Monitoring Neighboring Vehicles for Safety via V2V Communication*

Bo Xu Ouri Wolfson
Department of Computer Science
University of Illinois at Chicago
{boxu,wolfson}@cs.uic.edu

Hyung Ju Cho
Daum Communications
Jeju, South Korea
hjcho@daumcorp.com

Abstract— **A driver should constantly keep an eye on vehicles closest to her/his location in order to avoid collisions. Unfortunately, the driver often does not see closest vehicles due to obstacles (other vehicles, trees, buildings, etc). In this paper, we introduce a novel type of query called a continuous range *k*-nearest neighbor (CRNN) query in vehicular ad-hoc networks, and propose a scheme for query processing. Our scheme aims at optimizing the usage of the wireless network bandwidth, computational cost, and local storage while preserving information on the continuous movement of vehicles within the broadcast range of a vehicle. Both simulation and road test results confirm the effectiveness and superiority of our scheme over an existing method.**

Keywords-vehicular ad-hoc network, vehicle safety, collision avoidance, location awareness, nearest neighbor queries

I. INTRODUCTION

Vehicle-to-vehicle (V2V) communication technologies are rapidly evolving, and this evolution provides opportunities to utilize these technologies in support of advanced vehicle safety applications. For example, the Vehicle Safety Communications (VSC) project initiated by the U.S. Department of Transportation identifies 34 communication-enabled vehicle safety applications [4]. Many of these applications require that a vehicle be continuously aware of the location, heading, and speed of neighboring vehicles via V2V communication. Three of them, which are ranked by VSC as the highest potential benefit applications, are listed in Table 1. From this background it can be seen that monitoring neighboring vehicles plays a critical role in V2V based vehicle safety applications.

In this paper, we introduce a novel type of query, called a *continuous range k-nearest neighbor* (CRNN) query, as a common tool for monitoring neighboring vehicles on roads. The CRNN query issued by a vehicle continuously monitors the locations of the *k* nearest vehicles that are within a range from the vehicle's location. The locations of these vehicles are plotted on the car navigation system display, and enable the driver to see the vehicles that are blocked by a track, a turn, a blind spot, etc. So the CRNN query is a combination of the classical *k*NN query and range query. The reason for limiting the range is that, obviously, the driver is not interested in vehicles that are too far away. The reason to limit the number of nearest neighbors to *k* is to avoid information overload. If there are too many vehicles within the range of interest, then the screen will become cluttered, making the information useless.

TABLE I. HIGHEST POTENTIAL BENEFIT SAFETY APPLICATIONS ENABLED BY V2V COMMUNICATION [4]

Cooperative Collision Warning	This application collects surrounding vehicle locations and dynamics and warns the driver when a collision is likely.
Lane Change Warning	This application warns the driver if an intended lane change may cause a crash with a nearby vehicle.
Pre-Crash Sensing	This application prepares the driver for imminent, unavoidable collisions.

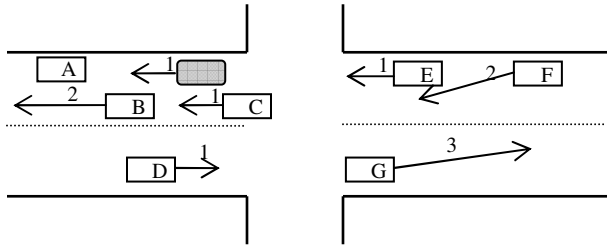

Figure 1. Example of CRNN query

We present the following example of the CRNN query with real-life semantic in Figure 1. A, B, C, D, E, F, and G are vehicles. The arrows and numbers on the vehicles indicate their headings and speeds. Note that the location and velocity of a vehicle can be continuously updated by an on-board GPS receiver. For simplicity, the current time is supposed to be 0. Consider a CRNN query that asks for every point in time the two closest vehicles from our car (shaded rounded rectangle). The query result is a sequence of answer-pairs <[0,1),{B,C}>, <[1,2),{A,C}>. It means that B and C are the two NNs of our vehicle for the time interval [0,1) and B is replaced by A when the time proceeds to 1 since A is closer to our vehicle than B at time 1.

