Gremlin: Scheduling Interactions in Vehicular Computing

Kyungmin Lee, Jason Flinn, and Brian D. Noble
Univeristy of Michigan

ABSTRACT

Vehicular applications must not demand too much of a driver's attention. They often run in the background and initiate interactions with the driver to deliver important information. We argue that the vehicular computing system must schedule interactions by considering their priority, the attention they will demand, and how much attention the driver currently has to spare. Based on these considerations, it should either allow a given interaction or defer it.

We describe a prototype called Gremlin that leverages edge computing infrastructure to help schedule interactions initiated by vehicular applications. It continuously performs four tasks: (1) monitoring driving conditions to estimate the driver's available attention, (2) recording interactions for analysis, (3) generating a user-specific quantitative model of the attention required for each distinct interaction, and (4) scheduling new interactions based on the above data.

Gremlin performs the third task on edge computing infrastructure. Offload is attractive because the analysis is too computationally demanding to run on vehicular platforms. Since recording size for each interaction can be large, it is preferable to perform the offloaded computation at the edge of the network rather than in the cloud, and thereby conserve wide-area network bandwidth.

We evaluate Gremlin by comparing its decisions to those recommended by a vehicular UI expert. Gremlin's decisions agree with the expert's over 90% of the time, much more frequently than the coarse-grained scheduling policies used by current vehicle systems. Further, we find that offloading of analysis to edge platforms reduces use of wide-area networks by an average of 15 MB per analyzed interaction.

CCS CONCEPTS

• **Human-centered computing** → *Mobile computing*; • **General and reference** → *Design*; • **Software and its engineering** → *Software system structures*;

KEYWORDS

Vehicular applications; Driver distraction; Edge offload

ACM Reference Format:
Kyungmin Lee, Jason Flinn, and Brian D. Noble Univeristy of Michigan. 2017. Gremlin: Scheduling Interactions in Vehicular Computing. In *Proceedings of SEC '17*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3132211.3134450

SEC '17, October 12–14, 2017, San Jose / Silicon Valley, CA, USA

ACM ISBN 978-1-4503-5087-7/17/10…$15.00
https://doi.org/10.1145/3132211.3134450

1 INTRODUCTION

Vehicular computing applications execute in an attention-limited environment. Unlike desktop applications that typically demand most of a user's attention, vehicular applications must require much less attention so as to avoid distracting their users from the primary task of driving the vehicle. Consequently, many vehicular applications such as turn-by-turn directions, location-based reminders, and messaging middleware run in the background and try to interact with a driver only when the interaction will be meaningful. Instead of the user initiating the interaction at a convenient moment, e.g., by opening a desktop application, the vehicular application initiates the interaction, e.g., via an audio tone from an in-vehicle HMI (human-machine interface).

Thus, vehicular computing systems must *schedule* interactions, which we define to be a related sequence of user inputs and outputs that corresponds to performing a single logical task. When an application wishes to interact with the driver, the system should consider the priority of the interaction (e.g., is this a critical alert about an upcoming road hazard?), the complexity of the interaction (e.g., is this a short audio message or will the interaction involve navigating several screens on the vehicle touchscreen?), and the difficulty of driving conditions (e.g., is the user driving in heavy traffic in icy conditions or in light traffic on a pleasant day?). Based on such assessments, it should either initiate the interaction with the driver or defer it until a more appropriate time.

Current vehicular infotainment systems employ simple scheduling policies that consider only the high-level interaction type (e.g., voice-based vs. text-based) and whether or not the vehicle is moving. For instance, Chevrolet MYLINK [16], Toyota Entune [57], and Mazda Connect [38] allow voice-based interactions for new text messages but disallow interactions that require reading from the touchscreen unless the vehicle is stopped. These coarse-grained scheduling policies ignore two critical factors: (1) a driver's available attention changes significantly as driving conditions vary, and (2) interactions of the same type can demand different amounts of attention.

Although these existing scheduling policies assume the driver's available attention is unchanging, numerous studies [37, 47, 50, 58] show wide variability due to speed, traffic volume, and driver experience. For instance, Senders et al. [50] report a 38% decrease in available visual attention as speed increases from 30 MPH to 60 MPH, and Patten et al. [47] show a 30% difference in available cognitive attention due to varying driver experience. This large variation means that the voice-based interactions allowed in "normal" driving conditions by current systems are inappropriate for complex driving situations.

Conservatively disallowing all interactions is not good policy either. Research has shown that when interactions are disallowed,

divers seek alternative, riskier approaches to obtain information. AT&T [3] reported that 62% of drivers keep their smartphones in easy reach and 70% engage in smartphone activities (e.g., checking new text messages) at least some of the time while driving. Survey participants cited habit, the fear of missing something important, and the belief that both driving and smartphone interaction can be done safely as the three main reasons for this behavior. Comments in a National Highway Traffic Safety Administration report state that "Consumers have numerous connectivity options, particularly via portable electronic devices. They will quickly migrate to alternate, and potentially more distracting and less safe, means of staying connected if the use of in-vehicle or integrated options is overly curtailed." [41]. In other words, overly conservative scheduling policies may actually have a negative effect by causing drivers to bypass safer in-vehicle HMIs and use smartphone UIs.

A second issue with current scheduling policies is that not all interactions of the same type demand the same amount of attention. For instance, an audio tone demands less attention than a complex spoken sentence, and a pop-up requires less attention than a multi-screen interface with text and graphical elements. Current guidelines and standards for assessing interaction content and minimizing attention demand in a vehicular setting [11, 19, 39, 51, 55] recognize that such distinctions must be made. Simply ignoring them can overload the driver or fail to initiate useful interactions.

Based on these observations, we have built *Gremlin*, system support for scheduling application-initiated interactions in vehicular settings. Gremlin continuously performs four tasks: (1) it monitors driving conditions to estimate the driver's available attention, (2) it records interactions for analysis, (3) it uses edge computing infrastructure to analyze each interaction's attention demand and build a model that predicts attention demand of future interactions, and (4) it uses interaction priority, its estimate of attention supply, and its model of attention demand to *schedule* each new interaction by either allowing it to proceed or deferring it.

