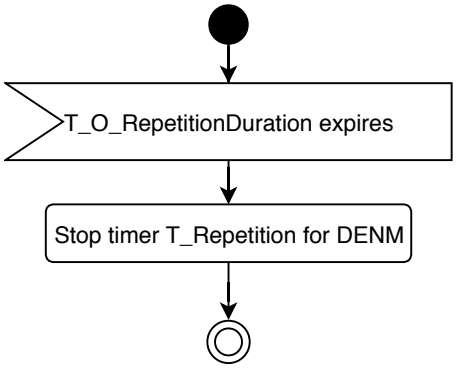


Figure E.5 – *T_RepetitionDuration* expiration

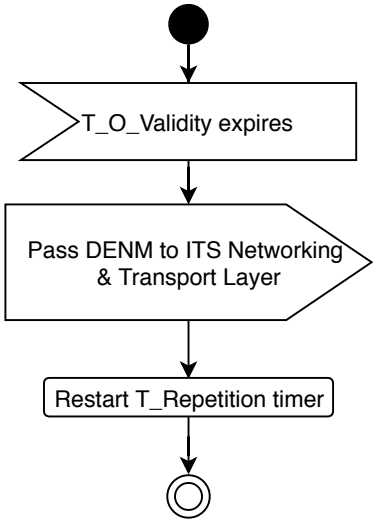


Figure E.6 – *T_Repetition* expiration

E.2 DENM forwarding operation

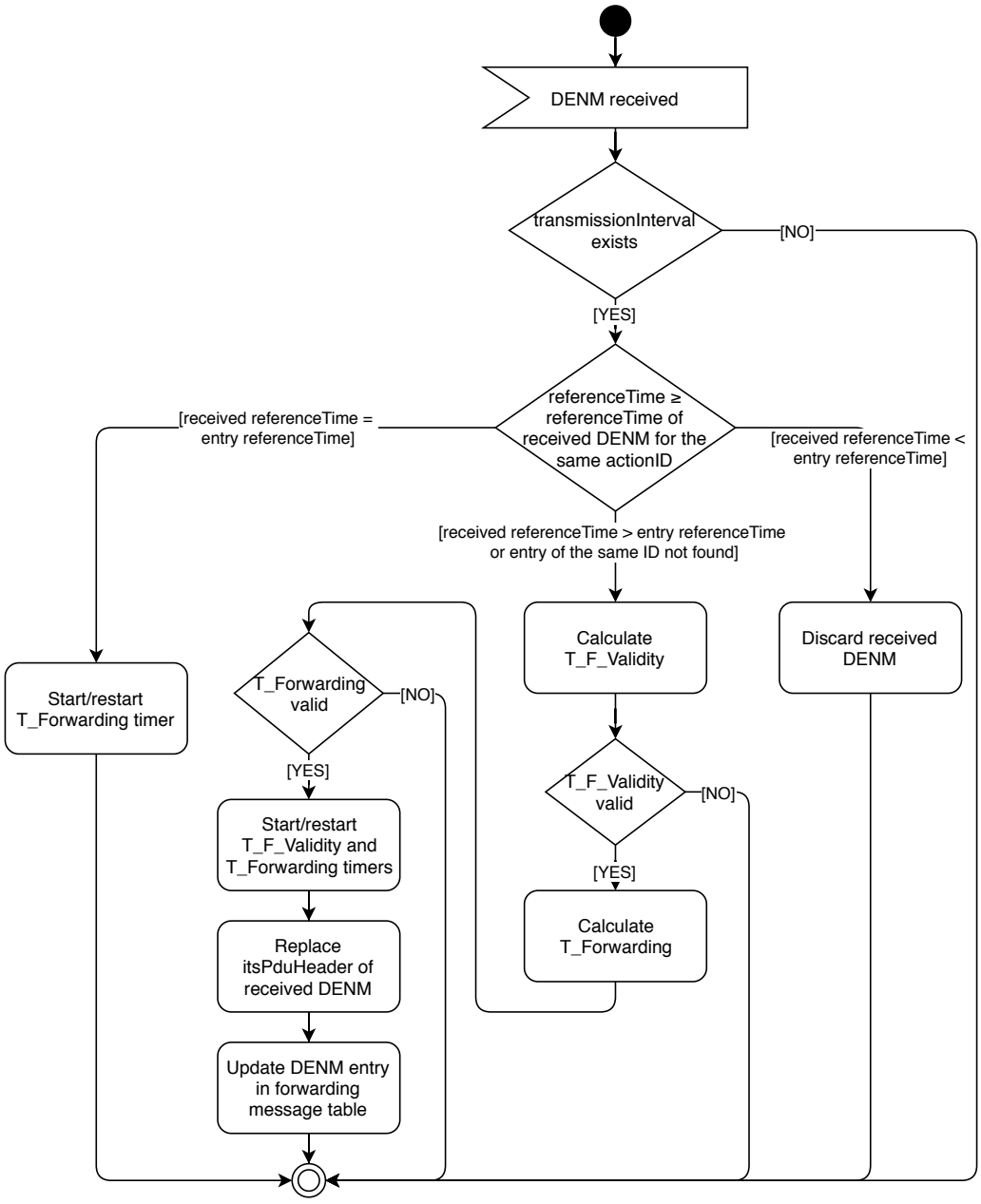


Figure E.7 – KAF operation

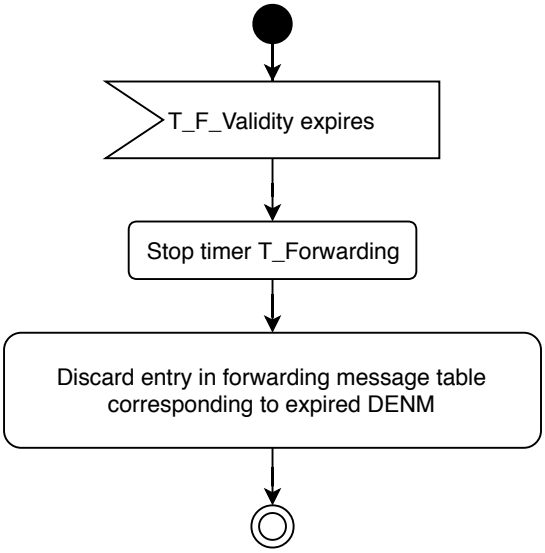


Figure E.8 – *T_F_Validity* expiration

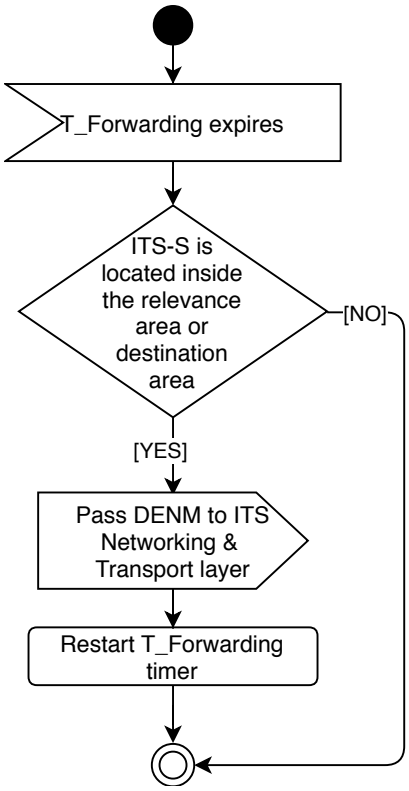


Figure E.9 – *T_Forwarding* expiration

E.3 DENM reception operation

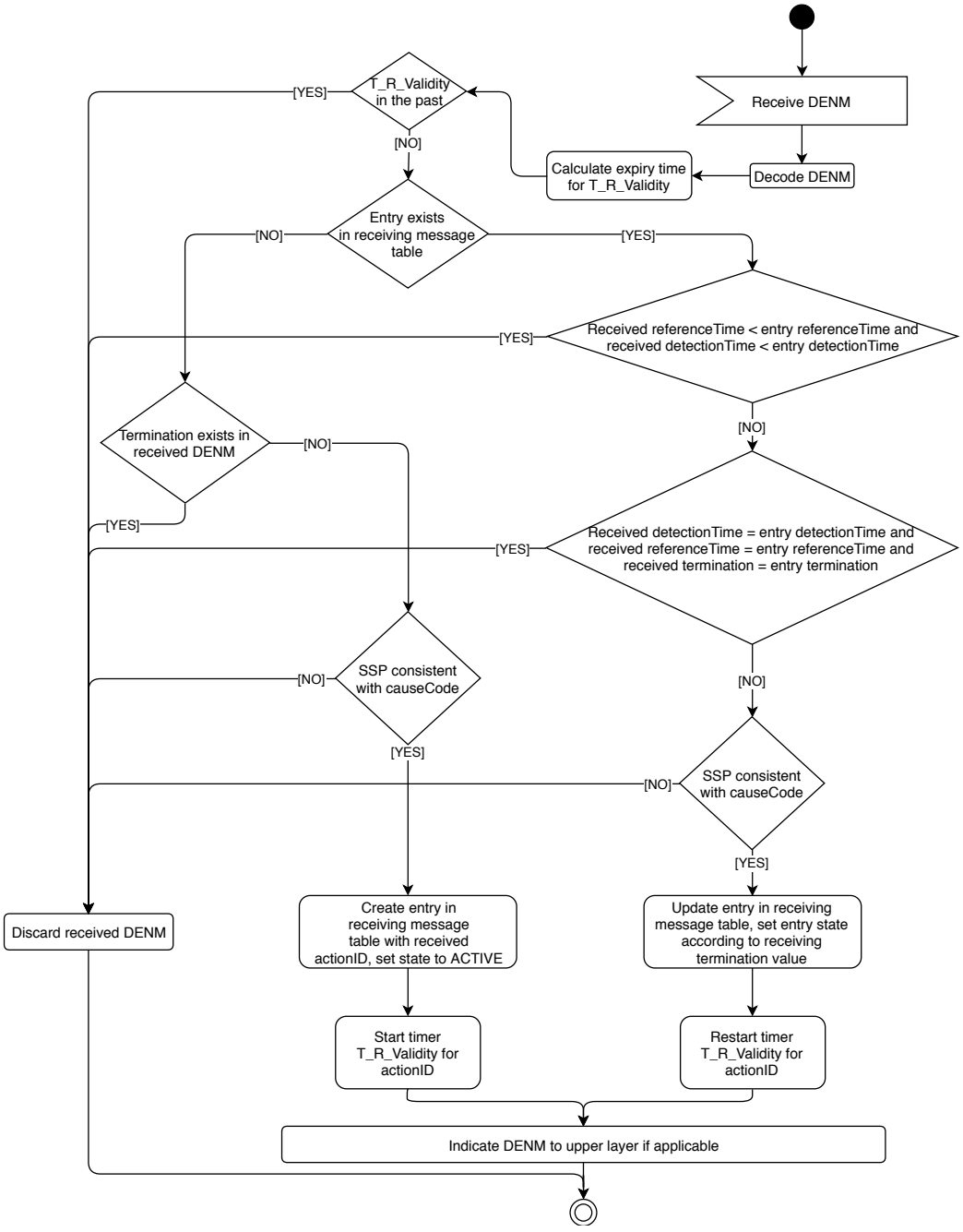