Processing CRNN queries is a challenging problem since vehicles move continuously whereas it is impossible to track their locations continuously due to the limitation of communication bandwidth and processing capability. In this

* Research supported by NSF DGE-0549489 and IIS-0957394.

paper we adopt the following paradigm to address this problem. We let each vehicle periodically broadcast its motion information including location and velocity. Based on the motion information received from other vehicles, a vehicle predicts the future motion of these vehicles and returns the result for the CRNN query accordingly. When an update of motion information is received, the vehicle modifies the prediction of future motion and returns the new result for the CRNN query. This is a standard approach that has been used for tracking moving objects in a centralized database system (see e.g., [1]). In this paper we adapt this approach to a V2V environment.

Now there are two issues that need to be addressed in the above paradigm. First, how frequently should each vehicle update its motion information? Observe that the more frequently each vehicle updates its mobility information, the more accurately it can be monitored by the other vehicles, but on the other hand the higher communication overhead. We postulate that the update frequency should be in reverse proportion to the traffic density. The reason is that when the traffic is sparse, a larger location uncertainty is tolerable. We develop a method to compute the update frequency based on this intuition. We compare our method with the method described in the VSC project. Road tests and simulations show that our method reduces the communication overhead by 70%.

Another issue is how to efficiently process the CRNN query given the motion information of other vehicles. A naïve solution is so-called *off-line processing*, which computes a priori all the answer pairs. Off-line processing is inefficient in our environment because motion updates may come in at any time. When a motion update comes in, the answer pairs have to be re-computed and thus the a priori computation is wasted. A more efficient solution is *on-line processing*, which computes and returns one answer pair at a time. When the answer pair become invalid, the next answer pair is computed and returned. In on-line processing, when a motion update comes in, only the computation for the current answer pair is wasted. We quantify the advantage of on-line processing and show that the computational cost of processing a motion update is $O(\log^2 n)$ time in the on-line case whereas it is $O(n \log n)$ time in the off-line case.

In summary, this paper makes the following contributions:

1. We store, broadcast, and manage motion information such as location and velocity of vehicles instead of transient current location in order to overcome limited wireless bandwidth, small storage, and update overhead,
2. We introduce a processing scheme for CRNN queries and quantify its advantage over a naïve solution.
3. We devise formulae for the dynamic adjustment of two parameters of the CRNN query, i.e., the number of NNs and the radius of the query region.
4. We propose a method to minimize the total amount of disseminated reports while preserving information on continuous movement of a vehicle.
5. Simulation and road test results are provided to confirm the validity and effectiveness of our proposed method over an existing method.

The rest of this paper is structured as follows: Section 2 describes the model. Section 3 presents our motion-reports dissemination solution. Section 4 presents the processing of CRNN queries. Section 5 discusses the applicability of the proposed methods to vehicle safety applications. Section 6 concludes the paper and discusses future work.

II. THE MODEL

A *vehicular ad-hoc network* consists of a set of vehicles $V=\{V_1,V_2,\ldots,V_n\}$ capable of short range wireless communication where n is the number of vehicles participating in the network and its value may vary over time. The wireless capability is associated with a broadcast range which is the maximum physical distance between communicating vehicles. Vehicles within the broadcast range are called *neighbors*. Note that the point-to-point channel is not used.

A *motion-report*, or a *report* for short, is a message broadcasted by a vehicle which includes its current location and velocity information. Each vehicle V_i has a local reports database RDB_i which stores the motion-reports that V_i has received from neighbors. The size limit of RDB_i is S_i bytes. When a new report is received by V_i, if space is sufficient, the report is stored to RDB_i. Otherwise (i.e., if space is insufficient), an existing report is eliminated from the database to make space for the new report. Which report to eliminate depends on the storage management scheme.