Gremlin quantifies attention demand by breaking each interaction into low-level I/O components such as pressing a button, reading text, and listening to audio. It uses past studies [46, 52–54, 59, 62] that quantify attention demand for these individual activities to understand the attention demand of the whole interaction and build its model. This analysis can be computationally demanding, as it involves, e.g., using speech recognition to translate audio to text and processing the text to determine the level of complexity, as well as identifying individual UI elements from a stream of GUI dumps and matching these with a video record of the display during the interaction. As these computational demands are too much for cellphones and similar mobile devices, Gremlin offloads the analysis and performs it on a server. However, offloading analysis to the cloud would consume too much wide-area network bandwidth. The average recording size for an interaction is 15 MB even after compression, so analyzing only a few interactions a day would be a drain on cellular data plans. Therefore, Gremlin is designed to offload analysis to edge computing infrastructure co-located with network access points.

Gremlin quantifies available attention as the time the driver can safely spend on a secondary task. It estimates how driving conditions affect the two dimensions of attention most critical in driving: visual and cognitive attention. For each dimension, it computes a separate deadline that represents how long the driver can safely look away from the road or spend on a non-driving cognitive task such as listening to a voice message. These values are derived from both laboratory studies [9, 47, 52, 58] and real-life crash data [61].

Given these deadlines, Gremlin determines if a proposed interaction can be scheduled based on its priority and predicted completion time. Most application-initiated interactions have lower priority than driving and should only be scheduled if they can be serviced within the deadline. However, some interactions are very high priority (e.g., road hazard alerts) and should always be delivered.

Thus, this paper makes the following contributions:

- It introduces a method for quantifying a driver's available attention based on data from laboratory studies and real-world crash data.
- It introduces a method for quantifying the attention demand of interactions based on fine-grained analysis of the I/O components comprising each interaction.
- It proposes an architecture in which mobile platforms balance computational and bandwidth concerns by recording interactions and offloading the analysis of the recording to nearby edge computing infrastructure.
- It develops a quantitative algorithm for scheduling interactions in vehicular computing that makes better decisions than current qualitative algorithms.

Gremlin is a research prototype that demonstrates the feasibility of these ideas. Our evaluation shows that Gremlin makes reasonable decisions, but a comprehensive evaluation with many hours of simulator and road testing would be required before deployment. Section 5 discuss these and other current limitations of Gremlin.

We evaluate Gremlin by comparing its decisions to those recommended by a vehicular UI expert for seven interactions in three different driving scenarios. Gremlin's decisions agree with the expert over 90% of the time, and they match the expert recommendations significantly more often than any coarse-grained scheduling policy. We further show that offloading of analysis to edge platforms reduces use of wide-area networks by an average of 15 MB per analyzed interaction compared to cloud offload.

2 DESIGN CONSIDERATIONS

We begin by discussing three important issues we considered while designing Gremlin.

2.1 What to schedule

Gremlin defines an *interaction* to be a related sequence of user inputs and outputs that corresponds to performing a single logical task. Typically, the interaction will start with a *notification*; this often will be an audio tone, e.g., the one used by Android's default `NotificationManager`. Some applications use custom notifications, such as a spoken sentence, a visual pop-up, or some combination of the above. The driver might respond to the notification and interact with the application via audio and voice commands, or the interaction may take the form of reading information from the vehicle touchscreen and responding with button presses or steering wheel controls. Interactions may be as simple as hearing an audio tone, reading text on a pop-up, and dismissing the screen, or as

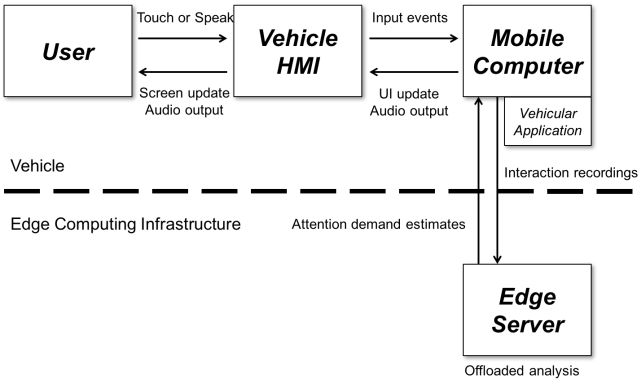

Figure 1: Overview of Gremlin's compute environment

complex as a conversation with a digital assistant to find a nearby restaurant with available seating. For interactions with multiple steps, the driver may pause between steps to look at the road. The interaction ends when the logical task is complete, typically when the notification is dismissed or the application returns to a home screen.

Some prior systems have scheduled mobile notifications. In contrast, Gremlin is designed to schedule the entire interactions that those notifications initiate. In a complex driving situation, a user may have sufficient attention available to respond to an audio tone (the notification). However, that response may initiate a more complex interaction (listening and responding to a text message), for which the driver cannot spare attention at the moment. This situation is undesirable. The driver may be implicitly led to perform an activity for which attention cannot be spared, causing distracted driving. Alternatively, the driver must decide to abort or pause the interaction upon realizing that it will be inappropriate; unfortunately, simply remembering that a task is pending places a cognitive burden on the user. We believe it is far better to only deliver a notification when the driver is able to perform the interaction that the notification initiates.

2.2 Where to schedule interactions

Gremlin could be implemented as a library that provides support for each application to schedule its own interactions. Alternatively, Gremlin could be implemented as a single system component, either in the OS or system middleware, that schedules interactions for all applications. Each design has advantages and disadvantages.

Applications have the most knowledge about the expected contents of the interactions they initiate (e.g., audio that will be output and screens that will be displayed). This knowledge allows for more accurate prediction of the attention demand of upcoming interactions. Implementing scheduling at the application layer also makes deployment easier, since few modifications would be required to system software.

However, scheduling interactions at the application level jeopardizes driver privacy. Determining available attention requires understanding current driving conditions, which in turn requires access to raw vehicle sensor data such as the current location, the vehicle speed, etc. The driver must trust that each application does not leak such data, even though studies have shown many current mobile applications sending similar data to the cloud [12]. In contrast, system software is already trusted with the privacy of sensor data, so scheduling at the system layer reduces the risk of leaks.

A second disadvantage of application-level scheduling is that coordination among independently-developed applications is difficult [40]. If applications are unaware of each other, one application may initiate a new interaction while a driver is currently handling an interaction from another. For instance, exiting a highway may cause a restaurant application to recommend nearby eating options, a gas station application to show the lowest-cost fueling option, and turn-by-turn route directions to inquire whether the route should be recalculated. Coordinated scheduling by a single entity could prevent these interactions occurring at the same time.

