Misbehavior Detection in Vehicular Ad Hoc Networks

A thesis submitted in partial fulfillment of the
requirements for the degree of

Bachelor of Technology (Honours)

in

Computer Science and Engineering

by

Ashish Vulimiri (05CS1018)

advised by

Dr. Arobinda Gupta

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

May 2009

Certificate

This is to certify that the thesis titled **Misbehavior Detection in Vehicular Ad Hoc Networks** submitted by Ashish Vulimiri (05CS1018) to the Department of Computer Science and Engineering is a bonafide record of work carried out by him under my supervision and guidance. This thesis has fulfilled all the requirements as per the regulations of the Institute and, in my opinion, has reached the standard needed for submission.

Dr. Arobinda Gupta
Associate Professor
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
May 2009

Contents

List of Figures

Chapter 1

Introduction

A Mobile Ad Hoc Network, or MANET, is a wireless network consisting of mobile nodes in which all the network-level activity is carried out by the nodes themselves, without additional infrastructure support. Each node in the network acts both as a router and as an endpoint – that is, all nodes in the network participate in message forwarding. Since the nodes in the network are mobile, the topology of the network is not static and changes with time.

A Vehicular Ad Hoc Network, or VANET, is a special kind of MANET in which the mobile nodes are all vehicles equipped with an On-Board Unit (OBU) that enable them to send and receive messages from and to the other nodes in the network. In addition to communication among the vehicles, a VANET might also interface with communication points provided by on-road infrastructure.

Several applications have been proposed for VANETs. Bai et. al. [5] survey these applications and divide them into three categories:

1. **Commercial applications**, in which the the VANET is used to provide commercial services such as streaming audio and video, internet access etc.

2. **Convenience applications**, in which the OBUs communicate with the other vehicles and with infrastructure points in order to provide the driver such services as traffic flow information, parking availabity in-

formation, automatic toll payment etc.

3. **Safety applications**, in which the VANET is used to identify conditions that could potentially endanger the driver's safety.

In this thesis, we shall be concerned entirely with safety applications. Some examples of safety applications in VANETs include:

1. **Post-Crash Notification**: A vehicle involved in a crash broadcasts a PCN alert to the vehicles in its vicinity to inform them of the existence and the location of a crash, thus enabling them to take evasive action.

2. **Slow or Stopped Vehicle Advisory**: An SVA alert is broadcast by a vehicle that suddenly slows down or stops on the road, again enabling the recipients to take evasive action.

3. **Road Hazard Condition Notification**: A vehicle that observes a possibly hazardous condition on the road, such as road slipperiness, broadcasts an RHCN alert warning the other vehicles about this condition.

4. **Road Feature Notification**: A vehicle that observes a road feature, such as a sharp curve or a hill, that would require a driver to take corrective action, broadcasts an RFN alert informing the other vehicles about this feature.

A notification of a safety condition received over the network may or may not be correct. When incorrect, the notification might be false – that is, the notification might indicate a safety condition that does not exist. This can happen in applications (such as the SVA application) that are binary in nature – either the indicated safety condition exists, or it does not. Alternately, in applications (such as the RHCN application) that are not binary in nature – that is, where the notification specifies a value in addition to indicating the existence of a condition – it is also possible that the condition does exist but that the indicated value is inaccurate.

An incorrect notification may be caused either by malfunctioning equipment, or by deliberate malicious behavior. For instance, a vehicle might attempt to use incorrect crash notifications in order to clear the road for itself. Irrespective of the cause, it is necessary that a vehicle receiving a safety notification have a mechanism in place to detect (and possibly correct) any possible inaccuracies in the notification. In what follows, we shall refer to inaccuracies in a safety notification as *misbehavior*, and the problem of identifying misbehavior as the *misbehavior detection* problem. Any method implemented by an OBU to solve the misbehavior detection problem shall be referred to as a *Misbehavior Detection Scheme* (or MDS).

1.1 Motivation

An OBU that needs to implement an MDS has three sources of information about the system that might help it in the construction of the MDS:

1. **Primary Information**: This refers to information obtained directly from the vehicle in which the OBU is installed. Primary information would include details of the driver's trajectory, and the readings from any sensors installed in the vehicle.

2. **Secondary Information**: This refers to unsolicited information received from other vehicles, in the form of messages over the VANET. Examples include the SVA and RHCN alerts detailed earlier.

3. **Information from collaboration**: This refers to information received through active collaboration between the vehicles, using message passing over the VANET.

While some prior work exists on using primary information to design an MDS for VANET applications [9], the use of secondary information and information from collaboration in the design of an MDS has not yet been studied. The goal of this thesis is to investigate the use of these two forms of information in the design of an MDS for VANET applications.

Three approaches may be taken to the use of secondary information in an MDS:

1. The secondary information may be used to augment the primary information in an existing primary information based MDS in order to improve its performance.

2. The secondary information may be used to directly construct an MDS in which the OBU passively receives information over the network, and uses it to reason about the possibility of misbehavior.

3. We may construct an MDS in which the OBUs from different vehicles actively collaborate with each other, using message passing over the network, in order to reason about the possibility of misbehavior.

   The design of an MDS that attempts to incorporate information from collaboration between the nodes in the VANET is complicated by two factors:

   a) The existence of malicious nodes.

   b) The possibility of messages being lost, due to problems at the nodes or due to losses over the network, and of nodes leaving the network, which is possible since the nodes in the network are all mobile.

This thesis shall investigate all three approaches separately.

The examples in this thesis shall focus on the PCN application. In the PCN application, a vehicle that is involved in a crash broadcasts a PCN alert to inform the vehicles in its vicinity of the existence and the location of the crash. The goal of the MDS in this application would be to identify false PCN alerts – alerts that are generated when there is no actual crash.

1.2 Contributions

This thesis makes the following contributions:

1. An existing primary information based MDS for the PCN application [9] is considered, and secondary information is used to improve its performance by constructing models of i) the degree of the driver's belief in the veracity of the alert, and ii) the effect of the variation in this belief on the primary information.

2. A secondary information model is defined and used to construct an MDS that is capable of using information received passively over the VANET to reason about the possibility of misbehavior. The PCN application is used as an example to illustrate how the parameters of this model can be estimated from historical and experimental data.

3. An MDS that uses information from collaboration between the nodes in the network is constructed, and analytical and experimental results quantifying its performance are presented.

1.3 Thesis Organization

The remaining chapters of this thesis are organized as follows:

Chapter 2 surveys past work on misbehavior detection in VANETs.

Chapter 3 discusses an existing primary information based MDS for the PCN application that bases its predictions on models of driver behavior, and describes how secondary information may be used to improve these behavioral models, and thus the accuracy of the MDS. Experimental evaluations of this improvement are presented.

Chapter 4 decouples the model of secondary information from the model of driver behavior described in chapter 3. An improved version of the secondary information model is presented and is used to define an MDS that is capable of directly utilizing secondary information. The estimation of the parameters of the secondary information model is detailed for the PCN application.

Chapter 5 discusses the problem of designing a collaborative MDS (an MDS in which the OBUs actively collaborate with each other over the VA-

NET), and its relation to the distributed consensus problem. Analytical results are described and experimental results for the PCN application are presented.

Chapter 6 summarizes the contributions of this thesis, and identifies possible directions for future research.

Chapter 2

Related Work

Bai et al. [6] present a characterization and classification of a number of important V2V applications.

The need for security in VANET applications has also been well-established. Several works [16][15][14][8] investigate the requirements and challenges involved in providing secure V2V communication, and propose general architectures for security in such scenarios. The 1609.2 standard [1] also proposes the functionalities of a security layer in V2V communication. However, though some of these works stress the need for misbehavior detection, no specific scheme for misbehavior detection is given for any application. The effect of Sybil attacks on V2V communication and position verification schemes to mitigate them have been discussed in [17] and [13]. However, these schemes are very specific to one type of attack only.

Golle et al. [10] present a model to integrate information from different sensors and use it to identify malicious information and malicious nodes. However, the algorithm for actual detection is only sketched through examples, and the computational aspects of the scheme are not investigated in detail.

A primary information based MDS for the PCN application is described by Ghosh et al. [9]. However, the MDS described here is based entirely on primary information and does not consider secondary information.

No work has yet been done on misbehavior detection using information

received over the VANET – either secondary information, or information from collaboration.

As shall be shown in Chapter 5, the problem of designing a collaborative MDS is closely related to, although distinct from, the distributed consensus problem[12]. In particular, we shall be concerned with a variant of the problem known as the approximate agreement problem[7]. While no solution to the approximate agreement problem has yet been described for general ad hoc networks such as the VANETs in our applications, the solution that we describe for the VANET scenario in this thesis builds on prior work on solutions to the approximate agreement problem for partially connected networks [11][2][3][4].

Not all types of possible modes of faults can occur in the VANET scenario. The fault model we use in our application was constructed based on a modification of the fault model proposed by Azadmanesh et al. [3].

Chapter 3

Using Secondary Information for Modelling Driver Belief

It was noted in Chapter 1 that an On-Board Unit attempting to reason about the occurrence of misbehavior has two sources of information about the system that may help it reach a decision:

1. Information about the driver's trajectory on the road. This information is always present, and is an example of **primary information**.

2. Messages received over the VANET, or **secondary information**. This information is dependent on the existence of external message sources (other vehicles or infrastructure points) in the vicinity of the vehicle, and thus may or may not always be present.

There are application scenarios in which the primary information may be used to reason about the possibility of misbehavior in the system. Consider the Post Crash Notification (PCN) application as an example. In the PCN application, a vehicle that is involved in a crash broadcasts a message informing the other vehicles in its vicinity of the existence and the location of the crash. Information about the driver trajectory may be used in this case to reason about the correctness of the alert – intuitively, if the vehicle does not veer as far from the alleged crash-site as it should, it is likely that the alert was inaccurate.

We concentrate on such application scenarios in this chapter and demonstrate how secondary information, if present, may be used to augment the primary information to improve the accuracy of the MDS. We use the primary information based MDS developed by Ghosh et al. [9] for the PCN application as an example.

