Proceedings of 3rd International Workshop on Intelligent Transportatio (WIT), pp. 49–54, March 2006, Hamburg, Germany

# Vehicular Ad-Hoc Networks: Single-Hop Broadcast is not enough

Sascha Schnaufer, Holger Füßler, Matthias Transier, Wolfgang Effelsberg
Computer Science IV
University of Mannheim
Mannheim, Germany
{schnaufer,fuessler,transier,effelsberg}@informatik.uni-mannheim.de

***Abstract*— Safety Applications in Vehicular Ad-Hoc Networks (VANETs) pose tough requirements to the communication system: It has to strictly follow given quality-of-service constraints or to get along with very sparse or very dense networks. While the networking community works hard on these problems, their solutions are very often hard to integrate into well-known protocol or network-programming architectures. On the other hand, vehicular safety experts desire well-defined programming interfaces for the communication part of their applications. Very often, this is solved by using only the most simple network protocols such as single-hop broadcast or simplistic flooding, neglecting results of Ad-Hoc research. In this paper, we study the problems created by this dilemma and propose SLOPE (Self-Organizing Communication with Protocol Elements), an attempt to separate protocol functionality in application and communication domain leaving the common protocol practice with network layering and sockets. With this system and our current approach of implementation, creating VANET safety application protocols becomes as simple as extending a Java class and filling out some methods.**

## I. Introduction

Vehicular Ad-Hoc Networks (VANETs) are a special kind of Mobile Ad-Hoc Networks (MANETs), where wireless-equipped (road) vehicles form a network without any additional infrastructure. While many communication scenarios exist for these networks, government-sponsored research activities like the German Network-on-Wheels project [1] mainly focus on the application of VANETs to increase vehicular safety with extra room for applications increasing driver convenience.

The quest for a communication system to alleviate active vehicular safety has led to a break with many conventional paradigms of protocol architectures. Such a system is related as much to traditional wireless IP networks as to sensor networks which use quite different types of network protocols. [2] extensively discusses challenges in this area and states that a network stack in such a safety-critical system can hardly be regarded as separable as in IP networks. On the contrary: We expect that VANETs will be used with highly specialized protocols that require only a minimal amount of packets being sent, since one applications' packet might prevent another application from delivering a vital message. Thus, in the remainder of this work, we will restrict the discussion to a communication system and communication protocols that only serve safety purposes. We assume that the system operates on a government-regulated safety-only frequency and that any other communication would occur on a different channel. In the following, we will derive the basic principles upon which we will build our system.

**Why is single-hop broadcast insufficient?** While on one hand people working on communications mostly do not have the knowledge required to design and tune vehicular safety applications, vehicular safety engineers are mostly unaware of the consequences that the sending of a packet might have on the own or other applications' functioning. Let us illustrate this by an example: Consider an application that is disseminating information about an accident. A simple approach would be to use single-hop broadcast to send a message to all neighbors and to simply repeat the packet if receiving one. To avoid the packet to circulate forever, a time-to-live value, that will be decremented on every retransmission, is added. Additionally, the application will discard packets that are not relevant for the neighborhood. While this is a straight-forward solution to the communication problem, it will certainly not fulfill its task under all network conditions. The reason is that single-hop broadcast is unacknowledged and it can consequently be

destroyed by collisions. Additionally, there can be an explosion of nodes re-sending a packet, when the neighbors, and then the neighbors of the neighbors, etc., try to re-broadcast it. In literature, this effect is known as the "Broadcast Storm Problem" [3]. There are several methods like delaying the re-transmission for a random time to conquer this problem [4], and they require a distributed flooding algorithm.

**Why does not (optimized) flooding solve the problem, either?** In accordance with the identified problems, the next step is to require the communication system to provide a flooding mechanism that uses the available know-how. But still, that might not be the solution for all the problems, since every data packet is doomed to travel un-altered for the selected number of hops and every car that tries to add information has to send a different packet. Obviously, the communication system is not able to detect redundancy, since it does not *understand* the packets' contents. Additionally, flooding stops immediately at the borders of the current network partition, leaving it to the application to repeat the floods to reach cars that were in another partition when the flooding took place. That potentially creates a lot of redundancy.

**What about storing the packets and sending them to new neighbors?** To overcome the restrictions induced by partitions, one could store the packet destined for an area and resend it whenever a radio packet from a node that was not considered a neighbor before, is received. However, this solution also has some shortcomings: (a) the store would obviously be limited and thus it is likely to overrun at a partition's edge, (b) for every packet received the whole packet-store would have to be checked, requiring CPU time and creating delay, and (c) any node carrying the packets would create a huge concentrated network load whenever two partitions join. Additionally, the packet store would also contain a lot of redundancy, since VANET events like the detection of an accident or certain road conditions are usually correlated and cause more than one car to issue packets.