Based on these considerations, Gremlin schedules interactions at the system layer. Specifically, we implement Gremlin as an Android service. Gremlin works for unmodified applications by interposing on standard Android services such as the `NotificationManager` to know when interactions start and end, observe each interaction, and learn models of attention demand for each type of interaction. Gremlin further mitigates the lack of application-specific knowledge by providing interfaces that allow applications to optionally specify which actions comprise an interaction.

2.3 How to quantify attention

Gremlin's goal is to move from the current qualitative scheduling approach based on broad application categories (audio vs. visual) and coarse characterizations of driving conditions (moving vs. not moving) to a more nuanced, quantitative scheduling algorithm. This required us to develop a way to express attention supply and demand with a common numerical value.

Prior studies in the psychology literature [2, 60] find that attention is best represented with a multi-channel model. These studies argue that attention should be viewed as a composition of multiple independent dimensions such as visual, audio, and cognitive attention rather than as a single value. Furthermore, they find that a person can perform two tasks simultaneously as long as those tasks do not overload any single dimension of attention (e.g., one can simultaneously walk and listen to music). During conversations with vehicular industry usability experts, we learned that visual and cognitive are the two most important dimensions of attention for driving because the driver needs to monitor the road carefully (visual attention) and respond quickly to new events (cognitive attention). Numerous studies of driver distraction [9, 13, 29, 35, 47, 52, 58, 59, 63] employ the multi-channel model and focus solely on visual or cognitive attention. Therefore, Gremlin adopts the multi-channel model.

Next, we realized that each attention dimension is best quantified as the time that can be safely spent on a visual or cognitive task before returning attention to the primary task of driving. Visual attention supply is the time that the driver can safely look away from the road, and cognitive attention is the time that the driver can spend on a discrete cognitive task. Gremlin views these values as real-time *deadline* constraints, $V_{deadline}$ and $C_{deadline}$.

Gremlin quantifies the visual attention demand of an interaction as $V_{completion\_time}$ by summing the time needed by a driver to perform low level I/O tasks that comprise the interaction such

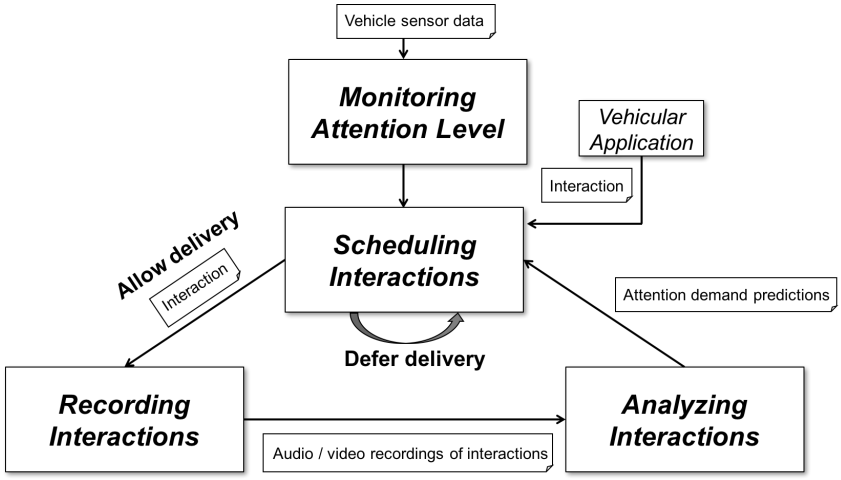

Figure 2: Gremlin performs three tasks continuously to schedule interactions

as reading text from the screen and pushing a response button. Similarly, the cognitive attention demand, $C_{completion\_time}$, is quantified by measuring the additional delay in driver response time as a result of performing secondary tasks [43, 52, 53, 56]. For example, $C_{completion\_time}$ captures the cognitive demand imposed by audio-based interactions based on type (sound or speech) and complexity.

Quantification of attention enables the use of simple real-time scheduling algorithms. If the initiated interaction has lower priority than driving (as is commonly the case), then the interaction should be allowed if $V_{completion\_time} < V_{deadline}$ and $C_{completion\_time} < C_{deadline}$; otherwise it should be deferred until the condition is satisfied.

3 IMPLEMENTATION

We first present an overview of Gremlin, and then we describe each of its main components in detail.

3.1 Overview

Figure 1 shows the computational environment Gremlin targets. Vehicular applications execute on a mobile device such as a cellphone. The mobile computer is linked via Bluetooth to the vehicular HMI which provides I/O capabilities. Current vehicular platforms such as Ford's AppLink and the MirrorLink standard allow applications on a cellphone to display on a dashboard touchscreen, receive UI events from the touchscreen and steering wheel controls, and perform audio I/O through the vehicular HMI. The cellphone links to nearby edge computing infrastructure through one-hop or local-area wireless connections. The infrastructure supports offloading of stateless computation from the mobile phone to the edge platform. Edge offloading enables the execution of computation too demanding for the cellphone without incurring the cost of sending large amounts of data to the cloud over expensive wide-area networks.

The analysis required by Gremlin has high computational demand; our evaluation shows that it requires an average of 12.7 seconds per interaction on a well-provisioned server. Currently, both in-vehicle compute platforms and modern smartphones lack sufficient resources to run Gremlin's analysis. Offload of analysis is thus the only workable option.

We also note that sufficiently powerful compute resources are unlikely to be included in future vehicles for several reasons. First, most vehicles are commodity products, and manufacturers optimize designs to save pennies; adding substantial general-purpose compute power is too expensive. Second, it takes several years to bring a vehicle from concept to market; once manufactured, vehicle components are designed to last for at least ten years. Even a leading-edge processor installed in a new vehicle will be woefully inadequate a decade layer. Finally, reliability is very important in consumer rankings of vehicles, and the challenge of debugging and patching software in general-purpose vehicular computers is daunting. For these reasons, the industry trend is for the vehicle to provide input/output capability and sensors, while smartphones brought into the vehicle execute applications.

As shown in Figure 2, Gremlin continuously performs four tasks: (1) it monitors driving conditions to estimate the driver's available attention, (2) it records interactions with the user initiated by applications, (3) it analyzes the recorded interactions to generate a model of attention demand for each interaction type, and (4) it combines those models with estimates of the driver's available attention to schedule new interactions. Tasks 1, 2, and 4 execute on the mobile computer in the vehicle, while task 3 is offloaded to edge servers.