Figure E.10 – DENM reception operation

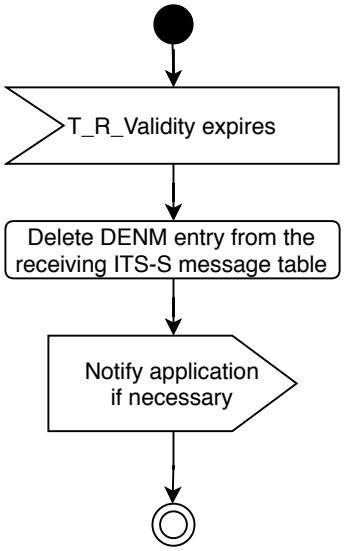


Figure E.11 – *T_R_Validity* expiration

Appendix F

ETSI ITS implementation's classes

F.1 Manager

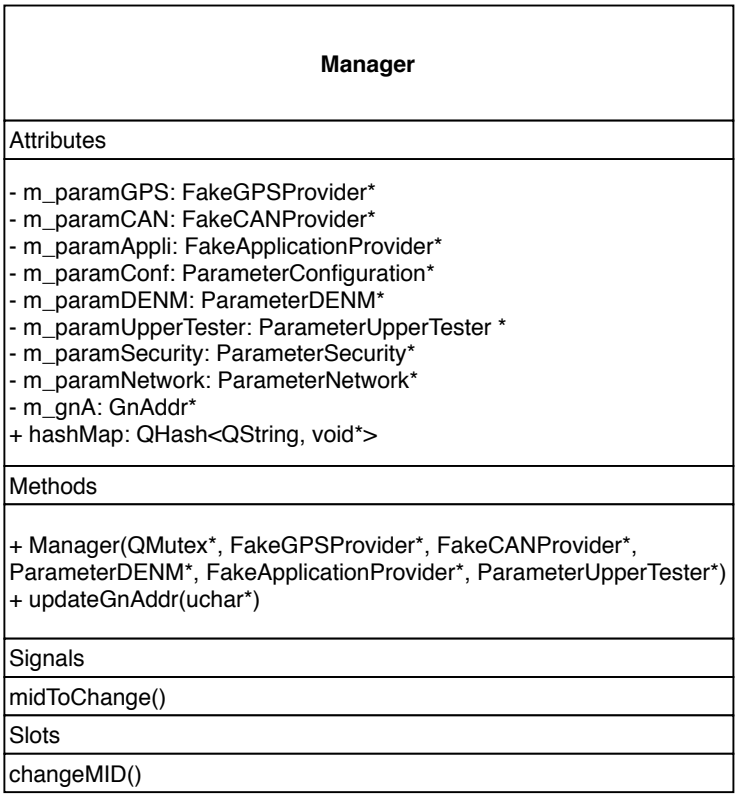


Figure F.1 – Manager class diagram

F.2 Sensor

(a) FakeGPSProvider class diagram

(b) FakeCANProvider class diagram

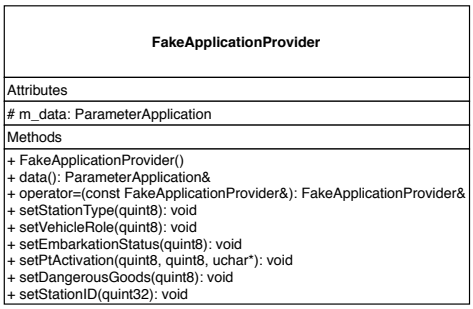
(c) ParameterCAN class diagram

(d) GPSProvider class diagram

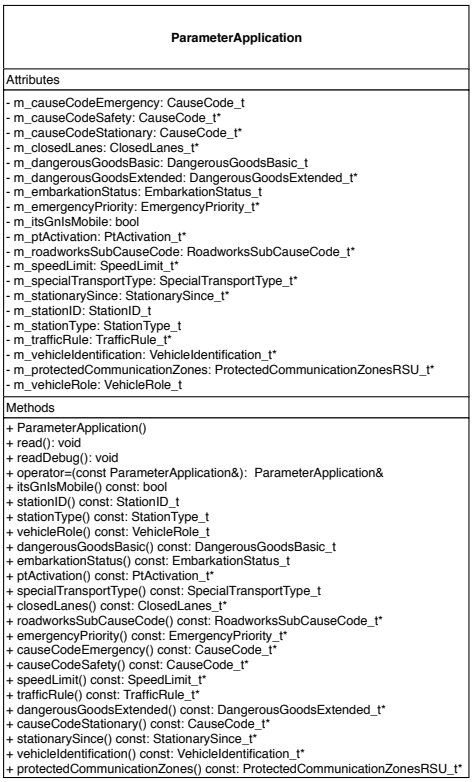
(e) ParameterGPS class diagram