3.1 MDS for PCN

For the sake of completeness, we describe here in brief the MDS proposed by Ghosh et al. [9] for the PCN application. This MDS uses precomputed descriptions of expected driver behavior to compute an expected driver trajectory in the presence of a crash, and then compares this expected trajectory to the actual path followed by the driver. If the deviation is larger than a certain threshold, misbehavior is declared.

The behavior of the driver is modelled using a discrete time Markov chain. The states of the Markov chain are the lanes on the road, the current state indicating the lane that the vehicle is currently in. We assume a discretized model where time is divided into slots and where lane-changes may occur only at the end of a timeslot. The transition probability between any two states i and j is the probability that a vehicle in the ith lane moves to the j lane in one timeslot.

Note that the transition probability matrix may not be a constant and might vary with time. We shall refer to the instantaneous value of the transition probability matrix at any given point of time as the "mobility model". The mobility model is of interest only in the time interval between the time the original PCN alert is received and the time the vehicle passes the alleged crash-site. The following parameters shall be used to define the variation in the mobility model:

1. The freeflow mobility model P, which indicates the behavior of the driver when no crash was observed.

2. The crash-site mobility model T, which is the mobility model that ap-

plies at the point of time when the vehicle is just passing the alleged crash-site.

3. A weight $\alpha(t)$, which indicates how close the driver's mobility model is to the crash-site mobility model. It is assumed that the mobility model at any intermediate point of time is a linear combination of P and T, and the parameter α is defined so that the mobility model that applies at time t is $(1-\alpha(t))P + \alpha(t)T$ if a crash has occurred.

4. $M(t) = (1-\alpha(t))P + \alpha(t)T$, the mobility model that applies at time t if a crash has occurred.

Thus, under the assumption that the driver, upon receiving the PCN alert, knows immediately if the alert is correct or incorrect, the mobility model is described by $M(t)$ if the PCN alert is true, and by P if the alert is false.

If P, T and α are known, the description of the mobility model $M(t)$ obtained from these three parameters may be used to compute an expected crash-modulated trajectory. The distance between the actual trajectory of the driver and this expected trajectory is computed using a Euclidean metric – if the actual and expected positions in the tth timeslot are respectively (x_t', y_t') and (x_t, y_t), the distance d is computed as $d = \sum_{t=1}^{t_0}[(x_t - x_t')^2 + (y_t - y_t')^2]$, where t_0 is the time at which the car reaches the crash site. Misbehavior is declared if d is larger than the set threshold ϵ.

Depending on the relation between the value of the distance d and the misbehavior detection threshold ϵ, and the occurrence or not of misbehavior, four cases are possible:

	Incorrect alert	**Correct alert**
$d > \epsilon$ **(Detected)**	True Positive	False Positive
$d < \epsilon$ **(Not detected)**	False Negative	True Negative

The MDS attempts to control the value of ϵ to minimize the probability of false positives and false negatives. More precisely, it attempts to minimize the *dis-utility function* $U(\epsilon) = (1-w)FP(\epsilon) + wFN(\epsilon)$, w being a weight that controls the relative importance of the possibility of a false negative against

the possibility of a false positive. This metric U shall be used later to compare the performance of this MDS against the improved version presented below that uses secondary information to augment the primary information.

3.2 Driver Belief

As was noted earlier, the above model assumes that the driver, on receiving the alert, knows immediately if it is correct or not. However, the evidence available to the driver may not always be sufficient to allow him/her to reach this decision immediately. Instead, the available evidence may only support a partial belief in the crash. In this case, the actual mobility model would take on an intermediate value between P and M, the exact model followed being dependent on the degree of the driver's belief in the crash. Note also that this degree of belief is time-variant, and changes as he/she keeps receiving information pertaining to the crash.

Our work concentrates on the use of secondary information in modelling this degree of belief and its effect on the driver's mobility model.

3.3 The Belief Estimate β

The degree of the driver's confidence in the occurrence of the crash is quantified by a numerical value β. To each time t, the belief function $\beta(t)$, assigns a value in the range $[0, 1]$ representing the degree of belief the driver has in the occurrence of the crash. The degree of belief is 1 when the driver is certain the crash has occurred, and is 0 when the driver is certain the crash has NOT occurred.

We use $B(t)$ to denote the belief-modulated mobility model. Note that B must satisfy the following two conditions:

1. $B(t) = P$ when $\beta(t) = 0$, and
2. $B(t) = M(t)$ when $\beta(t) = 1$.

We define B by $B(t) = (1 - \beta(t))P + \beta(t)M(t)$ (i.e. a linear variation over the belief spectrum), but alternate definitions are possible.

3.4 Defining β: Event Model

We define an event as any observation that provides some information about the likelihood of a crash. An event may be observed by either the system alone, by the driver alone, or by both the system and the driver (note that all events perceived by the system are alerts received via the VANET). The contribution of the event to the cumulative belief function depends on who it is observed by.

3.4.1 System-only Events

When the system receives an event that is known not to be received by the driver too, it may do one of two things. If the system is designed so that the driver path-deviation is the sole input used to determine whether the notification is invalid, then this event would just be discarded. However, the system may also be configured so that the information from all such events is used as an additional input. In such cases, the information from all such events would be fused into a single belief function $\beta_{\mathcal{E}_S}$, which would then be one of the inputs used while defining β. We neglect such cases for now, and assume all system-only events are discarded.

In Chapter 4, we expand on this and show how system-only events can still be used in applications where the primary information is not relevant.

3.4.2 Driver-only Events

Such events cannot be modelled directly. Instead, the system would represent the net effect of these events as a noise function $\beta_{\mathcal{E}_D}$ that would need to be combined into β. Information about the nature of such events may be used while defining the noise function. For instance, if most of these events are visual inputs, we may assume the contribution of $\beta_{\mathcal{E}_D}$ increases with

time, since the accuracy of visual input increases as the driver approaches the crash site.

3.4.3 Common Events

It is the class of events that are perceived by both the driver and the system that we are most concerned with. We explain common events in detail in section 3.5.

3.5 Common Events

We assume that all common events belong to one of a finite number of event classes, each class being some category of events. For instance, the PCN alert would define one event class, and all the SVA alerts (a slow or stopped vehicle advisory, or SVA, is an alert broadcast by a vehicle that suddenly slows down or stops) would be combined into another. Each event class is defined by a characteristic set of attributes that define the information content of each event in the class. For instance, the class of SVA alerts would be defined by the position of the vehicle that slowed down and the the distance of the car from the crash site at which the event occurred. If $a_1, a_2, a_3, ..., a_n$ be the set of attributes defining an event class, any single event in that class is merely an assignment of values to these attributes. The following attributes are common to all event classes and will usually not be listed explicitly in the class description:

1. t: Time
2. (x, y): Position of the source of the alert.

We also use the variables $(x_c(t), y_c(t))$ to denote the position of the destination car (the car that receives the alerts) at time t.

We now formally define event classes. An event class is an ordered tuple consisting of:

$D = D_1 \times D_2 \times D_3 \times \cdots \times D_n$**:**

The set of all possible events. The D_i are the domains of each attribute in the event class. In the function definitions below, u shall denote a variable ranging over the set of possible events D, and u_i a variable ranging over D_i.

$f(t)$**:**

The frequency profile of the event class (range = $\mathbb{N}$). $f(t)$ is the number of events in this class occurring at time t.

$p(t, u_1, u_2, u_3, ..., u_n)$**:**

At a given time t, $p(t)$ is a joint (n-variate) probability distribution over the attribute-space. Each point in the attribute space is an event, and the p-value of this event represents the probability of the event occurring at time t.

$b(t, u_1, u_2, u_3, ..., u_n)$**:**

Assigns a belief value to each event. $b(event)$ represents the contribution of this event to the belief estimate. The range of this function depends on the exact mechanism used for calculating belief (a single number in a Bayesian network, an ordered pair (β, β') in a Dempster-Shafer set etc).

$c(f, p, b)$**:**

Combination rule. Describes how to combine f, p and b to define the contribution of this event class to the belief function.

Intuitively, at a time t, $f(t)$ defines the number of events that occur, $p(t)$ defines *what* events occur, and $b(t)$ defines the contribution of each event that occurs to the belief estimate. These three functions are combined using the combination rule c to define the net contribution of this event class to the overall belief estimate. We denote by $\beta_E(x)$ the function representing the net contribution of the event class E to the belief estimate β.

3.6 Belief Function

Let $\mathcal{E}_C$ be the set of all classes of common events. Then, since we do not consider system only events, the belief function β is defined by

$$\beta = \left(\bigcirc_{E \in \mathcal{E}_C} \beta_E \right) \circ \beta_{\mathcal{E}_D}$$

where $\beta_{\mathcal{E}_D}$ is the noise function representing all driver-only events, as defined earlier.

3.7 Example

We now show how to estimate the belief function in a system consisting of two classes of events, the class containing the PCN event and the class of SVA alerts.

3.7.1 PCN Event Class

This class consists of a single event, viz. the initial PCN notification.

Attributes:

This event class has two attributes, both describing the position of the crash – lane number and crash site distance. We define $D_1 = 1, 2, ..., n$ as the domain of the attribute describing lane numbers, and $D_2 = \mathbb{R}^+$ as the domain of the attribute describing the distance to the crash site.

$f(t)$**:**

We have $f(0) = 1$, and $f(t) = 0$ for all other t

$p(t, u_1, u_2)$**:**

We have $p(0, n, D) = 1$, and $p = 0$ everywhere else, n being the lane number in which the crash occurred.

$b(t, u_1, u_2)$**:**

Let $b(0, n, D) = \beta_0$. The definition at all other points is irrelevant.

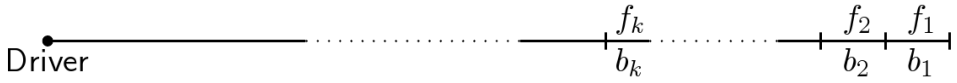

Figure 3.1: The SVA Event Model

Combination rule:

We combine f, p and b to define $\beta(t) = \beta_0$ for all t.