When the driver interacts with an application for the first time, Gremlin can immediately estimate the driver's available visual and cognitive attention ($V_{deadline}$ and $C_{deadline}$) based on vehicle speed, road curvature, driver experience, etc. However, it cannot estimate visual and cognitive attention demand ($V_{completion\_time}$ and $C_{completion\_time}$), since it has not yet seen any interactions to generate an attention model. Thus, for previously unseen interactions, Gremlin uses the default, conservative scheduling approach of current vehicle HMIs: it allow unknown interactions if the vehicle is stopped or if such interactions are solely audio-based. Once Gremlin has generated a model of demand for an interaction type, it uses that model to decide whether to allow or defer interactions of that type as described below.

Function	Arguments and return values
`gremlin_begin_notify`	`(IN application, IN id, IN priority) -> OUT success`
`gremlin_finish_notify`	`(IN application, IN id) -> OUT success`
`gremlin_cancel_notify`	`(IN application, IN id) -> OUT success`

Table 1: API for applications with custom interactions

During the second task, which begins whenever Gremlin allows an interaction, Gremlin captures audio and video of the input and output for that interaction. This includes commands spoken by the driver, audio output to the driver, information displayed on the touchscreen, etc. This data is encoded and offloaded to a nearby edge computer for analysis. Most Android applications use the `NotificationManager` to initiate interactions; Gremlin modifies this service to schedule interactions, as well as to learn when interactions start and end. Some applications use custom methods for interactions; Gremlin provides an API so that these applications can provide similar information and receive scheduling decisions. Gremlin infers that an application ends when the user dismisses the notification or leaves the application screen (i.e., returns to a home screen). Section 3.2 details how Gremlin records interactions.

In the third task, Gremlin analyzes recorded interactions to generate attention demand models for each interaction type. Gremlin determines $V_{completion\_time}$ by analyzing each screen's word count, text contrast ratio, and button sizes. Gremlin computes $C_{completion\_time}$ by analyzing the content and complexity of each audio input and output. These computed values are then returned back to the mobile computer to be incorporated into the attention model. An attention model is a distribution of $V_{completion\_time}$ and $C_{completion\_time}$ for interactions of the same type. Section 3.3 describes in more detail how Gremlin analyzes interactions.

The final task, scheduling, occurs when an application wishes to initiate a new interaction. Scheduling is done entirely on the phone based on a local model generated from offline analysis; it is done immediately when an application tries to initiate an interaction, and the scheduling decision does not wait for any offloaded operation to complete. Thus, if recently recorded interactions have yet to be analyzed when an application initiates a new interaction, Gremlin will make a decision based on a slightly stale model. This preserves good response time and still improves over current practice, which uses no model at all.

If Gremlin entirely lacks a model of attention for a given interaction type, it uses the default coarse-grained HMI scheduling policy. If the interaction type has higher priority than driving, it is allowed immediately. High-priority interactions are rare (e.g., a road hazard warning). Designating an interaction type as high-priority is a privileged operation; we envision that this will be done by the vehicle manufacturer.

Most interactions have lower priority than driving. For these, Gremlin continuously updates its estimates of the driver's available attention, $C_{deadline}$ and $V_{deadline}$. It allows an interaction if $V_{completion\_time} < V_{deadline}$ and $C_{completion\_time} < C_{deadline}$. Otherwise, it defers the interaction. Section 3.4 describes how Gremlin estimates available attention and schedules interactions.

3.2 Recording interactions

Gremlin records interactions in order to gather information sufficient to determine attention demand ($V_{completion\_time}$ and $C_{completion\_time}$) at a later time. An interaction starts with a notification or other application-initiated output and continues until the logical task associated with that output is done. For instance, an interaction can be as simple as a single audio message or a more complex sequence of audio outputs and spoken responses. A visual interaction may be a single screen, in which case it ends when the user dismisses that screen, or it may be a sequence of screens and touchscreen events.

Gremlin determines the start of interactions that use Android's default `NotificationManager` by interposing on its `notify()` method. It determines that the interaction has ended when either the application calls the `NotificationManager cancel()` method, dismissing the interaction, or when the display returns to a home screen. A few applications bypass Android and use custom methods for initiating interactions; Gremlin provides the API in Table 1 for such applications. Applications call `gremlin_begin_notify()` to start an interaction and `gremlin_finish_notify()` when the interaction is done. Gremlin uses these cues to determine when to start and stop its recording of the interaction.

Gremlin uniquely identifies each type of interaction by an $\langle application, id \rangle$ tuple, where *application* is the Android package name and *id* specifies different interactions for the same application. These values are provided directly as part of the API in Table 1 or determined from the parameters passed into the `notify()` method.

Gremlin uses Android's `screenrecord` binary to capture a video of the screen content encoded in H.264 format. It also dumps the GUI composition of the screen once per second to extract button size, text, and other information. We modified Android's `ViewServer` and `HierarchyViewer` to write this data to a single file. Gremlin uses Android's `AudioFlinger` to record all audio output, and it records audio input via Android's `AudioRecord` object. Audio recordings use a WAV encoding.

Because edge computing resources may not always be nearby, recorded interactions may be stored temporarily on the mobile computer until they can be analyzed. Storing many recordings for a long time is undesirable. First, mobile storage is limited and the recordings are large. If storage space is exhausted, Gremlin must delete older recording without analyzing them. Second, although there are not strict deadlines for completing analysis, timeliness does matter. If recordings are analyzed promptly, Gremlin reacts faster when it observes new interactions or sees changes in user or application behavior. Section 4.4 shows that recording imposes little performance and storage overhead on the vehicular computing system.

3.3 Analyzing interactions

We have developed a Java tool to analyze recorded interactions. Gremlin offloads this compute-intensive analysis by running it on an edge compute server. In this paper, we do not specifically address the trust and management issues of hosting offloaded computation, as the research community is exploring these issues in depth [5, 6, 24, 36].

Our design makes offload simpler by structuring the model generation as a series of stateless computations. Each interaction is analyzed individually based solely on the audio, video, and GUI recordings. The output of the computation, i.e., estimates of visual and cognitive demand for that interaction, are shipped back to the mobile computer, which incorporates those new values into a model. If an analysis fails to complete, it is simply restarted on another edge computer by shipping the recordings to that host.