We assume that at any point in time, every vehicle knows its neighbors (i.e., the vehicles within its broadcast range) by using a *neighbor-discovery protocol* where each vehicle disseminates a very short message including its vehicle id automatically at a fixed interval (e.g., 1 second). An *encounter* is the event in which vehicle V_i detects a new neighbor. While the neighbor stays within the broadcast range of V_i, it will not encounter V_i again, but it may do so after the neighbor disconnects.

Finally we introduce Greenshield's model which is widely used in traffic flow theory. Greenshield developed a model of uninterrupted traffic flow that predicts and explains the trends that are observed in real traffic flows [2]. While this model is not perfect, it is fairly accurate and relatively simple. According to the model, the relationship between the traffic speed and density is given by the following formula.

$$vel = vel_{ff} - \left(\frac{vel_{ff}}{den_{jam}} \right) \cdot den \qquad (1)$$

where vel is the traffic speed, den the density of vehicles on the road, vel_{ff} the free flow speed (i.e., the speed when there is no congestion), and den_{jam} the traffic jam density (i.e., how many vehicles fit per unit-length, vehicles/kilometer). vel_{ff} and den_{jam} are determined a-priori and are specific to each road.

III. DISSEMINATION OF MOTION-REPORTS

In this section we first introduce our scheme for disseminating motion-reports. Then we evaluate it by road tests and simulations.

A. *Scheme for Motion-reports Dissemination*

A report r consists of six attributes as follows: r = <vid, loc_x, loc_y, vel_x, vel_y,t>. The first attribute is a vehicle identifier which is introduced to identify the report issuer. The remaining five attributes are the dynamic attributes. t is the time when these dynamic attributes are measured. loc_x and loc_y are the x and y coordinates, respectively, of a vehicle at time t. vel_x and vel_y are x and y velocities, respectively, at time t. Note that a report can be identified with the first attribute vid and the last attribute t since a vehicle has unique location and velocity at a time. The dynamic attributes are updated by the GPS receiver periodically (e.g., every one second). However, a new report is created and disseminated only when a vehicle deviates from its expected location.

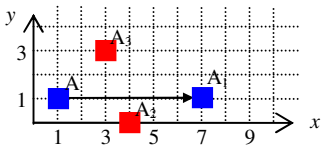

Figure 2. An example of report dissemination

Suppose that dynamic attributes of vehicle A at time t_0 are <1,1,6,0,t_0>. Then, when the time is (t_0+1), two cases may happen. In the first case, vehicle A has reached its expected location (i.e., loc_x=1+6, loc_y=1+0 at (t_0+1)) like A_1 as seen in Figure 2. In the second case, A has deviated from its expected location like A_2 or A_3 in Figure 2. When vehicle A reaches A_2 or A_3, vehicle A creates a new report and broadcasts it.

Additionally, when a new vehicle B is encountered, vehicle A creates and disseminates a report independent of the deviation of its expected location. This is because the new vehicle B has no information on the movement of vehicle A. For the same reason, the new vehicle B also disseminates a new report. In summary, a new report is generated and disseminated when either of the following two events occurs. (i) The deviation from the expected location exceeds a threshold d_{TH} (e.g., 3m) as seen in Figure 3. (ii) A new vehicle appears within the broadcast range.

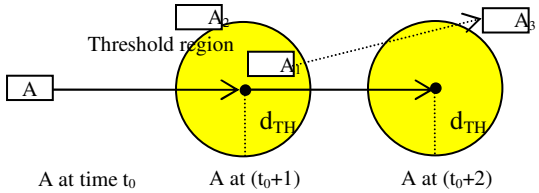

Figure 3. Threshold region with radius of d_{TH}

Figure 3 shows an example where a report is generated when vehicle A deviates from an expected location by more than d_{TH}. The circle area centered at the expected location and with radius d_{TH} is referred to as the *threshold region*. In this figure, suppose that the current time is t_0 and the center points of two big circles point out the expected locations of vehicle A at times (t_0+1) and (t_0+2), respectively. At time (t_0+1), when vehicle A reaches A_1, it does not generate a report since it is placed inside the threshold region at (t_0+1). However, when vehicle A reaches A_2 at (t_0+1), a report is generated. Similarly, at (t_0+2), when vehicle A moves from A_1 to A_3, a report is created at (t_0+2). A vehicle may be located at any point inside the threshold region if it does not generate a report.