3.7.2 SVA Alert Class

As was mentioned earlier, a slow or stopped vehicle advisory, or SVA, is an alert broadcast by a vehicle that suddenly slows down or stops. We use the class of SVA alerts to improve our estimate of β, based on the intuition that a crash is more likely to actually have occurred if a significant number of vehicles are slowing down or stopping near the alleged crash-site. An SVA alert may originate from a vehicle at any distance from the crash site. An SVA alert from a vehicle closer to the crash site obviously needs to be assigned a higher belief-value than a vehicle farther off from the site. Now this variation in belief assignment may be either a continuous function of the distance y from the crash site, or may be a discrete variation defined by thresholds on y. We shall proceed with a discrete definition, but as is noted later, a continuous variation can also be handled fairly easily.

We assume k segments, with the ith segment being the region from $x = d_{i-1}$ to $x = d_i$ ($d_0 = 0$). Define b_i as the belief score assigned to the ith segment, and f_i as the frequency of the events originating in the ith segment.

Attributes:

This event class has a single attribute, the distance y from the crash site when the vehicle slowed down. Domain is $[0, D]$.

$f(t)$**:**

$$f(t) = \sum_{i=1}^{k} f_i$$

$p(t, y)$:

$$p(t, y) = \frac{f_i \times \left(\frac{y - d_i}{d_{i+1} - d_i}\right)}{\sum_{j=1}^{k} f_j}, d_i \leq y < d_{i+1}$$

$b(t, d)$:

$$b(t, y) = b_i, d_i \leq y < d_{i+1}$$

$c(f, p, b)$:

On moving a distance δ towards the crash site, the increase in the sum of all beliefs contributed by the SVA events can be shown to be[1]

$$\left(\sum_{i=1}^{k} f_i \times b_i\right) \times \delta$$

If the belief assignment was defined as a continuous function we would instead have

$$\left(\int_{D}^{0} f \times p(y) \times b(y) dy\right) \times \delta$$

Any monotonically increasing function (sublinear, linear or superlinear) of the sum would be a reasonable rule of combination. We choose a linear function with slope m. Therefore, if the initial belief is β_0, the belief keeps increasing linearly at the rate $m \times \sum_{i=1}^{k} (fi \times b_i)$ until it becomes 1, at which point it remains constant with the value 1.

Note that the above assumes the alert was correct. If, on the other hand, there was no crash, the SVA alerts would not contribute anything to the belief function β.

[1] We have a simple linear variation with δ because all three of f, p and b are independent of the parameter t.

3.7.3 Combination

In the combined belief function β, the PCN event class defines the initial belief β_0, and the SVA alert class defines the shape (linear in our case) and the rate of the growth in belief.

3.8 Experimental Results

We concentrate on the cases where the driver's initial belief β_0 (the degree of confidence the driver places initially in the PCN alert itself) is known. We present below some simulation results for the case when $\beta_0 = 0$.

In what follows, P, M and B shall denote, respectively, the freeflow mobility model, the crash modulated mobility model and the belief modulated mobility model. ϵ shall denote the threshold used in the trajectory comparison. $|X - \overline{Y}|$ shall denote the distance between one instance of the X-modulated trajectory, and the expected Y-modulated trajectory.

A false positive is said to occur when the PCN alert is in fact true, but the system wrongly marks it as false, and a false negative is said to occur when the PCN alert is false, but the system wrongly marks it as true. We use FP to indicate the probability of a false positive, and FN to indicate the probability of a false negative. Our objective is to find the value of the threshold ϵ that minimizes $U(\epsilon) = (1 - w) \times FP(\epsilon) + w \times FN(\epsilon)$, as was noted in section 3.1.

We consider three possible cases:

1. When neither the driver nor the system perceive any events.

   - **False Positive**: Neither system nor driver perceive that the alert is in fact true. The system expects the driver to follow the P-modulated trajectory, and the driver does as expected. $FP = P(|P - \overline{P}| < \epsilon)$
   - **False Negative**: For the same reason as above, the system expects the driver to follow the P-modulated trajectory, which the driver does. $FN = P(|P - \overline{P}| > \epsilon)$

2. When the driver perceives and receives information from an event-stream about the crash, but the system does not.

   - **False Positive**: The driver perceives the event stream and follows the B-modulated trajectory, but the system does not and thus expects the driver to follow the P-modulated trajectory. $FP = P(|B - \overline{P}| < \epsilon)$
   - **False Negative**: Since there is in fact no crash, the driver does not perceive any events and continues to follow the P-modulated trajectory, as is expected by the system. $FN = P(|P - \overline{P}| > \epsilon)$

3. When both driver and system receive secondary information about the crash.

   - **False Positive**: Since both driver and system perceive the event stream, the driver follows the B-modulated trajectory and the system expects the same. Here we again have two cases:
     a) B is closer to M than to P. In this case, $FP = P(|B - \overline{B}| > \epsilon)$
     b) B is closer to P than to M. In this case, $FN = P(|B - \overline{B}| < \epsilon)$
   - **False Negative**: No events are generated since there is in fact no crash. The system expects the driver to follow the P-modulated trajectory, and the driver does so. $FN = P(|P - \overline{P}| > \epsilon)$

We show here simulation results for the two subcases in case 3 separately.

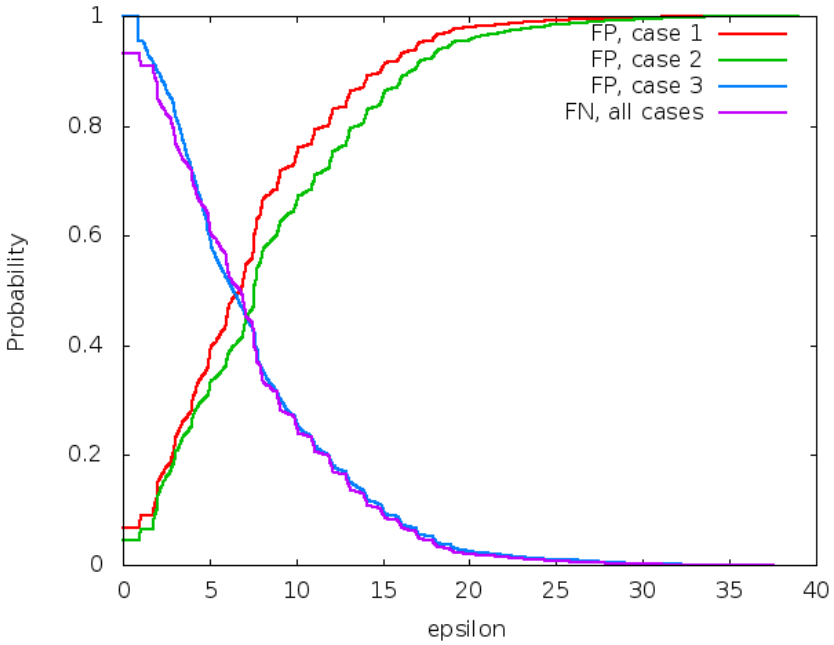

Figure 3.2: Utility variation with $\beta(t) = \sqrt{x(t)/D}$

Figure 3.2 shows the results for the first subcase, when B is closer to M than to P. Here we have $B(t) = (1 - \sqrt{x(t)/D})P + \sqrt{x(t)/D}M$, where $x(t)$ is the distance of the vehicle to the alleged crash site at time t. Since $x(t)/D < 1$, the function $\sqrt{x(t)/D}$ grows faster than linearly, so that B changes very quickly to M. This subcase represents situations in which a lot of secondary information corroborating the occurrence of the crash becomes available soon after the receipt of the PCN alert. The advantage provided by tracking the belief-variation is the most evident in such cases. Here the system always correctly predicts the type of trajectory the driver has followed – B-modulated if the alert is true, P-modulated if it is not – and thus a false positive or negative can occur only if the deviation between P and $\overline{P}$ or between B and $\overline{B}$ is larger than ϵ. Thus false positives and false negatives can be eliminated by setting the threshold ϵ arbitrarily high. Note that this is equivalent to ignoring the trajectory information, and using the secondary information alone to conclude that the alert is true.

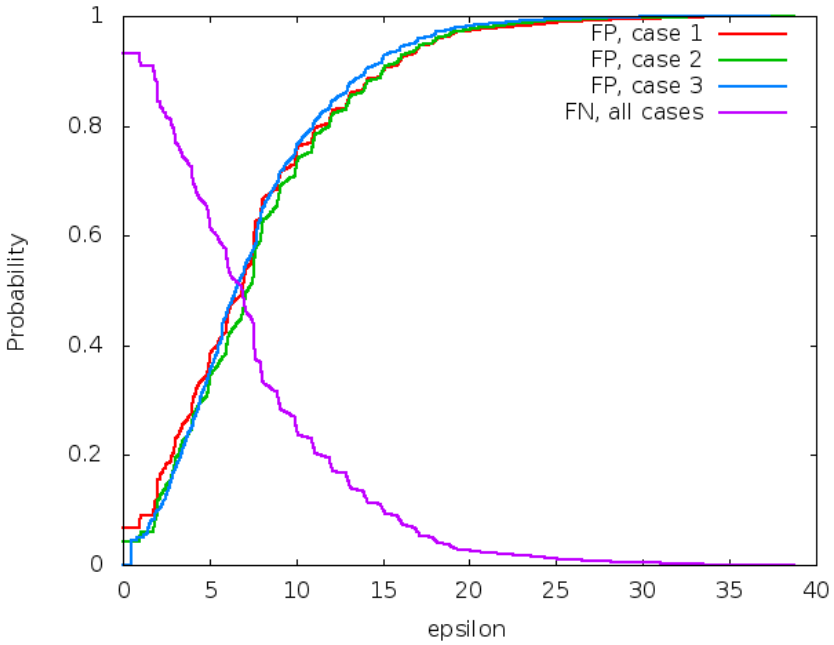

Figure 3.3: Utility variation with $\beta = (x/D)^2$

Figure 3.3 shows the results for the other subcase, when B is closer to P than to M. This represents cases in which the secondary information is not available for a significant duration of time and which, therefore, only weakly suggests the existence of the crash. By comparing the blue curve (case 3, system tracks β) and the green curve (case 2, system doesn't track β), it can be seen that for any given value of ϵ, $FP_3(\epsilon) < FP_2(\epsilon)$ (subscript denotes case number), so that $U_3(\epsilon) < U_2(\epsilon)$ (since $FN_3 = FN_2 = FN$). Thus, here too, using the secondary information improves the system performance.