We first show how Gremlin determines $V_{completion\_time}$ and $C_{completion\_time}$ for each recorded interaction. Our general approach is to first break an interaction into discrete input and output events, and then use studies of human performance with various interface types to quantify the time it takes to interact with each element. Summing the individual element times gives the total interaction time.

There are many factors that influence visual and cognitive demand. Our approach in Gremlin is twofold: we include the most important factors in our current analysis, but we also develop a framework and methodology which makes it easy to include additional factors in the future.

3.3.1 ***Computing visual attention demand.*** To compute $V_{completion\_time}$, Gremlin first determines the attention demanded by each unique screen that comprises the interaction. It finds all unique screens by scanning the GUI dump stream and matching each screen with a corresponding video frame using timestamps in the recorded data.

A screen's visual demand is currently determined to be the sum of the two components: text and buttons. To interact with a vehicle touchscreen, the driver uses visual attention to both read the information displayed on the screen and also to direct a finger to touch the appropriate screen location.

Gremlin determines the time to read text from the word count and text contrast ratio. The word count is determined from the GUI dump, and the contrast ratio comes from the screen dump. The average adult reads approximately 300 words per minute in normal conditions [62]. However, several studies have highlighted the importance of contrast ratio on visual performance [35, 59, 63]. For instance, Wang et al. [59] report up to a 38% difference in visual performance depending on the contrast ratio. From linear interpolation of the results of that study, Gremlin defines a function, $M_{contrast}$, that specifies how much the contrast ratio affects reading speed. Thus, for each screen the visual demand in seconds is: $(word\_count/5) * M_{contrast}(contrast\_ratio)$.

For each button pressed during the interaction, Gremlin estimates the visual demand required to locate and press the correct region of the touchscreen. We use the results of a study by Sun et al. that measures the reaction time of users touching one of a set of buttons with different sizes, spacing, and contents [54]. The dominant factor is button size; e.g., users require 1.7 seconds to touch a 20x20 button but only 1.2 seconds to touch a 50x50 button.

Gremlin sums these elements to estimate the visual demand of a screen. For instance, reading a screen with five words with contrast ratio of 5:1 and touching a 40x40 button requires 1.9 seconds of visual attention. Gremlin applies this analysis to each unique screen. It sets $V_{completion\_time}$ to the maximum of the demand values of all screens that comprise an interaction. The reason we calculate each screen individually is that drivers can return their eyes to the road between screens (as long as there are no timeouts or other time-based elements forbidden by vehicle UI guidelines [33]). The reason we use the maximum value is that Gremlin should not begin an interaction if the driver does not have sufficient attention available to finish it.

3.3.2 ***Computing cognitive attention demand.*** Cognitive demand, represented as $C_{completion\_time}$, is the time that the driver needs to understand and respond to an interaction. Unlike visual attention, this value is difficult to quantify directly from low-level elements. Instead, Gremlin uses studies that indirectly assess cognitive load by measuring the peripheral detection task (PDT) response time [20, 28, 47]. PDT response time is measured by initiating a signal in the peripheral vision of the driver every 3–5 seconds. PDT studies measure how long it takes the driver to notice the signal. Intuitively, this could correspond to the time needed to apply the brakes in an emergency. Higher cognitive workload results in higher PDT response times. For instance, one study [52] measured average PDT response time to be 900 ms when talking to a passenger as compared to 700 ms in a baseline case without any secondary activities. Based on these results, Gremlin would consider $C_{completion\_time}$ for talking to a passenger to be 200 ms.

Gremlin uses the results of multiple studies [37, 46, 47, 52, 53] that measure PDT response time while the driver performs secondary activities (e.g., giving a voice-based command, listening to music, etc.). Each study reports a slightly different baseline PDT response time for driving without any secondary activities. Thus, Gremlin considers $C_{completion\_time}$ of the activity to be the delta over baseline reported in each individual study. If multiple studies measure the same activity, Gremlin uses the average across all such studies. Gremlin omits visual activities from its computation of cognitive attention since their demand is already captured by $V_{completion\_time}$.

For audio-only interactions, Gremlin currently categorizes the interaction as belonging to one of four categories: listening only (no speech), listening only (with speech), verbal commands only, and listening with verbal responses. It determines whether or not the recording contains speech by performing speech recognition on the captured audio using PocketSphinx [8]. If the recording is found to contain speech, Gremlin determines the complexity of the speech by computing the Flesch reading-ease score (FRES) [32] on the recognized utterance. This allows Gremlin to refine its estimate of cognitive demand by leveraging results from studies that differentiate PDT response time according to the complexity of the verbal interaction [46, 52]. Comparing results in these studies shows that the complexity of the speech can increase cognitive load by up to 48%. If an interaction consists of multiple activities, Gremlin uses the maximum demand of any such activity as $C_{completion\_time}$.

3.3.3 ***Generating a model.*** After the offloaded analysis calculates $V_{completion\_time}$ and $C_{completion\_time}$ for each interaction, Gremlin deletes the recordings and incorporates the calculated values into its model. Gremlin maintains distributions of $V_{completion\_time}$ and $C_{completion\_time}$ for each unique interaction type it has recorded, indexed by the type's $\langle application, id \rangle$ tuple. Some interaction types (e.g., displaying a pop-up warning) have the same content each time and, thus, have distributions with very similar values. Other interactions may have different branches the user may take (e.g., a hierarchy of dialogs) or vary widely in content (e.g., reading text messages). In such cases, the distributions will have significantly different values. We wish to handle variance conservatively. Thus, Gremlin estimates visual and cognitive demand by calculating the 95% confidence interval; it sets $V_{completion\_time}$ and $C_{completion\_time}$ to the upper interval value.

3.4 Scheduling interactions

We first describe how Gremlin continuously updates its estimates of the driver's currently available visual and cognitive attention. We then show how these value are combined with demand models to schedule new interactions.

3.4.1 ***Determining available visual attention.*** Gremlin defines $V_{deadline}$ as time that drivers can safely take their eyes off the road to perform a secondary task. Gremlin estimates $V_{deadline}$ based on data from controlled studies that measure how long a driver's vision can be safely occluded in different driving conditions. In this paper, we present a general method for calculating $V_{deadline}$ from such studies, and we develop an implementation that currently considers three important factors: vehicle speed, road curvature, and lane width. All factors are easily obtainable from in-vehicle or cloud sources: speed can be read from the vehicle's OBD2 port, while curvature and lane width can be obtained by combining GPS location with information from OpenStreetMap [44] or an equivalent database.