Figure F.2 – Sensors class diagrams

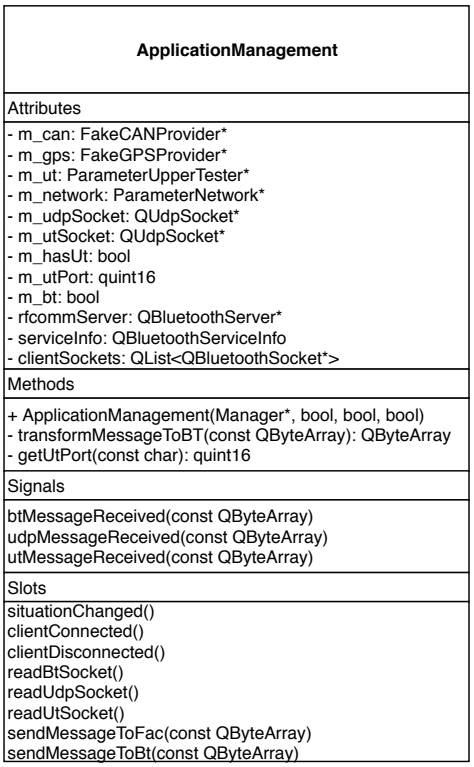
F.3 Application



(a) FakeApplicationProvider class diagram



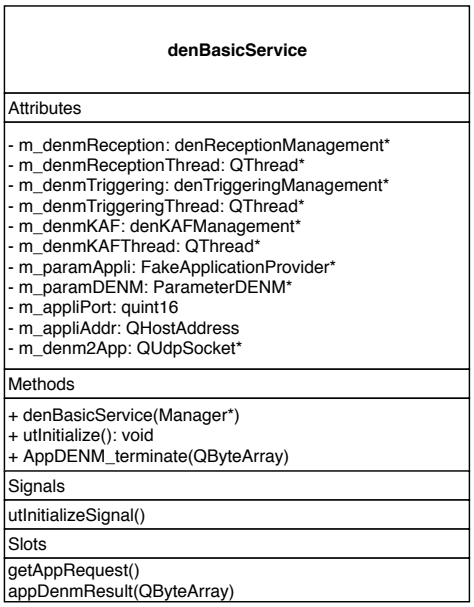
(b) ParameterApplication class diagram



(c) ApplicationManagement class diagram

Figure F.3 – Application layer class diagrams

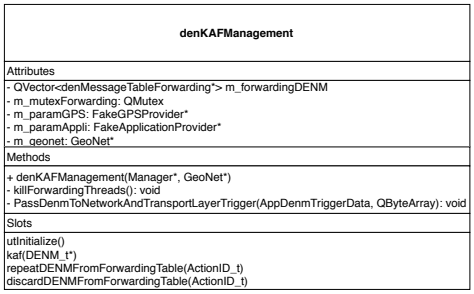
F.4 Facilities



(a) DEN Basic Service class diagram

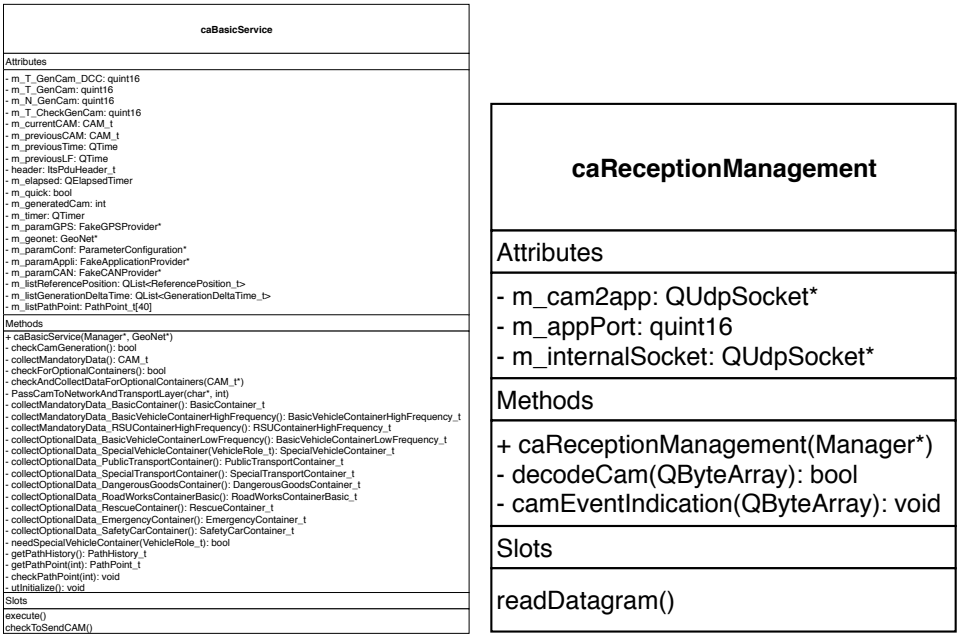
(b) DEN Triggering Management class diagram