As the traffic speed increases (i.e., the density of vehicles decreases), the uncertainty cost tends to decrease because the distance between vehicles increases. Thus, when the traffic speed is high, the update cost should be considered more important than the uncertainty cost. Conversely, the distance between vehicles decreases with decreasing traffic speed (i.e., increasing density of vehicles) so that the uncertainty cost should be deliberated more seriously in order to maintain exact information on nearby vehicles. Additionally, the performance of wireless technology like Wi-Fi decreases roughly quadratically as the range increases at constant radiation levels. This means that the communication cost which dominates the update cost increases with the distance between vehicles. Considering the traffic speed vel which affects the two costs, we propose the following formula which decides the value of d_{TH} at run-time.

$$d_{TH} = d_{\max\_th} \cdot \left(\frac{vel}{vel_{ff}} \right) \quad (2)$$

where $d_{\max\_th}$ is the upper bound value of d_{TH} and is provided by the system. For example, given $v_{ff} = 150km/h$ and $d_{\max\_th} = 10m$, when vel is 120km/h, 80km/h, and 40km/h, d_{TH} is 8m, 5.3m, and 2.7m, respectively.

Our reports dissemination method has the following two properties: (i) It enables all vehicles within the broadcast range of a vehicle to track the continuous movement of the vehicle. (ii) It consumes the minimum number of broadcasted reports required to achieve the first property. On the other hand, if a vehicle disseminates a report whenever the dynamic attributes are updated, then all the vehicles within the range can see continuous movement of the vehicle. However, this naïve approach is not acceptable since the wireless bandwidth is limited and its usage should be carefully designed. In summary, in our approach, the transmission of reports is triggered by events while the existing method sends reports automatically at regular intervals.

B. *Performance Evaluation*

We compared our scheme with an existing method which is taken from the VSC project [4]. In this method, each vehicle broadcasts its motion-report every one second and we call this the *VSC method*. Our method uses the dynamic attributes and broadcasts reports only when a vehicle deviates from its expected location or encounters a new vehicle. We call it the *dynamic method*. In most cases, the computational cost is trivial compared to the data dissemination cost. Therefore, the simulation results focus mainly on the number of reports broadcasted among vehicles.

Both road tests and simulations have been conducted. The road tests used three vehicles and PDAs (personal digital assistant) equipped with Wi-Fi (802.11b/g) and GPS receivers. The vehicles traveled along downtown roads in Chicago for the evaluation of each method which lasted for about 15 minutes. The size of each report is 24 bytes. We counted the number of reports received by each vehicle and used it as the performance measure.

The simulations considered 8000 vehicles in a 3km×3km area and tested under a large variety of environmental parameters including the traffic speed, the broadcast range, and the frequency with which a vehicle changes its velocity. The conclusions drawn from the simulations are consistent with those from the road test results. Due to space limitations the simulation results are omitted.

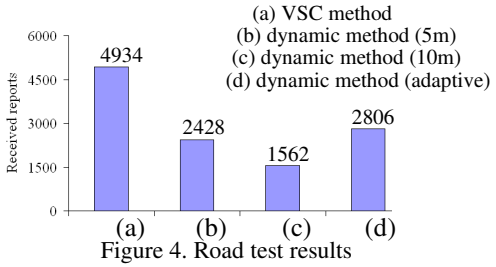

Figure 4. Road test results

Figure 4 shows the road test results. In this figure, the numbers in parentheses indicate the values of d_{TH} used by the dynamic method, and the numbers on top of columns refer to the total numbers of received reports. The average, maximum, and minimum speeds of the vehicles are 16km/h, 64km/h, and 0km/h, respectively. For the dynamic method, the value of d_{TH} is determined by Equation (2) introduced in section III.A. Notice that the dynamic method (10m) outperforms the VSC method by 70%.