Occlusion testing is a standard methodology for measuring the effect of road conditions on visual demand. Such studies [9, 50, 58] simulate various driving conditions by outfitting drivers with occlusion glasses that prevent them from viewing the road. During the study, drivers activate a switch to receive a clear view for 0.5 seconds. By measuring the time between activations, the study determines how long drivers can look away from the road in the particular driving conditions being simulated. This value is precisely the $V_{deadline}$ that Gremlin needs to determine.

The studies we consider measure the effect of only a single driving condition at a time. However, they all use a common metric: the duration that a driver's view can be occluded after 0.5 seconds of a clear view. Thus, we combine the results of studies that measure different factors by considering the relative impact each factor has on the common metric. More specifically, Gremlin currently calculates:

$$V_{deadline} = V_{deadline}(speed) * Pe(r) * Pe(w) \quad \text{(Pe(r)=Curve radius penalty, Pe(w)=Lane width penalty)} \quad (1)$$

$V_{deadline}(speed)$ gives available visual attention based solely on the speed of the vehicle. We generate this function by performing a linear regression over the study results of Senders et al. [50]. Linear regression gives a good fit (R^2 coefficient of determination=0.96), Given speed expressed in MPH, this yields:

$$V_{deadline}(speed) = 4.056 - 0.041 * speed \quad (2)$$

Gremlin calculates the relative effect of road curvature, $Pe(r)$, from the results of Tsimhoni et al. [58]:

$$Pe(r) = \frac{\frac{0.5}{0.252+34.5*1/r}}{1.984} \quad \text{(r=radius of curve in meters)} \quad (3)$$

Based on the above study, $Pe(w)$ decreases $V_{deadline}$ by 2% as the lane width, w, decreases by each foot from the standard width of 12 feet.

Note that while Gremlin currently considers many important factors in estimated visual demand, there are more factors that we would ideally add to our model such as frequency of intersections and street directionality (one-way vs. two-way). We currently do not use these factors because we lack the controlled studies that measure them in isolation. However, our methodology makes it easy to add such data as new studies are performed.

3.4.2 ***Determining available cognitive attention.*** Gremlin defines $C_{deadline}$ as the time that the driver can safely spare for a cognitive task such as voice-based interactions. It currently estimates this value based on surrounding traffic volume and the driver's experience level. Traffic volume is already obtained from the cloud by turn-by-turn direction applications. A driver's experience must be set manually, but it changes extremely rarely.

Studies that assess the impact of driving conditions on cognitive attention commonly use PDT response time as a figure of merit. Unfortunately, these studies only measure how response time changes as conditions vary; they do not specify how higher response times negatively impact driving safety. Gremlin must determine how much cognitive attention is available for secondary tasks. This means that it must (1) understand how secondary tasks affect PDT response time, and (2) set a threshold value beyond which higher PDT response times are unacceptable for safe driving.

We first consider how to set the threshold. One method would be to ask usability experts to choose a value. We believe a better approach is to use actual road safety data.

For this purpose, we use a large-scale study of actual driving that recorded crashes and near crashes (CNC) and correlated them with secondary tasks being performed by the drivers of the vehicles [61]. The study calculates an *odds ratio* for each secondary task that estimates the likelihood of a CNC occurring while performing the task in comparison to driving without performing that task. An odds ratio greater than 1 indicates that the task has a positive correlation with CNC occurrences; e.g., reading has an odds ratio of 1.34.

In order to use this data, we found secondary tasks reported in the study that match well with tasks in existing PDT studies [43, 52, 56]; e.g., changing the radio station and talking on a cell phone. This yields a set of $\langle odds\_ratio, response\_time \rangle$ tuples. By performing a linear regression on these tuples (R^2 coefficient of determination=0.88), we find that a PDT response time of 940 ms corresponds to an odds ratio of 1.0; in other words, any combination of activities (e.g., driving plus performing a secondary task) with a PDT

	Novice driver	Experienced driver
Light traffic	73 ms	274 ms
Medium traffic	0 ms	253 ms
Heavy traffic	0 ms	152 ms

Table 2: $C_{deadline}$ values based on the driver's experience and traffic conditions.

response time higher than 940 ms is projected to increase the base probability of crashes and near crashes. To isolate the effect of the secondary task, we subtract the average response time measured across all studies for driving without performing a secondary task. This results in $C_{deadline}$ (the secondary task threshold) of 274 ms; any secondary task that adds more than this value to PDT response time should not be allowed while driving.

The above threshold represents a best-case scenario because the baseline is an experienced driver operating the vehicle in light traffic conditions. Patten et al. [47] show that PDT response times increase for less-experienced drivers and under heavier traffic conditions. As shown in Table 2, Gremlin uses the experience definition of Patten et al. and refines its $C_{deadline}$ values to reflect the delta in response times measured in this study. As might be expected, novice drivers have no attention to spare for any secondary task under medium and heavy traffic conditions and almost none under light conditions. Even experienced drivers have much less attention to spare in heavy traffic.

Gremlin further considers how specific driving situations impact $C_{deadline}$ based on a study by Martens et al [37]. For instance, driving around a sharp curve decreases $C_{deadline}$ by 113 ms and stopping at a stop sign decreases $C_{deadline}$ by 184 ms. Both situations can be determined by combining vehicle location and road data. Interestingly, these results indicate that interactions should rarely be initiated while the vehicle is at a stop sign, presumably because considerable attention is needed to determine when to enter the intersection. This contradicts recent proposals and deployed systems that allow interactions when the vehicle is not moving [49].

3.4.3 ***Allowing or deferring interactions.*** Unmodified applications attempt to initiate an interaction by calling the `notify()` method of Android's `NotificationManager`. If an application uses its own custom method for notifications, it must be modified to call `gremlin_begin_notify()` as shown in Table 1.

Gremlin maintains a white list of interaction types ($\langle application, id \rangle$ tuples) that are high priority. Updating this list is a privileged operation; we envision that the list is set by the vehicle manufacturer and maintained via software updates. Interactions on this list are considered to be critical and are delivered immediately.