(c) DEN Reception Management class diagram



(d) DEN KAF Management class diagram

Figure F.4 – DEN layer class diagrams

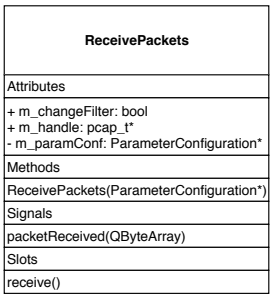


(a) CA Basic Service class diagram

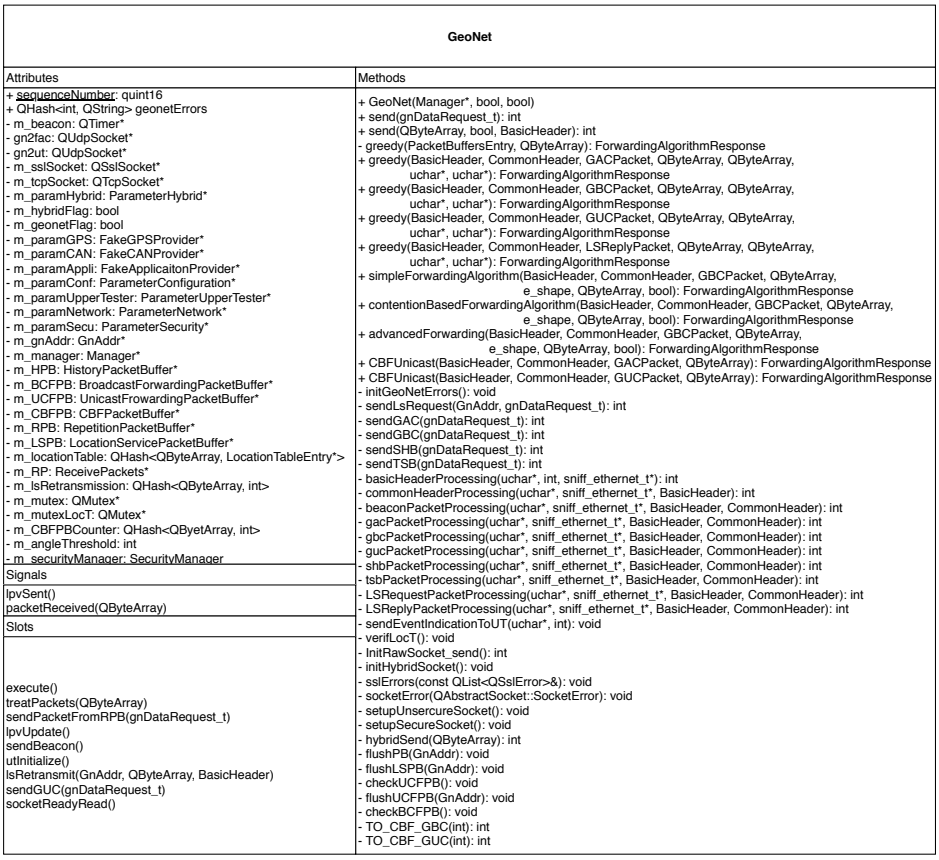
(b) CA ReceptionManagement class diagram

Figure F.5 – CA layer class diagrams

F.5 GeoNetworking



(a) ReceivePackets class diagram



(b) GeoNet class diagram

Figure F.6 – GeoNetworking layer class diagrams

F.6 UpperTester

Figure F.7 – UpperTester class diagram

Appendix G

Scoop@f optimization

G.1 Scoop@f extension

```
RSUContainerHighFrequency ::= SEQUENCE {
  protectedCommunicationZonesRSU ProtectedCommunicationZonesRSU OPTIONAL,
  ...,
  serviceAdvertismentContainer ServiceAdvertismentContainer,
  positionEnhancementContainer PositionEnhancementContainer,
  environmentAndContextContainer EnvironmentAndContextContainer
}

ServiceAdvertismentContainer ::= SEQUENCE {
  advertisedServiceItsAid AdvertisedServiceItsAid,
  serviceAccessCapabilities ServiceAccessCapabilities,
  channelUsedByTheAdvertisedService ChannelUsedByTheAdvertisedService,
  communicationProfileUsedForTheService CommunicationProfileUsedForTheService,
  rsuMacAddress RsuMacAddress,
  rsuIpAddress RsuIpAddress
}

AdvertisedServiceItsAid ::= INTEGER(0..255)

ServiceAccessCapabilities ::= BIT STRING {
  globalNetwork (0),
  continuousExchangeCapability (1),
  storeForwardCapability (2)
} (SIZE(8))

ChannelUsedByTheAdvertisedService ::= ENUMERATED {
  cch(0),
  sch1(1),
  sch2(2),
  sch3(3),
  sch4(4),
  sch5(5),
  sch6(6)
}

CommunicationProfileUsedForTheService ::= ENUMERATED {
  btpgeonet(0),
  tcpipv4(1),
  tcpiv6(2)
}

RsuMacAddress ::= OCTET STRING (SIZE(6))

RsuIpAddress ::= CHOICE {
  rsuipv4Andv6Address Ipv4Andv6,
  rsuiPv4Address IPv4Address,
  rsuiPv6Address IPv6Address
}

Ipv4Andv6 ::= SEQUENCE {
  rsuiPv4Address IPv4Address,
  rsuiPv6Address IPv6Address
}

IPv4Address ::= OCTET STRING (SIZE(4))
IPv6Address ::= OCTET STRING (SIZE(16))

PositionEnhancementContainer ::= SEQUENCE {
  gpsPositionDeltaLatitude DeltaLatitude OPTIONAL,
  gpsPositionDeltaLongitude DeltaLongitude OPTIONAL,
  gpsPositionDeltaAltitude DeltaAltitude OPTIONAL,
  satelliteConstellationLocallyAvailable SatelliteConstellationLocallyAvailable OPTIONAL,
  tracesLeadingToTheRsu TracesLeadingToTheRsu OPTIONAL
}
```

```
SatelliteConstellationLocallyAvailable ::= INTEGER(0..255)
TracesLeadingToTheRsu ::= SEQUENCE SIZE(1..7) OF PathHistory

EnvironmentAndContextContainer ::= SEQUENCE {
  localMeteorologicalData LocalMeteorologicalData OPTIONAL,
  roadEnvironment RoadEnvironment OPTIONAL,
  trafficCondition TrafficCondition OPTIONAL
}

LocalMeteorologicalData ::= BIT STRING (SIZE(8))
RoadEnvironment ::= INTEGER(0..255)
TrafficCondition ::= INTEGER(0..255)
```