Chapter 4

Using Secondary Information for Misbehavior Detection

In Chapter 3 we demonstrated how secondary information may be used to improve the accuracy of a primary information based MDS. However, the model of secondary information constructed for this purpose is more general and can be used to directly construct a generic MDS that is capable of reasoning about any form of misbehavior that can be analyzed using secondary information.

In this chapter, we first define what the final output of such an MDS could be. We then present a refined version of the secondary information model defined in Chapter 3, and show how it may be used to compute this output.

4.1 MDS Estimate

As noted in Chapter 1, we are concerned primarily with safety applications. Our objective is to design a method that enables the vehicular On Board Unit to check the veracity of any notifications of security conditions that it receives via the network. We have been referring to such methods as Misbehavior Detection Schemes (MDS's) thus far.

Now, safety conditions may be of two types:

- **Binary**: In some cases, the MDS needs to make a binary (true/false)

decision regarding the veracity of the alert. Consider, for example, the MDS developed for the Post Crash Notification in Chapter 3. This MDS simply attempts to determine if there is in fact a crashed vehicle at the location indicated in the PCN alert or not. In such cases, the MDS we wish to construct would just need to output an estimate of the likelihood of the truth or falsity of the safety condition.

- **Non-binary**: In other applications, however, the MDS would need to estimate a value in addition to checking the veracity of the alert. One example of such conditions is the Road Hazard Condition Notification, which is an advisory broadcast by a vehicle when it detects a possibly dangerous condition such as a slippery road, fog etc. Consider an RHCN sent out by a vehicle indicating road slipperiness. The notification sent by the vehicle would also contain an estimate of the coefficient of friction on the road at that point (since different vehicles can sustain different maximum speeds for the same coefficient of friction). Thus, in addition to checking if the alert is true or not, the MDS may also need to construct an estimate of the value of the coefficient of friction if the alert is in fact true.

The MDS described in this chapter outputs a description of a random variable indicating the final estimate. That is, it outputs a probability distribution describing what the precise value assigned to the safety condition could be. The range of this random variable depends on the exact safety condition under consideration. For binary conditions, the range is simply $\{\langle true\rangle, \langle false\rangle\}$. For non-binary conditions, the range would be application dependent. For the RHCN application, the range would be $\{\langle false\rangle\} \cup (\{\langle true\rangle\} \times \mathbb{R}^+)$ – that is, if the alert is true, the MDS would also need to estimate the coefficient of friction (a positive real number).

We define a function V that assigns to each time t a random variable $V(t)$ ranging over the output space. We shall use the notation $\beta(t, v)$ to denote the probability $P(V(t) = v)$. For binary conditions, we shall usually only describe $\beta(t, true)$ (note that $\beta(t, false)$ is simply $(1-\beta(t, true))$, and use the

notation $\beta(t)$ to indicate $\beta(t, \mathtt{true})$. Note that this agrees with the specialized definition of β for the PCN MDS in Chapter 3.

4.2 Basic Model

We wish to define a scheme that would allow the OBU to compute the posterior probability of the primary event being true, given the description of the secondary information (the set of alerts) received over the network. To describe the secondary information, we shall use the same basic event model as in Chapter 3. Event classes shall be represented as a set of attributes describing the information content of each event in the class, and individual events as an assignment of values to each of these attributes.

We shall again use the attributes t and (x, y) to denote the time and position at which the event originated for all event classes, and u to denote a vector of class-specific attributes for each event class. We assume that the attribute space is discrete, and not continuous. In particular, we use a discretized model of time and space. Thus, the event-space may be represented as a multidimensional grid, each cell of which may be the source of 0 or 1 event.

Note that if the grid has N cells, there are 2^N possible configurations that the OBU might observe, depending on whether each individual cell in the grid generates an event or not. Thus, it is obviously infeasible to simply precompute and store the posterior probability corresponding to a given configuration for all the possible configurations. In order to provide a reasonably efficient method for the OBU to compute the posterior probability for a given configuration (i.e. the probability of the primary event being true given the set of events it has observed), two approaches might be taken:

1. The configuration-space might be compressed, possibly by conflating similar configurations. That is, it might be possible to partition the set of all configurations into a set of classes, so that in order to determine the posterior probability for a given configuration, the OBU would first estimate which class the configuration belonged to, and

then apply a simple class-specific function based on such parameters as, for instance, the distance of the actual configuration to the mean configuration of this class, to obtain the actual probability value.

2. An explicit function describing how the posterior probability might be generated for a given configuration, i.e. for a given assignment of $0/1$ values to each of the cells in the grid, might be defined.

We shall focus on the second approach in what follows. Specifically, we shall define posterior probability values for each individual grid cell – the probability that the primary event is true given there was an alert raised at a certain cell, and the probability given no alert was raised at the cell – and then use a combination rule to compute the actual probability for the given configuration from these per-cell probabilities.

4.3 Event Model

We essentially use a refined version of the model in Chapter 3. Instead of defining the frequency function f explicitly, which would impose the requirement that the number of events occurring in a given timeslot be known a priori, we define a probability estimate of an event being generated at each cell, and combine it into the probability function p. We also partition p and define it as a combination of a set of component probabilities to simplify the task of instantiating the definition of p for a given problem instance.

4.3.1 Event Class

We now formally define event classes. An event class $\mathcal{E}$ is an ordered tuple consisting of:

$D = D_1 \times D_2 \times \cdots \times D_n$**:** The set of all possible events. The D_i are the domains of each attribute in the event class. In the function definitions below, u shall denote a variable ranging over the set of all possible events D, and u_i a variable ranging over the attribute space D_i. The

attributes t and (x, y), common to all event classes, shall usually not be listed explicitly when defining D for a particular event class.

$R_E(t, x, y, u)$**:** A function that maps to each distinct grid location $\langle t, x, y, u\rangle$ a binary random variable that indicates if an event with attribute values u is generated by a car at time t at the position (x, y). The range of each random variable is $\{0, 1\}$.

$R_{\mathrm{prim}}(t, x, y, v)$**:** A function that defines binary random variables indicating if the event corresponding to a primary alert with value v occurs at (x, y) at time t.

$CAR(t, x, y)$**:** A function that defines binary random variables indicating if there is a car at location (x, y) at time t.

$R_S(t, x, y, s)$**:** A function that defines binary random variables indicating if the sensor values of a vehicle at (x, y) at time t are s. We assume there exists a mapping $\sigma : S \rightarrow U$ from the sensor values to the event attributes.

$W(t, x, y, u)$**:** A function that assigns a numeric weight indicating relative importance to each grid location. We impose the condition that

$$\left(\sum_{t} \sum_{(x,y)} \sum_{u} W(t, x, y, u) \right) = 1$$

We also assume the primary event had value v_a and was generated by a car at location (x_a, y_a) at time t_a.

We use $GRID(t, x, y, u)$ to represent the actual configuration of the grid as observed by the OBU. Note that GRID essentially represents an observation of 0 or 1 values for each $R_E(t, x, y, u)$ in the event class.

4.3.2 Computing $\beta_{\mathcal{E}}$

Our aim is to compute an estimate of $\beta_{\mathcal{E}}(t_a, x_a, y_a, v_a)$, the probability of the alert being true given the description of the secondary information provided

by this event class. Note that by the definition of $\beta_{\mathcal{E}}$,

$$\beta_{\mathcal{E}}(t_a, x_a, y_a, v_a) = P(R_{prim}(t_a, x_a, y_a, v_a) = 1 | GRID) \tag{4.1}$$

In order to do this, we first compute for each individual secondary event $\langle t, x, y, u \rangle$ the probability $P(R_{prim}(t_a, x_a, y_a, v_a) = 1 | R_E(t, x, y, u) = 1)$, i.e. the probability of the primary alert being correct given the occurrence of this secondary event, and then combine these probabilities according to the combining rule defined below.

Using Bayes' rule,

$$\begin{aligned} & P(R_{prim}(t_a, x_a, y_a, v_a) = 1 | R_E(t, x, y, u) = 1) \\ = & \frac{P(R_{prim}(t_a, x_a, y_a, v_a) = 1) \times P(R_E(t, x, y, u) = 1 | R_{prim}(t_a, x_a, y_a, v_a) = 1)}{P(R_E(t, x, y, u) = 1)} \end{aligned} \tag{4.2}$$

Thus, the probability can be defined in terms of the a priori probabilities $P(R_{prim}(t_a, x_a, y_a, v_a) = 1)$ and $P(R_E(t, x, y, u)) = 1$, and the conditional probability $P(R_E(x, y, y) = 1 \mid R_{prim}(t_a, x_a, y_a, v_a) = 1)$. We later show how each of these probabilities can be estimated.

In order to define $\beta_{\mathcal{E}}(t_a, x_a, y_a, v_a)$, we use the weight function W to define the combination rule

$$\begin{aligned} & P(R_{prim}(t_a, x_a, y_a, v_a) = 1 | GRID) \\ = & \sum_t \sum_{(x,y)} \sum_u \{W(t, x, y, u) \times IND(t, x, y, u)\} \end{aligned} \tag{4.3}$$

where

$$\begin{aligned} & IND(t, x, y, u) \\ = & P(R_{prim}(t_a, x_a, y_a, v_a) = 1 | R_E(t, x, y, u) = GRID(t, x, y, u)) \end{aligned}$$

which, along with equation 4.1, gives us the final probability $\beta_{\mathcal{E}}(t_a, x_a, y_a, v_a)$.

The function W is an application specific weight assignment that reflects the relative importance assigned to each grid location. The highest weight

would be assigned to the cells where the occurence an event provides the strongest evidence for or against the truth of the primary alert.