Otherwise, Gremlin retrieves $C_{completion\_time}$ and $V_{completion\_time}$ for the interaction type from the models generated offline. If it does not have a model yet for a new interaction type, it applies the default HMI scheduling policy. Gremlin updates $C_{deadline}$ and $V_{deadline}$ continuously when information about vehicle location, traffic conditions, etc. changes. Updating the values has low overhead, requiring only a table lookup.

Next, Gremlin allows the interaction to proceed if $V_{completion\_time} < V_{deadline}$ and $C_{completion\_time} < C_{deadline}$. For unmodified applications, $NotificationManager$ simply delivers the requested notification. For custom applications, `gremlin_begin_notify()` returns 1 (success), informing the application that it should initiate its interaction.

If Gremlin decides to defer the notification, it continually checks if $V_{completion\_time} < V_{deadline}$ and $C_{completion\_time} < C_{deadline}$ as both $V_{deadline}$ and $C_{deadline}$ change. The notify methods block until this condition is satisfied. Afterward, the notification is initiated as above. An application whose interaction has been deferred may cancel its request by either invoking the appropriate method on the `NotificationManager` or calling `gremlin_cancel_notify()`. Calling the latter method causes `gremlin_begin_notify()` to return 0, informing the application that it should not perform the interaction.

Note that Gremlin can only enforce its scheduling decisions for applications that use the `NotificationManager` API; other applications must voluntarily decide to use and respect the results of calling the Gremlin API. To enforce behavior for all applications, vehicle manufacturers could potentially require all installed applications to interact with the driver through a standard service such as `NotificationManager`.

4 EVALUATION

Our evaluation answers the following questions:

- How well do Gremlin's scheduling decisions match those of a vehicular interface expert across a range of interaction types and driving scenarios?
- What are the computation and bandwidth requirements of offloaded analysis?
- How much overhead does Gremlin add?

4.1 Setup

We used a Nexus 9 tablet running Android 5.0.1 with Gremlin to run vehicular applications and emulate a vehicular HMI touchscreen. We selected the seven representative interactions shown in Table 3 for our evaluation. These include a mixture of vehicular and mobile applications, high-priority and low-priority interactions, audio-only and visual interactions, and simple and complex interactions.

We first used Gremlin to record five different interactions for each of the seven types. We used simulated driving traces to generate the interactions; for instance, the traces emulate changes in location for a moving vehicle that cause location-based alerts to pop-up and changes in speed that cause excessive speed warnings to be displayed. We explored a range of user actions for each interaction type by varying messages, locations, reminders, etc. for each observation. The last two columns in Table 3 show Gremlin's results for analyzing the recorded interactions. $V_{completion\_time}$ is visual attention demand, and $C_{completion\_time}$ is cognitive attention demand.

We evaluated each interaction in the three driving scenarios shown in Table 4; the scenarios differ substantially in attention demand. The last two columns of the table show the values Gremlin calculated for $V_{deadline}$ and $C_{deadline}$.

Interaction	Priority	Description	$V_{completion\_time}$	$C_{completion\_time}$
Waze-Hazard	High	Waze alerts the driver about an upcoming road hazard with an audio message and a visual pop-up.	1787 ms	66 ms
Waze-Camera	Low	Waze alerts the driver about an upcoming speed camera with an audio tone and a visual pop-up.	1136 ms	4 ms
Hangouts-Visual	Low	Google Hangouts alerts the driver about a new message with an audio tone. The driver selects the message, displays it, and responds.	18645 ms	4 ms
Hangouts-Voice	Low	Same as above, but all interactions are voice-based.	0 ms	212 ms
GeoTask	Low	GeoTask plays an audio tone and displays a pre-set reminder when the driver passes by a designated location.	2227 ms	4 ms
Speed Cameras	High	Speed Cameras warns the driver with an audio "speed limit exceeded" message. The screen shows the driver's current speed and the speed limit.	706 ms	66 ms
Foursquare	Low	Foursquare produces an audio tone and a notification to show nearby restaurants when the driver enters a new city. The driver clicks on the notification to display the list of restaurants.	22773 ms	4 ms

Table 3: Interactions used to evaluate Gremlin. For each interaction, the table shows the $V_{completion\_time}$ and $C_{completion\_time}$ values calculated by Gremlin over 5 observations.

Driving scenario	Description	$V_{deadline}$	$C_{deadline}$
Low attention demand	Experienced driver traveling at 20 MPH on a straight rural road with standard lane width and light traffic.	3243 ms	274 ms
Medium attention demand	Experienced driver traveling at 35 MPH on a slightly curving road (curve radius of 592 meters) with 11 foot lane width and heavy traffic.	2229 ms	152 ms
High attention demand	Novice driver traveling at 50 MPH on a sharply curving road (curve radius of 60 meters) with standard lane width and light traffic.	615 ms	73 ms

Table 4: Driving scenarios used to evaluate Gremlin. For each scenario, the table shows the $V_{deadline}$ and $C_{deadline}$ values calculated by Gremlin.

4.2 Decision quality

We evaluated Gremlin's scheduling decisions by comparing them to the decisions recommended by a vehicular interface expert from the University of Michigan Transportation Research Institute. The expert we recruited is not a member of our project team and was not aware of our work prior to this study.

We first explained the three driving scenarios, and we showed the expert a typical interaction for each of the seven types in Table 3 (the first recorded interaction in each case). We asked the expert to decide whether or not he would allow each interaction in each of the three driving scenarios. We did not share Gremlin's assessments until after the expert completed this task.

Table 5 compares Gremlin's decisions with those of the expert. A check indicates a decision to allow the interaction, and an x indicates a decision to disallow or defer the interaction. Overall, Gremlin's decisions matched those of the expert in 19 out of 21 cases (90.4%).

In the opinion of the expert, Gremlin had one false positive in which it allowed an interaction that it should not have allowed. Gremlin allowed the Waze-Camera interaction during the medium demand scenario. The expert felt that this interaction should have been disallowed since the driver must look at the screen to understand the reason for the audio output tone; the tone itself conveys no useful information.

In the opinion of the expert, Gremlin also had one false negative. The expert felt that the GeoTask interaction should be allowed in all scenarios, including the high-demand scenario in which Gremlin disallowed it, because the driver expects this output to occur at the given location. Since the driver has previously set this reminder, the activation should not come as a surprise. Gremlin lacks the context to make this type of inference.