G.2 Proposed extension

```
RSUContainerHighFrequency ::= SEQUENCE {
  protectedCommunicationZonesRSU ProtectedCommunicationZonesRSU OPTIONAL,
  ...,
  serviceAdvertismentContainer ServiceAdvertismentContainer OPTIONAL,
  positionEnhancementContainer PositionEnhancementContainer OPTIONAL,
  environmentAndContextContainer EnvironmentAndContextContainer OPTIONAL
}

ServiceAdvertismentContainer ::= SEQUENCE {
  rsuMacAddress RsuMacAddress,
  rsuIpAddress RsuIpAddress,
  ads ServiceAdvertisments
}

ServiceAdvertisments ::= SEQUENCE (SIZE(0..7)) OF ServiceAdvertisment

ServiceAdvertisment ::= SEQUENCE {
  advertisedServiceItsAid AdvertisedServiceItsAid,
  serviceAccessCapabilities ServiceAccessCapabilities,
  channelUsedByTheAdvertisedService ChannelUsedByTheAdvertisedService,
  communicationProfileUsedForTheService CommunicationProfileUsedForTheService
}

AdvertisedServiceItsAid ::= INTEGER {
  pki (0),
  tlogTransfert (1)
} (0..7)

ServiceAccessCapabilities ::= BIT STRING {
  globalNetwork (0),
  continuousExchangeCapability (1),
  storeForwardCapability (2)
} (SIZE(3))

ChannelUsedByTheAdvertisedService ::= ENUMERATED {
  cch(0),
  sch1(1),
  sch2(2),
  sch3(3),
  sch4(4),
  sch5(5),
  sch6(6)
}

CommunicationProfileUsedForTheService ::= ENUMERATED {
  btpGeonet(0),
  tcpIPv4(1),
  tcpIPv6(2)
}

RsuMacAddress ::= OCTET STRING (SIZE(6))

RsuIpAddress ::= SEQUENCE {
  rsuIPv4Address IPv4Address OPTIONAL,
  rsuIPv6Address IPv6Address OPTIONAL
}

IPv4Address ::= OCTET STRING (SIZE(4))
IPv6Address ::= OCTET STRING (SIZE(16))

PositionEnhancementContainer ::= SEQUENCE {
  gpsPositionDeltaLatitude DeltaLatitude OPTIONAL,
```

```
  gpsPositionDeltaLongitude DeltaLongitude OPTIONAL,
  gpsPositionDeltaAltitude DeltaAltitude OPTIONAL,
  satelliteConstellationLocallyAvailable SatelliteConstellationLocallyAvailable OPTIONAL,
  tracesLeadingToTheRsu Traces OPTIONAL
}

SatelliteConstellationLocallyAvailable ::= INTEGER(0..30)

EnvironmentAndContextContainer ::= SEQUENCE {
  localMeteorologicalData LocalMeteorologicalData OPTIONAL,
  roadEnvironment RoadEnvironment OPTIONAL,
  trafficCondition TrafficCondition OPTIONAL
}

LocalMeteorologicalData ::= BIT STRING (SIZE(8))
RoadEnvironment ::= INTEGER(0..255)
TrafficCondition ::= INTEGER(0..255)
```