The average value of d_{TH} for the dynamic method (adaptive) is 3.7 meters. Thus this method suffers from high update cost compared to the other dynamic methods. The dynamic method (10m) and VSC method achieve the best and worst performances, respectively. This is expected because the former has a larger value (i.e., 10m) of d_{TH} than the others and the latter requires vehicles to send reports every second. Since there is a trade-off between the costs of uncertainty and update in terms of d_{TH}, it can be said that even if the dynamic method (10m) outperforms the dynamic method (adaptive) in terms of the update cost, the former may suffer from the uncertainty cost.

IV. CRNN QUERIES

A. *Processing CRNN Queries*

We consider query processing at a vehicle V_q. V_q has an local reports database RDB_q that stores the motion information of each other vehicle V_1, V_2,…, V_n. A (*k*,*r*)-*CRNN query*, or a *(k,r)-query* for short, requests for every point in time the *k* nearest vehicles that are within a Euclidean distance *r* from the vehicle's location. If there are more than *k* vehicles within range *r*, then only the *k* nearest vehicles are returned. If there are fewer than *k* vehicles within range *r*, then all the vehicles within *r* are returned. The query processing assumes that every vehicle moves in a linear manner until updated. That is, each vehicle moves along a straight line with a constant speed. Under this assumption, the square of the distance d_i between V_q and another vehicle V_i is a quadratic function of time *t*:

$$d_i^2(t) = at^2 + bt + c \quad (3)$$

where *a*, *b*, and *c* are parameters dependent on the velocities and initial locations of V_q and V_i. The coefficient *a* is non-negative and thus the $d_i^2(t)$ function is convex. The $d_i^2(t)$ function is a parabola in the Time-SquareDistance space [3] as illustrated in Figure 5. For convenience of presentation, in the rest of this paper we use vehicle V_i and its square-distance function $d_i^2(t)$ interchangeably when there is no confusion.

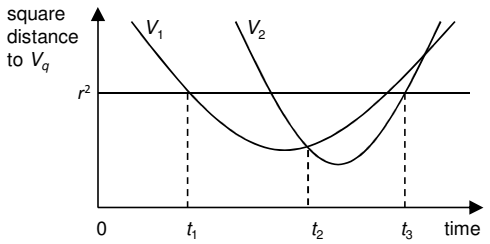

Figure 5. Example of answer-pairs in the Time-SquareDistance space

The CRNN query is issued at time 0. At that time V_q has a set of $\leq k$ nearest neighbors within range *r*, but as time progresses, the *k*NN set may change. A time interval during which the answer set remains unchanged is referred to as an *answer interval*. An *answer-pair* is a pair consisting of an answer interval and its associated answer set. As an example, in Figure 5, the answer-pairs for a (1,*r*)-query are: $<[0,t_1), \phi>$, $<[t_1,t_2), \{V_1\}>$, $<[t_2,t_3), \{V_2\}>$, $<[t_3, \infty), \phi>$.

Now we discuss the processing a CRNN query. A straightforward solution is *off-line processing*, which computes and returns a priori all the answer-pairs. When a motion update is received, the answer-pairs are re-computed and the new answer-pairs are returned. This solution is inefficient because in our environment motion updates may come in at any time. When a motion update comes in, the answer-pairs have to be re-computed and the a priori computation is wasted. A more efficient solution is *on-line processing*, which computes and returns one answer-pair each time. When the time proceeds to the end of the current answer interval, the next answer-pair is computed and returned. In this paper we adopt on-line processing.