**Where do we go from here?** To solve the discussed problems, the system should combine the abilities of network and application layer: it should make use of the application's ability to understand the packets contents in order to eliminate redundancy, and it should use the knowledge about broadcast storms to control retransmissions. Using the *information connector* proposed in [2], the system could, e.g., react to events signaling new neighbors. These can then be informed by summarized information rather than by a bunch of packets. How desirable such a protocol might be, it cannot easily be separated into a network layer and an application layer problem, and its design thus requires integrated knowledge of both layers.

Since standardization efforts try to reach convergence, there is a need for cooperation and finally consensus between network layer (as in the Network-on-Wheels project [1]) and VANET safety application efforts (as in projects like WillWarn [5]). On the way to alleviate these issues, we present SLOPE (Self-Organizing Communication with Protocol Elements), a VANET communication platform that narrows the gap between vehicular safety specialists providing a simple, yet powerful method to use so-called protocol elements. As opposed to a layer-encapsulated protocol like, e.g., TCP, these protocol elements only become full protocols when complemented by application-side PEs (provided by application developers). Figure 1 shows the protocol architecture presented in [2]. The yellow line denotes the minimal communication system consisting only of single-hop services and the information connector, which uses a publisher/subscriber mechanism to distribute information between protocol entities. Note that this paper focuses on safety applications only. These would usually be built on top of the single-hop layer. The orange and blue puzzle pieces denote the placement of SLOPE within this (protocol) architecture. The provided PE (orange) is complemented by its blue application-provided counter part. Together, they form a VANET protocol that covers all layers from single-hop to application layer, dealing with single-hop, multi-hop, and transport issues. It is important to note that the only way to build a working protocol is if both protocol peers comply with each other.

It remains to remark that SLOPE is not only a protocol architecture but also a prototype system that is currently being built. The goal of this system is to achieve a better cooperation between communication and safety engineers. Besides offering PE interfaces, the SLOPE systems tries to hide communication specific issues like, e.g., packet encoding, to speed up the process of building a demonstrator system.

The rest of this paper is organized as follows: While the above example is rather simple, the next section will describe the principals of two important protocol elements. Section III gives an overview of the SLOPE prototype system and Section IV concludes the paper glimpsing at our current and future work.

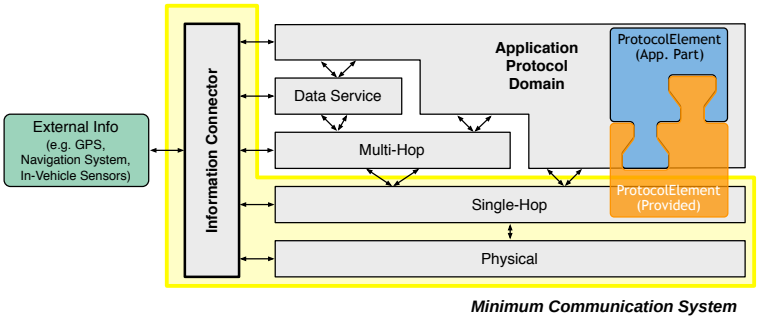


Fig. 1
PROTOCOL ELEMENTS WITHIN A VANET PROTOCOL ARCHITECTURE

## II. PROTOCOL ASPECTS

While this paper is mainly about system design, we want to outline two examples to show where cooperation between communication and application is required or beneficial. We will further point out some general requirements that exist for the application PEs as well as for the basic SLOPE system and its network PEs.

### A. Protocol Element Examples

**Information flooding:** The fundamental advantage in including application knowledge to VANET protocols is that it enables the network system to 'understand' the payload of the packet. With this knowledge, the system is able to detect redundancy in messages and since the medium is the bottle-neck resource the most important goal is to avoid redundancy. So, a typical task of the application PE is to provide a method that tells the PE provided by the SLOPE system if a given packet $p$ contains information that has already been sent or heard within a certain time window. Then, the system is able to safely discard a message if the channel conditions require it. Moreover, we can detect which neighbor has rebroadcast certain information and use that for passive acknowledgments.