G.3 TLog optimization proposition

```
Scoop-TLog-PDU-Description DEFINITIONS AUTOMATIC TAGS ::= BEGIN

IMPORTS

  CAMI FROM CAMI-PDU-Descriptions

  DENM FROM DENM-PDU-Descriptions {
    itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) en (302637) denm (1)
    version (1)
  }

  AccelerationControl, ActionID, CauseCode, DeltaReferencePosition, ExteriorLights, Heading,
  Latitude, Longitude, LongitudinalAcceleration, ReferencePosition, PositionOfOccupants, Speed,
  StationID, SteeringWheelAngle, Temperature, TimestampIts, YawRate
  FROM ITS-Container {
  itu-t (0) identified-organization (4) etsi (0) itsDomain (5) wg1 (1) ts (102894) cdd (2)
  version (1)
  }

  AccidentStatus, Actor, AirConditioningStatus, AutomobileSafetySystem, CalculationMode,
  ChannelOccupation, CountersChannel, DashboardIndication, DefaultCodes, DefrostingActivation,
  DistanceValue, DoorsStatus, Duration, ECallStatus, EnergeticCost, EquipmentCode, FaultyMessageCode,
  FrictionCoefficient, FunctioningStatus, HeatingLevel, IdentificationMode, IgnitionStatus,
  LengthClasses, LogSecurityEventType, LuminosityStatus, Neighborhood, NumberOfAuthentifiedMessage,
  Odometer, OriginallyQuery, Percentage, RadioInformation, Reliability, ScreenID, SeatBeltStatus,
  SendReception, StorageSize, StrengthBraking, SubActivityOperator, TotalDistanceTraveledSinceStart,
  TypeRequest, Username, VersionNumber, Wipers
  FROM Scoop-Container;

TLogHeader ::= SEQUENCE {
  logVersion      VersionNumber,
  softwareVersion VersionNumber,
  hardwareVersion VersionNumber,
  source          Actor,
  stationIDEgo    StationID
}

TLog-FaultyMessage ::= SEQUENCE {
  cause FaultyMessageCode
}

TLog-CAMI ::= SEQUENCE {
  send SendReception,
  cami CAMI
}

TLog-DENM ::= SEQUENCE {
  send SendReception,
  denm DENM
}

TLog-NetworkAccessPerformances ::= SEQUENCE {
  cntsCCH       CountersChannel,
  cntsSCH1      CountersChannel,
  busyRatioCCH  ChannelOccupation,
  busyRatioSCH1 ChannelOccupation
}
```

```
TLog-DatexIISituation ::= SEQUENCE {
  send                         SendReception,
  situationRecordCreationTime TimestampIts,
  situationVersionedId        IA5String (SIZE(1..45))
}

TLog-GeneralWorking ::= SEQUENCE {
  returnCode FunctioningStatus
}

TLog-ModulesWorking ::= SEQUENCE {
  itsG5State    FunctioningStatus,
  wifiState     FunctioningStatus,
  cellularState FunctioningStatus,
  gnssState     FunctioningStatus,
  hsmState      FunctioningStatus,
  tabletState   FunctioningStatus,
  storageUsed   Percentage,
  occupationRAM Percentage
}

TLog-Radio ::= SEQUENCE {
  channelCCH  RadioInformation,
  channelSCH1 RadioInformation
}

TLog-Configuration ::= SEQUENCE {
  equipmentCode          EquipmentCode,
  username               Username,
  establishmentLongitude Longitude,
  establishmentLatitude  Latitude,
  lengthClasses          LengthClasses,
  calculationMode        CalculationMode,
  speedLengthPeriod      Duration,
  classedDataPeriod      Duration,
  raisingStorageSizeMax  StorageSize,
  raisingPeriod          Duration
}

TLog-SecurityIncident ::= SEQUENCE {
  eventType     LogSecurityEventType,
  neighbor      Neighborhood,
  eventTime     TimestampIts,
  eventPosition ReferencePosition
}

TLog-Security ::= SEQUENCE {
  numberOfAuthentifiedMessage NumberOfAuthentifiedMessage,
  signatureDuration           Duration,
  signatureCheckDuration      Duration,
  validityCheckDuration       Duration,
  signatureCost               EnergeticCost,
  signatureCheckCost          EnergeticCost
}

TLog-ObjetsPKI ::= SEQUENCE {
  originallyQuery   OriginallyQuery OPTIONAL,
  typeRequest       TypeRequest,
  timestampRequest  TimestampIts,
  successfulRequest BOOLEAN,
  timestampResponse TimestampIts,
  keyGenerationCost EnergeticCost
}

TLog-DataStation ::= SEQUENCE {
  egoMapPosition                     ReferencePosition OPTIONAL,
  egoPosition                        ReferencePosition OPTIONAL,
  speed                              Speed OPTIONAL,
  yawRate                            YawRate OPTIONAL,
  wipers                             Wipers OPTIONAL,
  automobileSafetySystem             AutomobileSafetySystem OPTIONAL,
  distanceTraveledSinceMotorStarted TotalDistanceTraveledSinceStart OPTIONAL,
  longitudinalAcceleration           LongitudinalAcceleration OPTIONAL,
  accidentStatus                     AccidentStatus OPTIONAL,
  seatBeltStatus                     SeatBeltStatus OPTIONAL,
  eCallStatus                        ECallStatus OPTIONAL,
  frictionCoefficient                FrictionCoefficient OPTIONAL,
  openDoorsStatus                    DoorsStatus OPTIONAL,
  visibilityDistance                 DistanceValue OPTIONAL,
  positionOfOccupants                PositionOfOccupants OPTIONAL,
  defrostingActivation               DefrostingActivation OPTIONAL,
  airConditioningStatus              AirConditioningStatus OPTIONAL,
```

```
  heating                          HeatingLevel OPTIONAL,
  accelerationControl              AccelerationControl,
  steeringWheelAngle               SteeringWheelAngle,
  strengthBraking                  StrengthBraking,
  exteriorLights                   ExteriorLights,
  odometer                         Odometer,
  temperature                      Temperature OPTIONAL,
  ignition                         IgnitionStatus OPTIONAL,
  defaultCodes                     DefaultCodes OPTIONAL,
  indicationDashboard              DashboardIndication
}

TLog-DriverRequest ::= SEQUENCE {
  send        SendReception,
  actionID    ActionID OPTIONAL,
  screenID    ScreenID OPTIONAL,
  eventType   CauseCode,
  egoPosition ReferencePosition,
  reliability Reliability OPTIONAL
}

TLog-ClimaticEnvironmentContext ::= SEQUENCE {
  luminosity LuminosityStatus,
  defrosting DefrostingActivation
}

TLog-MapProjectionContext ::= SEQUENCE {
  actionID             ActionID,
  carMapPosition       ReferencePosition,
  eventMapPosition     ReferencePosition,
  correction           DeltaReferencePosition,
  referencePosition    ReferencePosition,
  eventPosition        ReferencePosition,
  eventPositionHeading Heading OPTIONAL
}

TLog-StateChangement ::= SEQUENCE {
  quitActivityIdentification IdentificationMode,
  operatorSubActivity        SubActivityOperator,
  snowplowBlade              BOOLEAN,
  flr                        BOOLEAN,
  flu                        BOOLEAN,
  workingSignal              BOOLEAN,
  gritter                    BOOLEAN,
  revolvingLight             BOOLEAN,
  punch                      BOOLEAN,
  externalTemperature        Temperature OPTIONAL,
  degradeMode                BOOLEAN
}

TLogFile ::= SEQUENCE {
  header TLogHeader,
  logs   TLogs
}

TLogs ::= SEQUENCE OF TLog

TLog ::= SEQUENCE {
  collectionTime TimestampIts,
  log            TLogType
}

TLogType ::= CHOICE{
  cami                        TLog-CAMI,
  climaticEnvironmentalContext TLog-ClimaticEnvironmentalContext,
  configuration               TLog-Configuration,
  dataStation                 TLog-DataStation,
  datexIISituation            TLog-DatexIISituation,
  denm                        TLog-DENM,
  driverRequest               TLog-DriverRequest,
  faultyMessage               TLog-FaultyMessage,
  generalWorking              TLog-GeneralWorking,
  mapProjectionContext        TLog-MapProjectionContext,
  modulesWorking              TLog-ModulesWorking,
  networkAccessPerformances   TLog-NetworkAccessPerformances,
  objetsPki                   TLog-ObjetsPKI,
  radio                       TLog-Radio,
  security                    TLog-Security,
  securityIncident            TLog-SecurityIncident,
  stateChangement             TLog-StateChangement,
  ...
}
END
```

Appendix H

k Nearest Neighbor Using Local Density Implementation

For the algorithms, we use a Sample_t structure defined as follows:

```
Sample_t:
{
    sample: integer
    neighborhood: list of integer
}
```

H.1 Sequential Algorithms

Algorithm 6 Samples

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- inputData: list of N lists of p floats, if the normalized matrix with the data
-- alpha: float, is the density of the neighborhood
-- rpz: list of struct Sample_t, is the list of selected samples and their neighborhood
-- k: integer, is the density parameter

-- append: function, is a function appending a Sample_t structure to a list of Sample_t structure
-- Sample: function, is the Algorithm 7, returning a Sample_t structure
-- purgeSamples: function, is the Algorithm 15, removing the selected sample and its neighborhood, returning the new number of data
-- normalize: function, is the Algorithm 16

alpha ← k/N
while N > 1 do
    rpz.append(Sample(N, p, inputData, alpha))
    N ← purgeSamples(inputData, rpz)
    if N > 1 then
        normalize(N, p, inputData)
    end if
end while
return rpz
```

H.2 Parallelized Algorithms

Algorithm 7 Sample

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- inputData: list of N lists of p floats, if the normalized matrix with the data
-- alpha: float, is the density of the neighborhood
-- distances: list of p lists of N lists of N floats, is the distances for each data, for each variable
-- neighbors: list of p lists of N lists of N integers, is the neighborhood for each data, for each variable
-- densities: list of N lists of p integers, is the densities of data for each variable
-- sumDensities: list of N integers, is the sum of densities for each data
-- max: integer, is the number of the denser data, i.e. the selected sample
-- variableNeighbors: list of N lists of p integers, is the neighborhood for each variable of the selected data
-- inter: list of integers, is the intersection of the neighborhood of the selected data
-- rpz: Sample_t structure, is the selected sample with its neighborhood
-- i: integer, is an iterator

-- euclidean: function, Algorithm 8, returning the distances for each data, for each variable
-- neighborhood: function, Algorithm 9, returning the neighborhood for each data, for each variable
-- density: function, Algorithm 10, returning the densities of data for each variable
-- sumDensity: function, Algorithm 11, returning the sum of densities for each data
-- maximum: integer, function, Algorithm 12, returning the number of the denser data, i.e. the selected sample
-- variableNeighborhood: function, Algorithm 13, returning the neighborhood for each variable of the selected data
-- intersection: function, Algorithm 14, returning the intersection of the neighborhood of the selected data
-- num: function, returns the number of the data
-- append: function, is a function appending an integer to a list

distances ← euclidean(N, p, inputData)
neighbors ← neighborhood(N, p, distances, alpha)
densities ← density(N, p, neighbors)
sumDensities ← sumDensity(N, p, densities)
max ← maximum(N, sumDensities)
variableNeighbors ← variableNeighborhood(N, p, max, neighbors)
inter ← intersection(N, p, variableNeighbors)
rpz.sample ← num(inputData[max])
for i ← 0 to N do
    if inter[i] = 1 then
        rpz.neighborhood.append(num(inputData[i]))
    end if
end for
return rpz
```