In addition, note that while the conditional probability $P(R_E(x, y, y) = 1 \mid R_{prim}(t_a, x_a, y_a, v_a) = 1)$ can be estimated directly, we may also compute it using an estimate of the distribution of cars over the grid (if available) by observing that

$$\begin{aligned} & P(R_E(t, x, y, u) = 1 | R_{prim}(t_a, x_a, y_a, v_a) = 1) \\ &= P(CAR(t, x, y) = 1 | R_{prim}(t_a, x_a, y_a, v_a) = 1) \\ &\times P(R_S(t, x, y, s) = 1 | CAR(t, x, y) = 1 \wedge R_{prim}(t_a, x_a, y_a, v_a) = 1) \\ &\times P(R_E(t, x, y, u) = 1 | R_S(t, x, y, s) = 1) \end{aligned} \tag{4.4}$$

We introduce an extra level of indirection by using the sensor values since it is possible that the sensor values are proper for generating the alert, but that the alert is not generated due to other reasons (such as the absence of an OBU).

4.3.3 Computing β

In order to combine the $\beta_{\mathcal{E}}$ for all the event classes into a single estimate β of the value of the alert, we use a simple weighted combination

$$\beta(t_a, x_a, y_a, v_a) = \sum_{\mathcal{E}} w_{\mathcal{E}} \beta_{\mathcal{E}}(t_a, x_a, y_a, v_a) \tag{4.5}$$

where

$$\sum_{\mathcal{E}} w_{\mathcal{E}} = 1$$

The weight $w_{\mathcal{E}}$ assigned to a class $\mathcal{E}$ indicates the relative trust placed in that class with respect to all the other classes.

We thus have a method that allows us to compute the probability of truth or falsity of a primary alert given a probabilistic description of all the secondary information observed. However, the probabilities defined above depend on several factors like the congestion model, the mobility models of the

cars, the actual primary and secondary events considered and the nature of the correlation between them, and the chance of the safety condition occurring. Hence these probabilities may be hard to obtain analytically. In the next section, we show how this model may be developed for a specific application scenario and show how these probabilities may be estimated empirically through simulation.

4.4 Example

We consider again the PCN application from Chapter 3, and demonstrate how the required probabilities may be estimated in a system consisting of two classes of events – the class containing the PCN alert, and the class containing the SVA alerts. The PCN alert is the primary event, and the secondary information is comprised of zero or more SVA alerts raised due to the PCN alert.

4.4.1 PCN Event Class

This class consists of a single event, the initial PCN notification.

Attributes :

This event has no distinct attributes apart from t, the time at which the alert was raised, and (x, y), the location of the crash site.

Random Variables :

We shall use $\mathsf{CRASH}(t, x, y)$ to denote the random variable corresponding to the primary event ($\mathsf{R}_{\mathrm{prim}}(t, x, y, \mathtt{true})$).

4.4.2 SVA Event Class

An SVA alert may originate from a vehicle at any distance to the crash site.

Attributes :

This event class again has no distinct event attributes other than t and (x, y).

Random Variables :

We denote the random variables R_E by *SVA*. We let the range of x be $\{1, 2, \ldots, D\}$, and that of y be $\{1, 2, \ldots, y_1, y_1 + 1, y_1 + 2, \ldots, y_{l-1} + 1, \ldots, y_l\}$, l being the number of lanes on the road.

4.4.3 Estimating the Probabilities

We describe here how all the component probabilities needed to compute the final estimate $\beta(true)$ of the PCN alert being true might be estimated. The estimates can be obtained from a combination of historical data of driver behavior and experimental results obtained by simulating different crash scenarios.

$P(CRASH(t_a, x_a, y_a) = 1)$:

In general, this depends on the congestion on the road and on driving habits. This can be estimated for a given time duration and road segment based on historical data collected by the relevant authorities, such as the transportation authorities. Note that crashes are not very frequent in general, and the probability value can be taken to be very low even in a congested scenario.

$P(SVA(t, x, y) = 1 | CRASH(t_a, x_a, y_a) = 1)$:

These values may be obtained by simulating a crash at t_a at (x_a, y_a) and then recording the average number of alerts generated at each grid cell (t, x, y) per simulation run. This value is the required probability.

$P(SVA(t, x, y) = 1)$:

Note that an SVA alert might be raised even when there is no crash due to road congestion. Applying Bayes' rule, the required probability is given by

$$
\begin{aligned}
&P(SVA(t,x,y)=1)\\
&= P(SVA(t,x,y)=1|NOCRASH) \times P(NOCRASH)\\
&+ \sum_{t,x,y} P(SVA(t,x,y)=1|CRASH(t,x,y)=1) \times P(CRASH(t,x,y)=1)
\end{aligned}
$$

Thus, the probability value can be obtained by simulating the no-crash as well as various crash scenarios and observing the average number of SVA alerts at each grid cell.

$W(t,x,y)$:

The weights may be obtained by simulating the crash and no-crash scenarios and computing the average number of extra alerts generated at each cell in the crash scenario. Multiple approaches might be taken to obtain the weight values from this information. One strategy might be to assign the weights in the proportion of the alerts raised, with the highest weight assigned to the cell generating the most alerts. An alternate approach might be to fix a certain threshold based on the shape of the alert distribution across the grid, assign a weight 0 to all the cells having fewer alerts than this threshold, and distribute the total weight equally among the remaining cells.

Chapter 5

Collaborative MDS

We have so far concentrated on the problem of using secondary information to develop an MDS in which the OBU passively collects primary and secondary information and uses it to reason about possible misbehavior. However, an alternate model is possible in which vehicles share information through message passing and arrive at a collective decision regarding the possibility of misbehavior. We investigate such schemes, which we refer to as *collaborative* MDS's, in this chapter.

5.1 Introduction

The collaborative scheme we design is an adjunct to the vehicles' independent MDS processes – we assume every vehicle participating in the collaborative scheme also executes an independent MDS. The vehicles exchange the information collected through their independent MDS process in order to arrive at a corrected estimate of the possibility of misbehavior. The estimate obtained through the independent MDS is not constant, and changes with time as more information is received. However, the collaborative scheme we explain here proceeds on the assumption that all the vehicles have a constant, final, estimate of the possibility of misbehavior obtained through the independent MDS. One possible way of accounting for the variation in the value from the independent MDS, while still incorporating the benefits of a

collaborative scheme, would be to interleave periodic runs of the collaborative MDS into the execution of the independent MDS. However, we have not yet studied the performance or possible utility of such an approach.

Therefore, the scheme we design must allow a group of vehicles with fixed values obtained from a locally executed process to communicate and reach an updated estimate of the actual value. While this problem appears quite similar to the consensus problem in distributed systems, there are a number of distinguishing features that prevent us from applying solutions to the consensus problem directly to the VANET scenario. In the following section, we discuss the distributed consensus problem and its exact relation with the collaborative MDS.

5.2 Distributed Consensus

The distributed consensus problem is the problem of designing a protocol that allows a set of nodes, each holding a certain value, in a fully connected network to reach a final state in which

1. All the non-faulty nodes have the same value.
2. If all the non-faulty nodes have the same initial value, the final value that the non-faulty nodes settle on is this initial value.

Note that achieving perfect agreement may not always be required in all applications – it might suffice for the values at all the nodes to be within an ϵ-distance of each other. In such cases an algorithm that seeks to achieve full distributed consensus might prove overly expensive in terms of time and/or message complexity. Indeed, there exist systems in which a distributed consensus algorithm can never terminate but in which the ϵ-convergence criterion stated here is reachable. We therefore focus on a variant of the distributed consensus problem known as the *approximate agreement* problem[7], in which it is only required that the values at all the nodes be within a specified ϵ-distance of each other after a certain number of rounds (they need

not have precisely the same values). We may formally state the approximate agreement problem as follows:

The approximate agreement problem is the problem of designing a protocol that allows a set of nodes, each holding a certain value, in a fully connected network to reach a final state in which

1. The values at all the non-faulty nodes are within an ϵ-distance of each other.

2. The final value at each non-faulty node is within the range of values of the non-faulty nodes before the execution of the protocol.

There are several distinguishing characteristics of the VANET scenario which make our problem significantly more challenging than the traditional approximate agreement problem. Some of these characteristics are listed below:

1. The most important characteristic is that the approximate agreement problem only requires that all the nodes agree upon *some* value. However, we also require that the final value agreed on be *meaningful* in some sense. We shall later impose the requirement that the final value agreed on be reasonably close to some deterministic function of the initial values at the nodes, such as the mean.

2. The network may not be fully connected, and it is possible that not every node can communicate directly with every other node. While our scheme shall simply try to identify and proceed with a fully connected subset of the network, later work could try to adapt some of the known schemes for partially connected networks, such as those by Kieckhafer and Azadmanesh [11, 2].

3. The network is also not static since the nodes keep moving in and out of range relative to each other. Thus, the set of neighbors at each node changes with time.

4. Most solutions to the agreement problem consider what is known as the Byzantine fault model [12], in which it is assumed that faulty nodes may behave in an arbitrary manner. However, since all messages are broadcast in a VANET, it is not possible for a node to simultaneously send differing values to its neighbors. Thus, assuming a fully Byzantine fault model is not necessary. Indeed, such an assumption might prevent us from reaching optimality. However, it is possible that not all nodes might receive a broadcast message because of packet collisions, range issues etc.

In the following section we formulate the problem definition and describe the network and fault models we will be using.

5.3 Problem Definition

The basic problem is to define a protocol that transforms the **input** to the **output** in a system constrained by the network and fault models described below.

Input: *A set of nodes* $X_1, X_2, \ldots, X_n$, *each node* X_i *of which has an initial value* p_i, $\mathtt{LowerBound} \leq p_i \leq \mathtt{UpperBound}$. *Each* p_i *is obtained from an independent MDS process running at the node* X_i. *Out of the* n *nodes, at most* t *nodes can be misbehaving, and the rest are honest.*

Output: *Each honest node* X_i *settles on a final value* q_i, *such that each* q_i *is within an* ϵ*-distance of some deterministic function* f *of the initial* p*-values of all the honest nodes.*

The function f is defined to capture some intuitions about the final value that the nodes need to agree upon – for instance that the final value must be high if all the initial values are high, and that it must be low if all the initial values are low. An example of such a function would be the mean of all the initial values.