Extending the example outlined in Section I, we picture an application that is supposed to disseminate information about local hazard conditions detected by the car's sensors. For the reasons described above, the dissemination should (a) be multi-hop while avoiding broadcast storms, (b) be quick, (c) avoid redundancy, and (d) be aware of the network load. In addition, it should cover more than the current network partition. Thus, such an Information Flooding (IF) PE could behave as follows: if the application confirms that the information in a received packet has not been sent for some time, the packet can be rebroadcast. This eliminates some information redundancy. The counterpart, the PE on the network layer in return provides a jittered resend which can be parameterized by network density, load condition, or even by the geographic distance of the last sender and the current node as in Contention-Based Forwarding (CBF) [6]. In addition, the lower PE can select different forwarding modes to transmit a data packet to every neighbor. E.g., it could use anything from a simple single-hop broadcast up to a separate unicast to each neighbor. The selection of the method that is actually used can safely be left to the communication system, relying on its knowledge about which single-hop transmission method it the best for the current conditions.

**Periodic beaconing:** The second application protocol element is intended for SOTIS-like applications (Self-Organizing Traffic Information System) [7] that periodically transmit information vectors. To accomplish this, the application PE registers a method returning the current information to send. This method is called by the network PE right before sending.

The advantage of this PE pair is the synergy that arises when different protocols of this kind are running. E.g., the beacons used by the communication system to asses the neighborhood situation are relatively small messages, and another application could fill the unused packet space with information. Since channel access is often the critical part of the MAC protocol, increasing the packet size is cheaper than sending separate packets. Of course, the network system can adapt the rate of the periodic messages to the current network density and load [8].

### B. General Requirements

**Application state database:** The application part of the Protocol Element carries the complete application logic to assess the safety-related situation by

evaluating sensor events gathered by the local information connector *and* by external messages. In addition, it initiates the sending of information or packets. To be able to decide if some received information is new, an application has to keep track of the messages received before and expire them if required. This can be done by means of a state database that stores knowledge about the situation in the surroundings or traffic information about streets further away.

**Computational complexity in the application:** An important issue within the SLOPE system is the computational complexity of the methods that are modeled by the protocol state machines. If an application takes some seconds to check whether some information was already received, or if it even blocks while waiting for an event, it is unsuited for a safety system. Thus, there will be an upper bound for the reaction time of an application PE that the application has to obey in order to not constrain the corresponding network PE. Nevertheless, any delay slows down the protocol and diminishes the performance.

**Price per unit:** Another important factor is the price per communication unit. In order to quickly achieve a high market penetration, the unit has to be inexpensive. Vendors might want to build on-board units with different resource configurations. E.g., one unit might have less memory and CPU power than others or might even not be connected to on-board sensors. While this would, of course, have a negative impact on the features provided by the system, the protocols could easily incorporate such "simpler nodes" into the network. An application PE on such a system could, e.g., only relay information packets on the base of random jitters (that are not based on geographical positions) and—if it does not understand the packets' contents or is not able to keep track of previously seen information—assume it has not seen them yet. From a protocol perspective, such a system would stress the channel resources more than a full-featured one, and it will be subject of our further research, how many of these "dumb nodes" can be handled by the network.

**Load saturation:** There is a lot of discussion about using as much of the channel as possible in order to fully exploit the scarce resource. Much as tempting that may sound, a fully loaded ad-hoc network has many drawbacks, especially for protocols using unacknowledged link layer broadcasts. The reason for this is the interference behavior of radio systems that leads to the hidden terminal problem. Each packet that is sent might destroy packets in the surroundings. In a network with safety-critical messages, this means that no unnecessary packets must be sent. Therefore, the access granted to uncontrolled applications in an open communication system has to be restricted. In a closed communication system, however, only the own applications have to be controlled. Under this circumstances, it is feasible to build a communication system that is likely to work. Open networks, i.e., networks where arbitrary applications are allowed to send messages would either have to relentlessly drop these low-priority messages—which could potentially break the applications—or to accept a network load that may prevent emergencies from being propagated fast enough. This would be worsened further if multi-hop packet protocols [9] were allowed. Summing up, for VANET safety channel communication, we strongly advocate a non-open communication system operating at a minimum channel load.

## III. Architectural Overview

In this section, we will explain the SLOPE prototype system. It serves as a basis for the example applications described above and aims to fulfill the described requirements. As outlined previously, the SLOPE prototype system is designed to simplify the development and testing of VANET safety protocols. However, there is a trade-off between ease of use on one side and performance on the other. During application development the former is more important, while the final system will have to be more performance-oriented and resource-saving. Mainly out of these reasons, we have chosen Java as a platform for all SLOPE components in our prototype system. Link layer packets are sent using the *Java Native Interface* [10] in combination with *Linux Packet Sockets*. The SLOPE components are depicted in Figure III. The yellow box contains the complete safety communication on-board unit. In the final system, we envision this to run on a single box, complemented by on-board sensors and actors, depicted in the green box. On one hand, this system enables the communication system to be, e.g., aware of the car's position, and on the other hand, all logic that reacts to events generated by the communication system resides in this green box. This allows car-vendors to easily interface their proprietary technics with the system.