We start with a simple case in which k=1 and r=∞, i.e., the (1,∞)-query which is the traditional nearest neighbor query. Then we extend to general *k* and *r*. Observe that for the (1,∞) query, our problem is nothing but dynamically maintaining the lower envelope of a set of parabolas in the Time-SquareDistance space. In computation geometry this problem is known as the *kinetic minimum maintenance* problem. The authors of [5] introduce a solution to this problem, which is called a *kinetic tournament*. The idea is to use a simple divide-and-conquer strategy. The algorithm partitions the vehicles into two approximately equal-sized groups (arbitrarily), and recursively maintains the minimum of each group. A final

comparison at the top level yields the global minimum. If viewed from the bottom up, this is exactly a tournament for computing the global winner. Each comparison of two vehicles is associated with an event that describes when this comparison will be violated in the future. When a violation happens, the new winner is produced and is percolated up the tournament tree, until it is either defeated or declared the global winner.

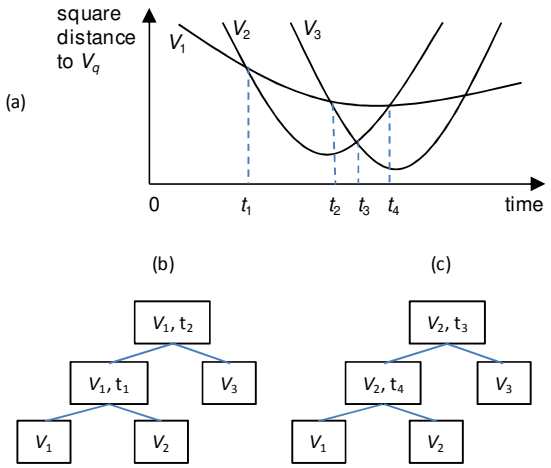

Figure 6. An example of kinetic tournament.

Figure 6 gives an example of kinetic tournament. 6(a) shows three vehicles V_1, V_2, and V_3 in the Time-SquareDistance space. 6(b) shows the tournament tree constructed at time 0. Two events are scheduled at this moment: (i) V_1 crossing V_2 at time t_1; (ii) V_1 crossing V_3 at time t_2. Since $t_1<t_2$, the V_1-crossing-V_2 event is triggered earlier, at t_1. In response to this event, the winner between V_1 and V_2 is changed, and the new winner is percolated up (see 6(c)). The V_1-crossing-V_3 event is eliminated, and two new events are scheduled, at time t_4 and t_3 respectively.

Now we quantify the intuition that less computational cost is wasted due to a motion update in the on-line case than in the off-line case. For kinetic tournament, computing the next answer pair involves the percolation of the new winner. Because a tournament tree is balanced, the percolation visits at most O(log$n$) nodes. A visit to each of these nodes may result in elimination of an existing event and creating of a new event. If we use a priority queue to store the relevant events (at most O(n)), then the elimination or creation of an event takes O(log$n$) time. Thus, the percolation takes O(log^2n) time. In other words, for on-line processing O(log^2n) time is wasted due to a motion update. Now consider off-line processing. The state-of-the-art is a divide-and-conquer algorithm which computes all the answer-pairs in O($n$log$n$) time (see [6]). Thus in the off-line processing, O($n$log$n$) time is wasted due to a motion update, much higher than the on-line case.

It can be shown that the cost of processing a motion update is O(log^2n) for kinetic tournament and O($n$log$n$) for off-line. Details are omitted due to space limitations.

Finally, let us extend the kinetic tournament to general k and r. The extension to general k is straight-forward: instead of picking the lowest vehicle for each comparison, pick the k lowest ones. The extension to general r is as follows. Before constructing the tournament tree, use a horizontal line with the height of r^2 to truncate the vehicles in the Time-SquareDistance space. The portion of each vehicle that is below the horizontal line participates in the tournament.