5.3.1 Network Model

We make the following assumptions about the nature of the network:

1. The network is fully connected. All messages are transmitted via broadcast.

2. The set of nodes is not static, and some nodes might exit the system after a certain number of rounds. As shall be shown later, however, our analysis shall concentrate mostly on quantifying the improvement in the nodes' values during one round. Therefore, the node losses are not particularly important, since the final multi-round analysis would simply need to combine the single-round results with a model of the node loss pattern.

3. There is a fixed probability with which a message sent to a node might be lost. This is not an unreasonable assumption since message-transfers during a round essentially happen through a set of simultaneous broadcasts over a single shared medium, and losses not caused by problems at the sender (which would affect all recipients equally) happen primarily due to collisions in the shared medium.

   Note that messages originating from honest nodes might also not reach all the other nodes. This is in contrast to the network model in the traditional approximate agreement problem.

4. The message losses are assumed to be fully independent of each other. This may not be completely reasonable, since it ignores potentially valid conditional dependencies – for instance, it might be the case that a message loss at one recipient increases the likelihood that other messages sent to that node will also be lost. However, these conditional dependencies are fairly difficult to model and understand, especially given that we work with a fairly general, high-level model of the network. We ignore these conditional dependencies for now.

5.3.2 Fault Model

The following modes of faults are possible in the system:

1. A (faulty) node might broadcast an incorrect value that is not in the range $[LowerBound, UpperBound]$ of the initial values at the honest nodes. This erroneous value may or may not be delivered to all the nodes in the network. It is not, however, possible for a node to send different values to different nodes.

2. A node might broadcast a correct value, but this value may not be delivered at all (or even any) of the nodes in the system due to network losses.

5.4 Algorithm Description

The algorithm that we shall propose shall be of the class of mean-subsequence-reduce (MSR) algorithms, the first of which was proposed by Dolev et al. [7]. An MSR algorithm is one that implements the following four basic steps in each round at each node:

1. **Collect**: Broadcast the node's current value, and collect all the values received from the other neighbors, storing them in a sorted multiset.

2. **Reduce**: Remove the highest k and lowest k values from the multiset, where k is a parameter decided by the exact choice of algorithm.

3. **Select**: Select a subsequence of the remaining values.

4. **Mean**: Take the mean of all the selected values and replace the node's current value with the newly computed mean.

The differences among algorithms in this class lie in the way each of the first three steps is implemented. Many algorithms on the MSR theme have been proposed to deal with differing fault models, network models etc. Some examples are cited below where relevant.

We next discuss the various ways the first three steps might be performed in a VANET and analyze their advantages and disadvantages. Based on this analysis, we describe the final algorithm in subsection 5.4.2

5.4.1 Issues for MSR algorithms in VANETs

We discuss separately the issues involved in implementing each of the first three steps of the algorithm.

Collect

When collecting the values from the neighbors in a VANET, missing values (lost messages) must be dealt with somehow. We split the algorithms into two categories based on the form of strategy they use for handling lost messages, and discuss them separately.

Loss Replacing Algorithms

Whenever a message from a certain node is lost, these algorithms replace the missing value with an alternative. Three strategies may be followed for choosing the replacement value:

1. The node might simply choose a default value in case of a message loss.
2. The node might choose the replacement based entirely on the current or historical values of its neighbors.
3. The node might choose its own value as the replacement.

We consider the three different strategies below[1], and their applicability to the VANET scenario.

Default Value :

This is the original approach, suggested by Dolev et al. [7] when they

[1]The explanations for the failure of these strategies were validated through a simulation-based analysis. The details of the analysis are omitted for brevity.

proposed the MSR algorithm class. It is assumed that there is some initial default value, common to all the nodes, that the nodes use whenever there is a message loss. While this approach does guarantee convergence, the problem is that it tends to skew the value that is converged to towards the default value. While this was acceptable to Dolev et al. since their focus was primarily on just guaranteeing convergence, in our case the final value converged to is also important, as stated earlier.

Neighbors' Values :

One example of the second strategy would be for the replacement value used when a message from a node is lost to be the last value ever received from that node. This does not even guarantee convergence since after a point (once a node leaves the network) all the remaining nodes remain stuck at different remembered values (difference occurring due to the possibility of message loss) for this node, and this holds back convergence.

Other examples of the second strategy might choose to employ the values of the other nodes in the system, and not just the node which originated the lost message. While any approach that uses historical information would fail to ensure convergence for reasons similar to those noted in the previous paragraph, an approach that might work is to use a deterministic function (such as the mean) of the values that *have* been received at this node as the replacement. However, since we eventually take a mean (step 4 of the MSR algorithm), these approaches turn out to be effectively equivalent to the other category of *collect* algorithms – the category of loss ignoring algorithms.

Own Value :

In this approach, suggested by Azadmanesh and Krings [4], each node uses its own value as the replacement for any missing value. While this approach does guarantee eventual convergence, the rate of convergence might be extremely slow, especially in systems where the proba-

bility of message loss is very high, since in such cases the nodes change very slowly from their initial values.

Loss Ignoring Algorithms

This alternate category was suggested by Azadmanesh and Kieckhafer [3]. Here, a node that does not receive a value from a neighbor simply ignores this neighbor, and proceed with the set of values that it has actually received in that round. This approach, while more natural than the replacement approach discussed earlier, is harder to analyze theoretically since in this case the nodes might have different-sized multisets in each round, which significantly complicates the theoretical analysis.

Simulation results also indicate that this approach performs rather well. Some results are presented in section 5.6.

We adopt the loss ignoring approach in our algorithm.

Reduce

In this step, we remove the highest and lowest k values from the multiset of values. We take k to be t, an upper bound on the number of faulty nodes (nodes which broadcast values that are outside the range $[LowerBound, UpperBound]$) in the system. This step is necessary in order to handle the cases when there are in fact t faulty nodes in the system – it is possible that all the faulty nodes broadcast very high values (in which case we would have to discard the t highest values), or that all the faulty nodes broadcast very low values (in which case we would have to discard the t lowest values).

While this step will cause some skew in the final value converged to (more if t is larger than necessary), it is unavoidable if we are to guarantee that the final value converged to lies within the range of initial values at the honest nodes.

Select

We shall only consider the all-select function, which simply selects every value in the reduced multiset, in this chapter. Later work might try to investigate the effect the choice of select function has on the overall performance of the MSR algorithm.

5.4.2 Algorithm OMSR

Based on the discussion in subsection 5.4.1, we finalize the following choices for each of the first three steps in the MSR algorithm:

- **Collect**: Ignore lost messages, and store only the values from the messages successfully delivered at this node in the multiset.
- **Reduce**: Remove the highest and lowest t values from the multiset, where t is an estimate of the number of faulty nodes.
- **Select**: Select every value in the reduced multiset.

Algorithm 1 is the final algorithm executed by each node in the VANET. We shall henceforth refer to this algorithm as the algorithm OMSR (short for Omissive MSR).

```
1: real value ← estimate from independent MDS
2: integer t ← estimate of number of faulty nodes
3: multiset U ← ∅
4: repeat
5:    U ← {value}
6:    repeat
7:       Receive v from neighbor X
8:       if no other value from X is in U then
9:          U ← U ∪ {v}
10:      end if
11:   until timeout
12:   value ← mean(reduce_t(U))
13: until number of rounds threshold
```

Algorithm 1: Algorithm OMSR

5.5 Analytical Results

We assume there are n nodes $X_1, X_2, \ldots, X_n$ with initial values $V = \{v_1, v_2, \ldots, v_n\}$. The set of values V is initially distributed with mean μ_{init} and variance σ^2_{init}, with

$$\mu_{init} = \left(\frac{1}{n}\right) S_1 \tag{5.1}$$

and

$$\begin{aligned} \sigma^2_{init} &= \frac{1}{n} S_2 - \mu^2_{init} \\ &= \left(\frac{1}{n} - \frac{1}{n^2}\right) S_2 - \left(\frac{2}{n^2}\right) C_2 \end{aligned} \tag{5.2}$$

where for all k

$$S_k = \sum_{i=1}^{n} v_i^k \tag{5.3}$$

and

$$C_k = \sum_{1 \le i_1 < i_2} \sum_{1 \le i_2 < i_3} \cdots \sum_{1 \le i_{k-1} < i_k} v_{i_1} v_{i_2} \ldots v_{i_k} \tag{5.4}$$

In this section we quantify the effect the execution of one round of this algorithm would have on the distribution of the values, and compute estimates of the final mean μ and variance σ^2.

We assume that all the nodes in the system are honest, ie that $t = 0$. Note that the only step of the algorithm that the value of t affects is the reduce step. Thus the effect of a non-zero value of t would be to trim equal sized probability masses of weight $\frac{t}{n}$ each from the higher and lower extremes of the distribution of *all* values (honest and faulty), which might affect the mean and variance if the distribution were asymmetric. However, we ignore these effects of t for now.

Therefore, the only possible faults in the system are arbitrary message losses.

We consider the best case, worst case and average case performance of the algorithm separately.

5.5.1 Best Case

If none of the messages in the system are lost, then each node ultimately considers the same multiset of values, and therefore all nodes converge to the same value (the mean) after one round of the algorithm.

When using the all-select algorithm, this is the only message loss pattern in which perfect convergence happens in one round. This is because it is necessary for perfect convergence that all nodes use the same multiset of values, and since every node X_i always has its own value v_i in its own multiset U_i, it is necessary that v_i also be present in the multisets of all the other nodes in the system. Therefore, we must have $\forall i \forall j\ v_i \in U_j$, which is only possible if none of the messages are lost.

5.5.2 Worst Case

Suppose, without loss of generality, that the values at the nodes are sorted in the order $v_1 \leq v_2 \leq \cdots \leq v_n$. Before the execution of the algorithm, the difference between the highest and lowest values in the system is $(v_n - v_1)$. We now try to estimate what the worst possible value of this difference is after the execution of one round of the algorithm.