Besides the information connector and the 'link layer and below' radio system, the SLOPE system runs within a single Java Virtual Machine (VM), including the provided parts of the Protocol Elements.

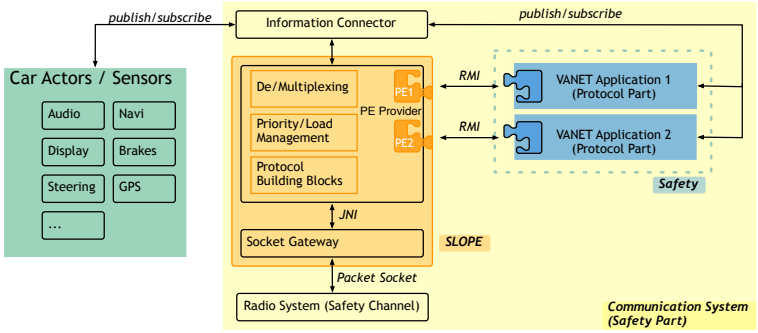


Fig. 2
SLOPE COMPONENTS

Implementing the application side of of a PE (the blue puzzle pieces) is as easy as creating a sub type of a provided Java class. This class encapsulates communication with the network counterpart allowing the application to be run either in the server VM or, preferably, in a separate VM. In the second case, the communication between corresponding protocol elements is carried out by means of Java Remote Method Invocation (RMI) [11]. Of course, RMI itself will create additional delay for each method call and it will slow down the system considerably. This is even worsened since SLOPE requires any calls to the application VM to be non-blocking in order to avoid that the application VM may pause the system. Despite the performance cost, we are convinced that the ability to run in different VMs is important in the development phase since it allows a separated development, almost regardless of the operating system platform. Moreover, even the SLOPE system's VM can run on a different system than the one carrying the radio system, allowing a centralized debugging. E.g., a number of SLOPE VMs could be run on one development computer while the wireless hardware could be located in an experimental setup.

In conventional IP networks a communication system offers services to deliver packets, i.e., chunks of bytes, to a given destination. To use these, an application opens a socket and sets some socket options like destination IP address and port. Then the application constructs its chunk of bytes and hands it to the socket. On the destination node an application opens a corresponding socket to receive the data. In this case the only communication between network layer and application is limited to setting and reading socket options [12] and returning values from the *send()* function call. In order to alleviate protocol development the client side of the PEs is much more comfortable. Building a packet is as easy as deriving from a provided packet class, and the SLOPE system handles all the packet marshaling. Of course, this comes at some performance cost. However, before roll-out this protocol part can easily be optimized with regard to packet size and marshaling/unmarshaling performance.

**Potentialities for performance improvement:** As noted above, performance was not the goal of the software design in this development phase. However, we have considered design alternatives for some aspects that are potentially performance critical. E.g., since the usage of RMI is fully encapsulated inside the PE glue architecture (and independent of packet marshaling), it could be replaced by a simple UNIX or IP socket scheme if desired. This could happen without changing the client code. In a later project phase, even Java could become optional on system and application side, and be replaced by a language generating OS native binaries like C++. However, this would also require the application part to be rewritten.

**Security considerations:** Security is a very important part of VANETs and also a lively part of VANET research, and it should always be considered. The SLOPE system, which is per se not an open communication system, is potentially easier to be secured than an open platform. Defining the yellow box as the communication system, no trust relationships within the box would have to be verified. Thus the whole box could be accredited and only the information connector would require secure access. In consequence, security efforts could concentrate on the protocols.

**Code integration vs. heavy weight socket interface:** Our approach is to integrate the execution of application logic whenever the communication system is required to understand a packet's content. A different approach would be to extend the network

header by some fields—such as, e.g., spatial or temporal validity—that allow the network layer to perform some aggregation. Of course, this would come at the cost of a heavier socket interface but would still work in combination with the information connector.