Algorithm 8 Euclidean

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- inputData: list of N lists of p floats, if the normalized matrix with the data
-- distances: list of p lists of N lists of N float, is the distances for each data, for each variable
-- v, n, i: integers, are some iterators

-- fabs: function, return the absolute value of a float value

for v ← 0 to p do
    for n ← 0 to N do
        for i ← 0 to N do
            distances[v][n][i] ← fabs(inputData[i][v] - inputData[j][v])
        end for
    end for
end for
return distances
```

Algorithm 9 Neighborhood

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- distances: list of p lists of N lists of N float, is the distances for each data, for each variable
-- alpha: float, is the density of the neighborhood
-- neighbors: list of p lists of N lists of N integers, is the neighborhood for each data, for each variable
-- v, n, i: integers, are some iterators

for v ← 0 to p do
    for n ← 0 to N do
        for i ← 0 to N do
            if distances[v][n][i] ⩽ alpha then
                neighbors[v][n][i] ← 1
            else
                neighbors[v][n][i] ← 0
            end if
        end for
    end for
end for
return neighbors
```

Algorithm 10 Density

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- neighbors: list of p lists of N lists of N integers, is the neighborhood for each data, for each variable
-- densities: list of N lists of p integers, is the densities of data for each variable
-- v, n, i: integers, are some iterators

for v ← 0 to p do
   for n ← 0 to N do
      densities[n][v] ← 0
      for i ← 0 to N do
         densities[n][v] ← densities[n][v] + neighbors[v][n][i]
      end for
   end for
end for
return densities
```

Algorithm 11 SumDensity

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- densities: list of N lists of p integers, is the densities of data for each variable
-- sumDensities: list of N integers, is the sum of densities for each data
-- n, v: integers, are some iterators

for n ← 0 to N do
   sumDensities[n] ← 0
   for v ← 0 to p do
      sumDensities[n] ← sumDensities[n] + densities[n][v]
   end for
end for
return sumDensities
```

Algorithm 12 Maximum

```
-- N: integer, is the number of data in the matrix
-- sumDensities: list of N integers, is the sum of densities for each data
-- max: integer, is the maximum value in sumDensities
-- idx: integer, is the index of the maximum value in sumDensities
-- n: integer, is an iterator

max ← sumDensities[0]
idx ← 0
for n ← 1 to N do
   if sumDensities[n] > max then
      max ← sumDensities[n]
      idx ← n
   end if
end for
return idx
```

Algorithm 13 VariableNeighborhood

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- max: integer, is the maximum value in sumDensities
-- neighbors: list of p lists of N lists of N integers, is the neighborhood for each data, for each variable
-- variableNeighbors: list of N lists of p integers, is the neighborhood for each variable of the selected data
-- n, v: integers, are some iterators

for n ← 0 to N do
   for v ← 0 to p do
      variableNeighbors[n][v] ← neighbors[n][max][v]
   end for
end for
return variableNeighbors
```

Algorithm 14 Intersection

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- variableNeighbors: list of N lists of p integers, is the neighborhood for each variable of the selected data
-- inter: list of integers, is the intersection of the neighborhood of the selected data
-- n, v: integers, are some iterators

for n ← 0 to N do
    inter[n] ← 1
    for v ← 0 to p do
        inter[n] ← inter[n] × variableNeighbors[n][v]
    end for
end for
return inter
```

Algorithm 15 PurgeSamples

```
-- inputData: list of N lists of p floats, if the normalized matrix with the data
-- rpz: list of struct Sample_t, is the list of selected samples and their neighborhood
-- n, i: integers, are some iterators

-- removeAt: function, remove the data in a certain index
-- length: function, returns a list length
i ← 0
for n ← 0 to rpz.neighborhood.length() do inputData.removeAt(rpz.neighborhood[n] - i)  -- The removeAt function
moves the data inside the list i ← i + 1
end for
return rpz.neighborhood.length()
```

Algorithm 16 Normalize

```
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- inputData: list of N lists of p floats, if the normalized matrix with the data
-- max: integer, is the maximum value of a variable for a data
-- min: integer, is the minimum value of a variable for a data
-- diff: integer, is the difference between maximum and minimum
-- n, v: integers, are some iterators

for v ← 0 to p do
    max ← inputData[0][v]
    min ← inputData[0][v]
    for n ← 1 to N do
        if max < inputData[n][v] then
            max ← inputData[n][v]
        end if
        if min > inputData[n][v] then
            min ← inputData[n][v]
        end if
    end for
    diff ← max - min
    if diff = 0 then
        diff ← 1
    end if
    for n ← 0 to N do
        inputData[n][v] ← (inputData[n][v] − min) / diff
    end for
end for
```

Algorithm 17 Parallelized Sample

Require: threadNumber: integer, the number (starting by 0) of the thread executing the algorithm
-- N: integer, is the number of data in the matrix
-- p: integer, is the number of variables in the matrix
-- inputData: list of N lists of p floats, if the normalized matrix with the data
-- alpha: float, is the density of the neighborhood
-- distances: list of p lists of N lists of N floats, is the distances for each data, for each variable
-- neighbors: list of p lists of N lists of N integers, is the neighborhood for each data, for each variable
-- densities: list of N lists of p integers, is the densities of data for each variable
-- sumDensities: list of N integers, is the sum of densities for each data
-- max: integer, is the maximum value of the sums
-- i_max: integer, is the number of the denser data, i.e. the selected sample
-- variableNeighbors: list of N lists of p integers, is the neighborhood for each variable of the selected data
-- inter: list of integers, is the intersection of the neighborhood of the selected data
-- rpz: Sample_t structure, is the selected sample with its neighborhood
-- varChunk: integer, is the number of variables computed by the thread
-- dataChunk: integer, is the number of data computed by the thread
-- varStart: integer, is the variable number for which the thread starts to compute
-- varEnd: integer, is the variable number (exclusive) for which the thread ends to compute
-- dataStart: integer, is the data number for which the thread starts to compute
-- dataEnd: integer, is the data number (exclusive) for which the thread ends to compute
-- NTHREADS: integer, is the number of threads
-- i, j, n, v: integer, is an iterator
-- init: boolean, used to know if someone is initializing data