The absolute worst case is simply when neither X_1 nor X_n receive any messages. In this case the difference continues to remain $(v_n - v_1)$. However, suppose at least k messages are delivered at each of X_1 and X_n. In this case the worst-case difference between the largest and the smallest values becomes

$$\max\left(\frac{v_{n-k+1} + v_{n-k+2} + \cdots + v_n}{k+1}, \frac{v_{n-1} + v_n}{2}\right) - \min\left(\frac{v_1 + v_2 + \cdots + v_k}{k+1}, \frac{v_1 + v_2}{2}\right)$$

Therefore, as long as at least one message is delivered at either of X_1 and X_n, the difference between the largest and smallest values in the system decreases after executing one round of the algorithm.

5.5.3 Average Case

Prior to the execution of the algorithm, each node X_i has a value v_i, and the set of all values V is described by a probability distribution with mean μ_{init} and variance σ^2_{init}, defined by equations 5.1 and 5.2 above. After executing one round of the algorithm, there are different possibilities for the final value at a given node in the system, depending on the exact message loss pattern. We may thus use the probability distribution over the set of all message loss patterns (which can be obtained from the network model) to define a probability distribution describing the value at a given node after the execution of this algorithm, and combine these together to also define the probability distribution of the final set of values V' at the nodes. We now compute the mean and variance of each of these distributions – that is, the distributions of the value at each X_i, as well as the distribution of the set of values V'.

We use the following notation:

p **and** q :

p is the probability of a given message being successfully delivered, and $q = 1 - p$ is the probability of the message being lost.

$E(X_i)$ **and** $E(X_i^2)$:

Denoting respectively the expected value of v_i' and the expected value of $(v_i')^2$, where v_i' is the value at node X_i after executing one round of the algorithm.

μ_i **and** σ_i^2 :

Denoting respectively the mean and variance of the distribution of v_i'. We have

$$\mu_i = E(X_i) \tag{5.5}$$

and

$$\sigma_i^2 = E(X_i^2) - \mu_i^2 \tag{5.6}$$

$E(X)$ **and** $E(X^2)$:

Denoting respectively the expected value of v' and $(v')^2$, where v' is a random element of the set V'. We have

$$E(X) = \frac{\sum_{i=1}^{n} E(X_i)}{n} \tag{5.7}$$

and

$$E(X^2) = \frac{\sum_{i=1}^{n} E(X_i^2)}{n} \tag{5.8}$$

μ **and** σ^2 :

Denoting respectively the mean and variance of the distribution of v'. We have

$$\mu = E(X) \tag{5.9}$$

and

$$\sigma^2 = E(X^2) - \mu^2 \tag{5.10}$$

We now compute $E(X_i)$ and $E(X_i^2)$. As the equations above demonstrate, the values of all the other parameters can be obtained from the values of these two.

Now, in one round of the algorithm, the node X_i is sent $n-1$ values, any or all of which might be lost over the network. There are thus 2^{n-1} possible message loss patterns. Of these, there are $\binom{n-1}{k}$ patterns in which k messages are delivered at the node, and the probability of each of these patterns occuring is $p^k q^{n-1-k}$. The final value v_i' for each of these patterns is of the form

$$\frac{v_i + v_{i_1} + v_{i_2} + \cdots + v_{i_k}}{k+1}$$

since the value v_i of the node itself is never lost.

$E(X_i)$:

For each $j \neq i$, the value v_j is successfully delivered at node X_i in $\binom{n-2}{k-1}$ of the $\binom{n-1}{k}$ patterns in which there are k deliveries at the node. Thus, the coefficient of v_j in $E(X_i)$ is[2]

$$\begin{aligned} c_j^i &= \sum_{k=1}^{n-1} \binom{n-2}{k-1} \frac{p^k q^{n-1-k}}{k+1} \\ &= \frac{1}{n-1} - \frac{1}{n(n-1)} \frac{1-q^n}{1-q} \end{aligned}$$

The value v_i is present in all the patterns, and thus its coefficient is

$$\begin{aligned} c_i^i &= \sum_{k=0}^{n-1} \binom{n-1}{k} \frac{p^k q^{n-1-k}}{k+1} \\ &= \frac{1}{n} \frac{1-q^n}{1-q} \end{aligned}$$

Thus,

$$\begin{aligned} E(X_i) &= c_i^i v_i + \sum_{j \neq i} c_j^i v_j \\ &= \left(\frac{1}{n} \frac{1-q^n}{1-q} \right) v_i + \sum_{j \neq i} \left(\frac{1}{n-1} - \frac{1}{n(n-1)} \frac{1-q^n}{1-q} \right) v_j \end{aligned} \tag{5.11}$$

μ_i:

By equation 5.5, μ_i is simply $E(X_i)$, and is thus given by equation 5.11. Note that for large values of n, the mean is approximately

$$\mu_i \approx \left(\frac{1}{np} \right) v_i + \sum_{j \neq i} \frac{1}{n-1} \left(1 - \frac{1}{np} \right) v_j \tag{5.12}$$

That is, the node X_i assigns a weight $\frac{1}{np}$ to its own value and splits the remaining weight $1 - \frac{1}{np}$ equally among the values of all the other nodes.

[2]The derivations of the final formulae for all the coefficients are based on straightforward combinatorial arguments. The details are omitted for the sake of brevity.

μ:

Combining equation 5.9 with equation 5.11, we see that the mean is given by

$$\begin{aligned}\mu &= \frac{1}{n}\sum_{i=1}^{n}\left[\left(\frac{1}{n}\frac{1-q^n}{1-q}\right)+(n-1)\left(\frac{1}{n-1}-\frac{1}{n(n-1)}\frac{1-q^n}{1-q}\right)\right]v_i \\ &= \frac{1}{n}\sum_{i=1}^{n} v_i\end{aligned} \tag{5.13}$$

which, along with equation 5.1 shows that $\mu = \mu_{init}$.

Therefore, the algorithm preserves the mean.

$E(X_i^2)$:

Let j and k be such that no two of i, j and k are equal. We denote by d^i_{rs} the coefficient of $v_r v_s$ in the expression for $E(X_i^2)$.

Now, any term in $E(X_i^2)$ corresponding to a loss pattern in which precisely r messages are delivered at the node X_i is of the form

$$\left(\frac{v_i + v_{i_1} + v_{i_2} + \cdots + v_{i_r}}{r+1}\right)^2$$

Observe that v_j^2 appears in this expression iff $v_i v_j$ appears in this expression. Further, the coefficient of $v_i v_j$ is twice the coefficient of v_j^2. Summing up over all the loss patterns, we get $d^i_{ij} = 2d^i_{jj}$.

Now, we have

$$\begin{aligned}\frac{d^i_{ij}}{2} &= d^i_{jj} \\ &= \sum_{r=1}^{n-1}\binom{n-2}{r-1}\frac{p^r q^{n-1-r}}{(r+1)^2} \\ &= \frac{1}{pn(n-1)}\sum_{r=1}^{n}\binom{n}{r}\left(1-\frac{1}{r}\right)p^r q^{n-r}\end{aligned}$$

Let

$$R_{-1} = \sum_{r=1}^{n} \binom{n}{r} \frac{1}{r}$$

Then

$$\begin{aligned} \frac{d^i_{ij}}{2} &= d^i_{jj} \\ &= \frac{1}{n(n-1)} \frac{1-q^n}{1-q} - \frac{1}{n(n-1)} \frac{R_{-1}}{1-q} \end{aligned}$$

Also

$$\begin{aligned} \frac{d^i_{jk}}{2} =& \sum_{r=2}^{n-1} \binom{n-3}{r-2} \frac{p^r q^{n-1-r}}{(r+1)^2} \\ =& \frac{2}{n(n-1)(n-2)} \frac{R_{-1}}{1-q} + \frac{1}{(n-1)(n-2)} \\ & - \frac{3}{n(n-1)(n-2)} \frac{1-q^n}{1-q} \end{aligned}$$

and

$$\begin{aligned} d^i_{ii} &= \sum_{r=0}^{n-1} \binom{n-1}{r} \frac{p^r q^{n-1-r}}{(r+1)^2} \\ &= \frac{1}{n} \frac{R_{-1}}{1-q} \end{aligned}$$

This set of equations completely defines the value of d^i_{rs} for any r and s. Thus we may obtain $E(X_i^2)$ by

$$E(X_i^2) = \sum_{j=1}^{n} d^i_{jj} v_j^2 + \sum_{j=1}^{n} \sum_{k=1}^{j-1} d^i_{jk} v_j v_k \tag{5.14}$$

$E(X^2)$:

Let d_ij denote the coefficient of $v_i v_j$ in the expression for $E(X)$.

Let i, j and k be such that no two of them are equal. We have

$$\begin{aligned} d_{ii} &= \frac{d^i_{ii} + (n-1)d^i_{jj}}{n} \\ &= \frac{1}{n^2 p} \end{aligned}$$

and

$$\begin{aligned} \frac{d_{ij}}{2} &= \frac{2\frac{d^i_{ij}}{2} + (n-2)\frac{d^k_{ij}}{2}}{n} \\ &= \frac{1}{n(n-1)} - \frac{1}{n^2(n-1)}\frac{1-q^n}{1-q} \end{aligned}$$

Using the notation of equations 5.3 and 5.4, we have

$$E(X^2) = \frac{1}{n^2 p}S_2 + 2\left(\frac{1}{n(n-1)} - \frac{1}{n^2(n-1)}\frac{1-q^n}{1-q}\right)C_2 \tag{5.15}$$

σ^2:

By equations 5.8 and 5.15, we have

$$\begin{aligned} \sigma^2 &= \frac{1}{np}S_2 + 2\left(\frac{1}{n-1} - \frac{1}{n(n-1)}\frac{1-q^n}{1-q}\right)C_2 - \mu^2 \\ &= \frac{1}{n^2}\left(\frac{1}{p} - 1\right)S_2 \\ &\quad + 2\left(\frac{1}{n(n-1)} - \frac{1-q^n}{n^2(n-1)p} - \frac{1}{n^2}\right)C_2 \end{aligned} \tag{5.16}$$

Therefore, we have

$$\sigma^2 \approx \frac{1}{n^2}\left(\frac{1}{p} - 1\right)S_2 - \frac{2}{n^3 p}C_2 \tag{5.17}$$

Compare this with σ^2_{init}, which, by equation 5.2 is given by

$$\sigma^2_{init} = \left(\frac{1}{n} - \frac{1}{n^2}\right)S_2 - \frac{2}{n^2}C_2$$

Therefore,

$$\sigma^2 \approx \frac{\sigma^2_{init}}{np} - \frac{1}{n^2}\left(1 - \frac{1}{np}\right) S_2$$

That is, the variance is reduced by a factor of roughly np. Note that $np > 1$ iff $p > \frac{1}{n}$, i.e. iff p is high enough that at least one message is expected to be delivered at each node in each round.