However, we are convinced that SLOPE outperforms it (a) in terms of simplicity from the application view and (b) in terms of flexibility in the development process. Concerning (a): In heavy weight socket protocols, the communication system has to have some understanding of the packet semantics, at least of some larger parts of the header. Our approach is to leave all these semantics to the application, and to query them on-demand. This allows to leave the packet definitions inside the application domain, which guarantees a very light weight and standardizable communication system. For (b): Abstracting the application protocols into one or multiple protocols requires yet more knowledge about them, but later on, the socket approach could be feasible. In addition, the final system will probably not be an open communication system, at least for the safety part. Thus, in contrast to conventional communication systems, where layering with designed-to-be-simple interfaces guarantees extendibility, the protocol integration does not hurt at all.

As outlined in Section II-B, the capability requirements for the on-board unit depend on the complexity of the contained communication system. Considering the price per unit, a tiny resource configuration is desirable. However, a significant gain in network performance could be reached if the system incorporates the application logic as outlined above. Moreover, the final system could combine moderate resource requirements with channel efficient networking while necessitating only a few dozen lines of application code.

## IV. Conclusions and Future Work

In this paper, we have presented SLOPE, a communication system tailored to suit VANET safety communication protocols. It acknowledges that these protocols have to be aware of effects on all layers. Furthermore, we have presented example protocol elements and we have shown how networking know-how can be beneficial for the application and vice versa. In addition to the concept, we outlined the prototype implementation that we are currently working on. With this Java-based system, creating and testing VANET protocols should be quick and easy.

Regarding our ongoing and future efforts, we strongly hope that SLOPE is adopted by application development groups to (a) prove the concept and (b) work towards the goal of reasonable all-layer VANET protocols by sharing the responsibility for the medium. On the network side we are working on the two PEs presented here and some more that might be of interest for application developers. Still focusing on communication, we will tailor VANET protocols to information dissemination and look into the effects of network load on overall performance. Further, the system will have to be tested and evaluated in order to assess the seriousness of the performance sacrifices that were made for the sake of simplicity.

Following the proposed development path, we are sure to be able to forge the desired system in a short time frame. The final system—which will probably not be based on Java—should be feasible with low resource requirements and still be able to incorporate most of the application logic.

## Ackowledgements

Sascha Schnaufer acknowledges the support of the German Ministry for Education and Research (BMB+F) for the "Network-on-Wheels" project under contract no. 01AK064F.

## References

[1] "The Network on Wheels Project," `http://www.network-on-wheels.de`.

[2] Holger Füßler, Marc Torrent-Moreno, Matthias Transier, Andreas Festag, and Hannes Hartenstein, "Thoughts on a Protocol Architecture for Vehicular Ad-Hoc Networks," in *Proc. of WIT 2005*, Hamburg, Germany, March 2005, pp. 41–45.

[3] Sze-Yao Ni, Tseng Yu-Chee, Chen Yuh-Shyan, and Sheu Jang-Ping, "The Broadcast Storm Problem in a Mobile Ad Hoc Network," in *Proc. of ACM MobiCom '99*, Seattle, Washington, August 1999, pp. 151–162.

[4] Brad Williams and Tracy Camp, "Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks," in *Proc. of ACM MobiHoc '02*, Lausanne, Switzerland, June 2002, pp. 194–205.

[5] "PReVENT :: WILLWARN Project Homepage," `http://www.prevent-ip.org/en/prevent_subprojects/safe_speed_and_safe_following/willwarn/`.

[6] Holger Füßler, Hannes Hartenstein, Jörg Widmer, Martin Mauve, and Wolfgang Effelsberg, "Contention-Based Forwarding for Street Scenarios," in *Proc. of WIT 2004*, Hamburg, Germany, March 2004, pp. 155–159.

[7] Lars Wischhof, André Ebner, Hermann Rohling, Matthias Lott, and Rüdiger Halfmann, "SOTIS - A Self-Organizing Traffic Information System," in *Proc. of IEEE VTC Spring '03*, Jeju, Korea, April 2003.

[8] Lars Wischhof, André Ebner, Hermann Rohling, Matthias Lott, and Rüdiger Halfmann, "Adaptive Broadcast for Travel and Traffic Information Distribution Based on Inter-Vehicle Communication," in *Proc. of IEEE IV 2003*, Columbus, OH, June 2003.

[9] Piyush Gupta and P.R. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, vol. 46, no. 2, pp. 388–404, March 2000.

[10] "Java Native Interface," `http://java.sun.com/j2se/1.5.0/docs/guide/jni/`.

[11] "Java Remote Method Invocation," `http://java.sun.com/products/jdk/rmi/`.

[12] W. Richard Stevens, Bill Fenner, and Andrew M. Rudoff, *UNIX Network Programming*, vol. 1, Addison-Wesley, 3rd edition, 2004.