-- threadBarrier: function, the thread waits until all thread reach this function
-- tryLock: function, the thread tries to lock a mutex, if it can (returns value = TRUE), it locks it, otherwise it continues
-- unlock: function, the thread unlocks a mutex
-- fabs: function, return the absolute value of a float value

-- Work setting
init ← FALSE
if NTHREADS > p **then**
  varChunk ← 1
else
  varChunk ← $\frac{\text{p}}{\text{NTHREADS}}$
end if
if NTHREADS > N **then**
  dataChunk ← 1
else
  dataChunk ← $\frac{\text{N}}{\text{NTHREADS}}$
end if
varStart ← threadNumber × varChunk
dataStart ← threadNumber × dataChunk
varEnd ← (threadNumber + 1)× varChunk
dataEnd ← (threadNumber + 1)× dataChunk
if threadNumber = (NTHREADS - 1) **then**
  varEnd ← p
  dataEnd ← N
end if
if threadNumber ⩾ p **then**
  varStart ← threadNumber
  varEnd ← threadNumber
end if
if threadNumber ⩾ N **then**
  dataStart ← threadNumber
  dataEnd ← threadNumber
end if
-- Euclidean algorithm
for v ← varStart to varEnd **do**
  for i ← 0 to N **do**
    for j ← 0 to N **do** distances[v][i][j] ← fabs(inputData[i][v] - inputData[j][v])
    end for
  end for
end for
threadBarrier()
-- Neighborhood algorithm
for v ← varStart to varEnd **do**
  for i ← 0 to N **do**
    for j ← 0 to N **do**
      if distances[v][n][i] ⩽ alpha **then**
        neighbors[v][n][i] ← 1
      else
        neighbors[v][n][i] ← 0
      end if
    end for
  end for
end for
threadBarrier()

```
 -- Densities setting
if tryLock() then
   if init = FALSE then
      init ← TRUE
      for i ← 0 to N do
         for j ← 0 to p do
            densities[i][j] ← 0
         end for
      end for
   end if
   unlock()
end if
threadBarrier()
init ← FALSE
 -- Densities algorithm
for v ← varStart to varEnd do
   for n ← 0 to N do
      densities[n][v] ← 0
      for i ← 0 to N do
         densities[n][v] ← densities[n][v] + neighbors[v][n][i]
      end for
   end for
end for
threadBarrier()
 -- sumDensity algorithm
for n ← dataStart to dataEnd do
   sumDensities[n] ← 0
   for v ← 0 to p do
      sumDensities[n] ← sumDensities[n] + densities[n][v]
   end for
end for
threadBarrier()
 -- Maximum setting
if tryLock() then
   if init = FALSE then
      init ← TRUE
      max ← sumDensities[0]
      i_max ← 0
      for i ← 1 to N do
         if sumDensities[i] > max then
            max ← sumDensities[i]
            i_max ← i
         end if
      end for
   end if
   unlock()
end if
threadBarrier()
init ← FALSE
 -- VariableNeighborhood algorithm
for n ← dataStart to dataEnd do
   for v ← 0 to p do
      variableNeighbors[n][v] ← neighbors[n][max][v]
   end for
end for
threadBarrier()
 -- Intersection algorithm
for n ← dataStart to dataEnd do
   inter[n] ← 1
   for v ← 0 to p do
      inter[n] ← inter[n] × variableNeighbors[n][v]
   end for
end for
threadBarrier()
 -- Sample selection
if tryLock() then
   if init = FALSE then
      init ← TRUE
      rpz.sample ← inputData[i_max].id
      for i ← 0 to N do
         if inter[i] then rpz.neighborhood.append(inputData[i].id)
         end if
      end for
   end if
   unlock()
end if
```

Algorithmes de Big Data adaptés aux VANET pour modélisation de comportement de conducteur

Les technologies Big Data gagnent de plus en plus d'attentions de communautés de recherches variées, surtout depuis que les données deviennent si volumineuses, qu'elles posent de réels problèmes, et que leurs traitements ne sont maintenant possibles que grâce aux grandes capacités de calculs des équipements actuels. De plus, les réseaux véhiculaires, aussi appelés VANET pour *Vehicular Ad-hoc Networks*, se développent considérablement et ils constituent une part de plus en plus importante du marché du véhicule. La topologie de ces réseaux en constante évolution est accompagnée par des données massives venant d'un volume croissant de véhicules connectés.

Dans cette thèse, nous discutons dans notre première contribution des problèmes engendrés par la croissance rapide des VANET, et nous étudions l'adaptation des technologies liées aux Big Data pour les VANET. Ainsi, pour chaque étape clé du Big Data, nous posons le problème des VANET.

Notre seconde contribution est l'extraction des caractéristiques liées aux VANET afin d'obtenir des données provenant de ceux-ci. Pour ce faire, nous discutons de comment établir des scénarios de tests, et comment émuler un environnement afin, dans un premier temps, de tester une implémentation dans un environnement contrôlé, avant de pouvoir effectuer des tests dans un environnement réel, afin d'obtenir de vraies données provenant des VANET.

Pour notre troisième contribution, nous proposons une approche originale de la modélisation du comportement de conducteur. Cette approche est basée sur un algorithme permettant d'extraire des représentants d'une population, appelés exemplaires, en utilisant un concept de densité locale dans un voisinage.

Big Data, réseaux véhiculaires, exemplaires, IoT, STIC

Big Data Algorithms Adapted to VANET for Driver's Behavior Modeling

Big Data is gaining lots of attentions from various research communities as massive data are becoming real issues and processing such data is now possible thanks to available high-computation capacity of today's equipment. In the meanwhile, it is also the beginning of Vehicular Ad-hoc Networks (VANET) era. Connected vehicles are being manufactured and will become an important part of vehicle market. Topology in this type of network is in constant evolution accompanied by massive data coming from increasing volume of connected vehicles in the network.

In this thesis, we handle this interesting topic by providing our first contribution on discussing different aspects of Big Data in VANET. Thus, for each key step of Big Data, we raise VANET issues.

The second contribution is the extraction of VANET characteristics in order to collect data. To do that, we discuss how to establish tests scenarios, and to how emulate an environment for these tests. First we conduct an implementation in a controlled environment, before performing tests on real environment in order to obtain real VANET data.

For the third contribution, we propose an original approach for driver's behavior modeling. This approach is based on an algorithm permitting extraction of representatives population, called samples, using a local density in a neighborhood concept.

Big Data, vehicular networks, samples, IoT, STIC

Discipline : INFORMATIQUE

Spécialité : Informatique

Université de Reims Champagne-Ardenne

CReSTIC - EA 3804

UFR Sciences Exactes et Naturelles,
Moulin de la Housse, 51867 Reims