B. *Dynamic Adjustment of k and r*

In a real-world environment, k and r need to be determined automatically at run-time since traffic flow varies depending on road condition. While drivers move on the highway, they are likely to focus on a small number of vehicles around them. However, on downtown roads or at intersections, a special attention should be paid to a larger number of nearby vehicles than on the highway.

According to Greenshield's model [2], the traffic speed and vehicle density are linearly related and the traffic speed increases with as the vehicle density decreases. For example, when the vehicle density approaches 0, vehicles can travel at the free flow speed.

In this paper, the value of k is determined by the following formula.

$$k = k_{\max} \cdot \left(1 - \frac{vel}{vel_{ff}}\right) \tag{4}$$

where $k_{\max}$ is the upper bound value of k and is provided by the system. According to the above formula, k decreases as the traffic speed increases, which coincides with our intuition. For instance, given $v_{ff} = 150km/h$ and $k_{\max} = 10$, when vel is 120km/h, 80km/h, and 40km/h, k is 2, 5, and 8, respectively.

Similar to the dynamic adjustment of k, we adapt the radius (i.e., r) of the query region to a variety of traffic conditions. The value of r is given by the following formula.

$$r = r_{\max} \cdot \left(\frac{vel}{vel_{ff}}\right) \tag{5}$$

where $r_{\max}$ that is a system parameter indicates the upper bound value of r. It is important to note that the value of $r_{\max}$ is limited to the maximum transmission distance of the wireless technology employed for the VANET. For example, for DSRC communication, the transmission distance is up to 1km and $r_{\max} \le 1km$; for WiFi communication, the transmission distance is up to 140m and $r_{\max} \le 140m$. According to the above formula, r increases with vel. On congested roads, CRNN query results show the vehicles within a short distance from current location. For instance, given $v_{ff} = 150km/h$ and $r_{\max} = 200m$, when vel is 120km/h, 80km/h, and 40km/h, r is 160m, 107m, and 54m, respectively. Finally, from Formulae (4) and (5), the relationship among vel, den, k, and r can be summarized as follows: (i) k increases with den (ii) r increases with vel.

V. SAFETY-RELATED SCENARIOS

In this section, we apply CRNN queries to the three vehicle safety applications listed in Table 1.

A. *Cooperative Collision Warning*

The cooperative collision warning service helps the driver avoid or reduce collisions with the rear end of vehicles through the notification or warning of the impending collision. The proposed methods for CRNN queries enable the system to track the change of the velocity and the distance to each of the nearby vehicles. As shown in Figure 7, when vehicle V_j reaches a collision possible region (CPR) of vehicle V_i, the possibility of collision is computed using their dynamic attributes. If the collision is expected to occur in a short time period (e.g., 3 seconds), both drivers of these vehicles will receive a collision warning signal such as screen blinking or beep sound. In addition, through the screen of the navigation system, the driver can see the location and velocity of the vehicles that may cause collisions.

Figure 7 depicts an example of cooperative collision warning where d_{CPR} (e.g., 30 meters) is referred to as the radius of the CPR. In this example, if vehicle V_j does not decrease the current speed (100km/h), it is likely to collide with vehicle V_i in 3 seconds because the difference of the speeds of the two vehicles is 40 km/h (i.e., 11 m/s). In this case, when the distance between the two vehicles approximates to d_{CPR}, the possibility of collision is determined based on their dynamic attributes. If the possibility of collision exists, the warning signal is generated. Similarly to the dynamic adjustment of k and r in Section 4.B, the value of d_{CPR} may be determined at run-time by the formula $d_{CPR} = d_{\max\_cpr} \cdot \left(\frac{vel}{vel_{ff}} \right)$ where $d_{\max\_cpr}$ (e.g., $d_{\max\_cpr} = 50m$) is the upper bound value of d_{cpr} and is provided by the system.