To summarize some of the more important results from the average case analysis:

- When computing the new value at the end of one round, each node is expected to assign a weight of $\frac{1}{np}$ to its own value, and to split the remaining weight across the values at all the other nodes. That is, it assigns a higher weight to the values at the other nodes if the message delivery probability is higher.
- After executing one round of the algorithm, the mean of the distribution of values across all the nodes is preserved, but the variance is reduced (the values move closer together) by a factor of around np. Again, the higher the message delivery probability, the sharper the reduction in the variance.

5.6 Experimental Results

Sample results from the execution of the algorithm are shown in fig 5.6. These results were taken in a 20 node system, with the initial values at the nodes uniformly distributed across the range $[0.2, 0.8]$, in which the probability of any given node exiting the system after a round was 0.1. There are 4 curves, each drawn for a different message loss probability (the value of the message loss probability is indicated in the legend). The points marked on the curve indicate the mean, and the upper and lower bars at each round number indicate the highest and lowest values in the system after the given number of rounds. As can be seen from the results, the range (max val - min

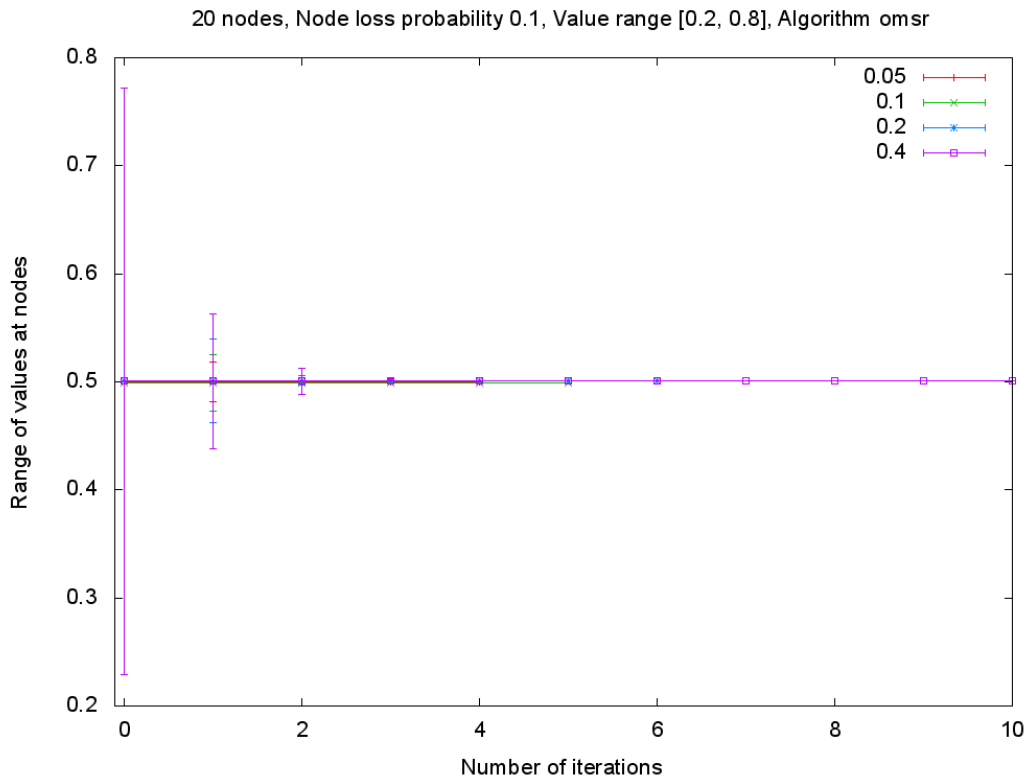

Figure 5.1: OMSR Results

val) decreases quite rapidly after the first round, even when the message loss probability is as high as 0.4.

Note also that as expected, the smaller the message loss probability, the smaller the range after the execution of a round.

Chapter 6

Conclusion

This thesis has demonstrated how secondary information and information from collaboration can be used in the design of a Misbehavior Detection Scheme (MDS). The preceding three chapters have shown how these two forms of information may be used to both augment an existing primary information based MDS, as well as to directly construct an MDS utilizing such information. The contributions of this thesis have been threefold:

1. We have demonstrated how secondary information may be used to augment the primary information in an MDS based on primary information, thus improving its performance
2. We have shown how to construct an MDS that utilizes secondary information alone, and
3. We have shown how information from collaboration might also be utilized to construct an MDS.

6.1 Future Work

Both the secondary information based MDS and the MDS based on information from collaboration might benefit from further research.

Future work on the secondary information based MDS might proceed along the following lines:

1. Although some possible approaches have been suggested, the exact method of assigning values to the weight function w has been left unspecified. Future work might investigate the performance of different approaches to defining w.

2. As was noted in section 4.2, two approaches might be taken to designing a function that returns the estimate of the final probability value for a given grid configuration – the function might be described over the space of grid configurations directly, or alternately the function might choose to assign individual probability values to each grid cell and then use the description of the grid configuration to combine them into a single posterior probability estimate. While we have chosen to take the second approach, future research might try to examine if the first approach is feasible.

Further research into the MDS based on information from collaboration might consider the following lines of enquiry:

1. While it has been noted that the presence of malicious nodes ($t > 0$) in the system might cause a skew in the final values at the nodes, the exact nature and extent of this skew needs to be investigated.

2. We have performed an extensive analysis of the possible choices for the *collect* step, and have noted that the *reduce* step in its current form is both sufficient and unavoidable in order to handle a given number of malicious nodes. However, the effect the choice of select function has on the performance of the algorithm has not been investigated. Future work might investigate alternate choices of select functions.

3. The relation between the secondary information based MDS and the MDS using information from collaboration needs to be investigated. As was noted earlier, in constructing this collaborative MDS, we have proceeded on the assumption that the secondary information based MDS's at all the nodes have reached a stable state, so that the values at all these nodes may be taken to be constant. This may not always be

a reasonable assumption, and future research might seek to construct an MDS that relaxes this assumption.

Bibliography

[1] *IEEE Trial-Use Standard for Wireless Access in Vehicular Environments - Security Services for Applications and Management Messages, IEEE Std 1609.2-2006*. 2006.

[2] A. H. Azadmanesh, A. W. Krings, and B. Ghahramani. Global convergence in partially fully connected networks with limited relays. *Journal of Information Technology and Decision Making*, 2(2):265–285, June 2003.

[3] M. H. Azadmanesh and R. M. Kieckhafer. Exploiting omissive faults in synchronous approximate agreement. *IEEE Transactions on Computers*, 49(10), October 2000.

[4] M. H. Azadmanesh and A. W. Krings. Egocentric voting algorithms. *IEEE Transactions on Reliability*, 46(4), December 1997.

[5] F. Bai, H. Krishnan, V. Sadekar, G. Holland, and T. ElBatt. Towards characterizing and classifying communication-based automotive applications from a wireless networking perspective. In *1st IEEE Workshop on Automotive Networking and Applications (AutoNet 2006)*, 2006.

[6] F. Bai, H. Krishnan, V. Sadekar, G. Holland, and T. ElBatt. Towards characterizing and classifying communication-based automotive applications from a wireless networking perspective. In *1st IEEE Workshop on Automotive Networking and Applications (AutoNet 2006)*, 2006.

[7] D. Dolev, N. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *Journal of the ACM*, 33(3):499–516, July 1986.

[8] M. Gerlach, A. Festag, T. Leinmuller, G. Goldacker, and C. Harsch. Security Architecture for Vehicular Communication. In *5th International Workshop On Intelligent Transportation*, Mar. 2007.

[9] M. Ghosh, A. Verghese, A. Kherani, and A. Gupta. Distributed misbehavior detection in VANETs. In *IEEE Wireless Communication and Networking Conference*, April 2009.

[10] P. Golle, D. Greene, and J. Staddon. Detecting and correcting malicious data in vanets. In *VANET '04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, pages 29–37, New York, NY, USA, 2004. ACM.

[11] R. M. Kieckhafer and M. H. Azadmanesh. Low cost approximate agreement in partially connected networks. *Journal of Computing and Information*, 3(1):53–85, 1993.

[12] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[13] T. Leinmuller, E. Schoch, and F. Kargl. Position verification approaches for vehicular ad hoc networks. *IEEE Network*, 2006.

[14] P. Papadimitratos, V. Gligor, and J.-P. Hubaux. Securing Vehicular Communications - Assumptions, Requirements, and Principles. In *Workshop on Embedded Security in Cars (ESCAR) 2006*, 2006.

[15] M. Raya, P. Papadimitratos, and J.-P. Hubaux. Securing Vehicular Communications. *IEEE Wireless Communications Magazine, Special Issue on Inter-Vehicular Communications*, 13(5):8–15, 2006.

[16] M. Torrent-Moreno, M. Killat, and H. Hartenstein. The challenges of robust inter-vehicle communications. In *Vehicular Technology Conference, 2005. VTC-2005-Fall. 2005 IEEE 62nd*, volume 1, pages 319–323, 2005.

[17] B. Xiao, B. Yu, and C. Gao. Detection and Localazation of Sybil Nodes in VANETs. In *Workshop on Dependability Issues in Wireless Ad Hoc Networks and Sensor Networks (DIWANS)*, Sept. 2006.