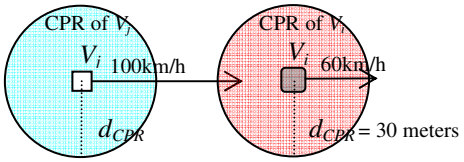

Figure 7. Example of cooperative collision warning

B. *Lane Change Warning*

The lane change warning service warns the driver if an intended lane change may cause a crash with a nearby vehicle. When vehicle V_j intends to change lane, it should change its direction toward the target lane. Similar to the cooperative collision warning system, if V_j reaches the CPR of vehicle V_i and there is a possibility of collision between the two vehicles in a short time, the collision warning signal is generated. The possibility of collision within the CPR is estimated using their dynamic attributes.

C. *Pre-Crash Sensing*

The pre-crash sensing service prepares the driver for imminent, unavoidable collisions. Figure 8 illustrates an example of pre-crash sensing. It is assumed that unfortunately both drivers of the two vehicles do not notice each other due to the obstacles. In this case, if any of the two drivers does not change the speed, the collision will occur. When vehicle V_j reaches the CPR of vehicle V_i, the possibility of crash between the two vehicles in a short time is estimated using their dynamic attributes. When an impending crash is detected, the warning signal is given. In addition, since the CRNN query result continuously returns nearby vehicles, drivers are not likely to change their velocity abruptly toward other vehicles

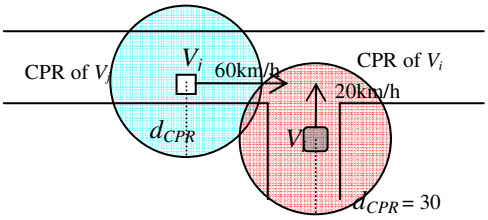

Figure 8. Example of pre-crash sensing

VI. CONCLUSIONS AND FUTURE WORK

In this paper we introduced a new type of query called CRNN queries which enable drivers to observe vehicles in the neighborhood and to avoid collisions. We developed a method for a vehicle to maintain the locations of neighboring vehicles via V2V communication. The method dynamically adjusts the location broadcast frequency using Greenshield's traffic flow model. Simulation and road test results showed that our method achieves a superior performance over its rival in terms of the bandwidth usage. Then we analyzed the computational cost of processing a CRNN query. In the on-line case it takes $O(\log^2 n)$ time to process a motion update; in the off-line case it takes $O(n \log n)$ time. Thus on-line processing is more efficient than off-line processing. Finally we demonstrated that CRNN queries are applicable to several safety-related application scenarios of great interest.

In the future, we plan to take the location uncertainty into account. Because the motion update is discrete, there is uncertainty when a vehicle predicts another vehicle's location. In this case, a CRNN query may ask for vehicles that are *possibly* the k nearest neighbors, or the vehicles that are *definitely* the k nearest neighbors. We will study CRNN queries under various query semantics and various location uncertainty models. Another research direction is how to make the processing more efficient by using appropriate spatio-temporal indexing structures.

VII. REFERENCES

[1] A. Sistla, O. Wolfson, S. Chamberlain, and S. Dao: Modeling and Querying Moving Objects. ICDE 1997: 422-432.

[2] B. Greenshield: A study of traffic capacity. Highway Research Board Proc., 14:448–477, 1935.

[3] Y. Li, J. Yang, and J. Han: Continuous K-Nearest Neighbor Search for Moving Objects. SSDBM 2004: 123-126.

[4] Vehicle Safety Communications Project Final Report, http://www.nhtsa.gov/DOT/NHTSA/NRD/Multimedia/PDFs/Crash%20Avoidance/2005/CAMP3scr.pdf, 2005.

[5] J. Basch and L. J. Guibas. Data Structures for Mobile Data. Journal of Algorithms, 31:1-28, 1999.

[6] P. K. Agarwal and M. Sharir. Davenport-Schinzel Sequences and Their Geometric Applications. In Handbook of Computational Geometry, edited by J. R. Sack and J. Urrutia, North-Holland, 2000.