Stepping back from direct comparisons, this study also confirms Gremlin's general approach of initiating interactions based on current driving conditions. For three of the seven interactions, the expert allowed the interaction in the low demand scenario and disallowed the same interaction in the high demand scenario. Simply denying all interactions would match the expert's opinion in only 10 of the 21 cases.

Many current vehicle HMIs allow voiced-based interactions and disallow any interactions that use the touchscreen when the vehicle is moving [16, 38, 57]. Both our results and the expert's opinions show that this approach works poorly. This default policy would

	Low demand		Medium demand		High demand	
Interaction	Gremlin	Expert	Gremlin	Expert	Gremlin	Expert
Waze-Hazard	✓	✓	✓	✓	✓	✓
Waze-Camera	✓	✓	✓	×	×	×
Hangouts-Visual	×	×	×	×	×	×
Hangouts-Voice	✓	✓	×	×	×	×
GeoTask	✓	✓	✓	✓	×	✓
Speed Cameras	✓	✓	✓	✓	✓	✓
Foursquare	×	×	×	×	×	×

Table 5: This table compares Gremlin's scheduling decisions with those of a vehicular UI expert. We show results for each combination of interaction type and driving scenario. A × mark indicates a decision to allow the interaction, and a ✓ mark indicates a decision to defer/disallow the interaction. Cases where Gremlin and the expert disagreed are highlighted in gray.

Interaction	Analysis time (seconds)	Unique video frames	Audio I/Os	Compressed size (MB)
Waze-Hazard	15.6	13.2	1	29.1
Waze-Camera	27.2	24.6	1	27.9
Hangouts-Visual	16.2	20.2	1	7.3
Hangouts-Voice	12.6	0	8	13.0
GeoTask	4.8	2.8	1	4.5
Speed Cameras	4.6	2.8	1.4	12.4
Foursquare	8.4	7.6	1	12.3

Table 6: The second column shows the average time to analyze each type of interaction. The next two columns show the average number of unique video frames and audio I/Os per recorded interaction. The final column shows the average size of the compressed recordings.

Application	Baseline	Recording	Overhead
Google Hangouts	20 ms	24 ms	4 ms
Waze	122 ms	148 ms	26 ms

Table 7: Overhead of Gremlin recording interactions as measured by GUI response time for two applications

match the expert opinion in only 9 of the 21 cases. For the voiced-based Google Hangouts interaction, the expert and Gremlin both find that the interaction should be disallowed in the medium demand and high demand scenarios, meaning that the driver is likely to be cognitively overloaded by the default policy in many driving conditions. On the other hand, both the expert and Gremlin identify many visual interactions that should be allowed while the vehicle is moving, either because they are high-priority such as road hazard alerts, or because they do not demand much attention, e.g., GeoTask reminders.

We also asked the expert for qualitative feedback about Gremlin. The expert agreed with Gremlin's approach of managing driver's attention but felt strongly that, in order for Gremlin to be practical, it should consider more factors, such as the driver's familiarity with the current road, the familiarity with the interaction, street conditions, and weather. We agree with assessment and view Gremlin as a prototype that can be extended to consider such factors.

4.3 Offloaded analysis

We measured both the computational and network demands of offloading analysis to edge infrastructure. We used a workstation with 4 3.1 GHz Xeon E5-2687W cores and 16 GB of RAM to represent a typical edge compute node. The second column in Table 6 shows the average time required to analyze each type of recorded interaction on this platform. The third and forth columns show, for each interaction type, the average number of unique video frames and audio I/Os analyzed.

Gremlin takes an average of 12.7 seconds to analyze an interaction on a workstation, but there is considerable variation across different types. For each second of recorded interaction, the analyzer takes approximately 1.1 seconds to perform visual analysis and 1.6 seconds to analyze audio. These results demonstrate the benefit of offloading analysis of recorded interactions; this workload is simply too compute-intensive to run on a typical mobile computer.

The final column in Table 6 shows the average size of the compressed recordings for each interaction type. The average size varies from 4.5 MB (GeoTask) to 29.1 MB (Waze-Hazard), with an average of 15.2 MB across all recorded interactions. At this size, recording 2-3 interactions per day and analyzing them in the cloud could consume approximately 1 GB per month of wide-area bandwidth. By offloading to the edge rather than the cloud, this data need only be sent over local-area network connections, potentially providing a considerable cost savings.

4.4 Overhead

We next measured Gremlin's overhead by comparing interactive application performance when Gremlin is recording with performance when Gremlin is not running. For this experiment, we replay touch events using RERAN [17] and measure the response time for each event in an interaction as the time between the user clicking on the screen and the application rendering the resulting UI elements. We measured Waze and Google Hangouts for this experiment. Google Hangout is highly interactive but does not contain complex graphical elements. Waze is one of the least interactive of our applications, but it has some of the most complex graphical elements.

Table 7 shows that Gremlin recording adds very little performance overhead. This overhead occurs rarely—only during application-initiated interactions. GUI response time for Google Hangouts and Waze is slowed by 4 ms and 26 ms, respectively. These values are well below the commonly-cited 50 ms threshold at which users perceive differences in GUI response times [15]. The overhead for Waze is larger due to an increase in the size of the captured video; larger portions of the screen are changed more often in this interaction.

We also attempted to measure the overhead of Gremlin scheduling on application performance. However, the overhead was not measurable within experimental error.

5 DISCUSSION

Gremlin is a promising prototype. Our evaluation shows that it makes reasonable decisions and imposes only a small performance overhead on the vehicular HMI. However, there are many challenges that need to be addressed before a Gremlin-like system can be deployed in vehicles.

As noted in the evaluation, the attention model needs to be expanded. We consider many important factors in assessing available attention and attention demand, but there are several other factors we would like to include. For assessing attention demand, we would like to include graphical UI elements and other controls beyond touchscreen buttons (e.g., haptic steering wheel controls). For assessing available attention, we would like to include factors such as the time of day, road familiarity, fatigue or other physical conditions of the driver, and how familiar a driver is with a particular interaction. Detection of roadside hazards and pedestrians could also inform scheduling decisions. We provide a methodology for incorporating standard PDT and road studies into Gremlin's model. However, such studies would still need to be performed in order to generate the model parameters.

Gremlin assumes that reducing the amount of data sent over wide-area networks between the edge and the cloud will result in significant cost savings. However, specific costs models will only become clear once significant compute capacity is deployed at the edge. Currently, this motivation holds in restricted edge deployments. For instance, when the phone stores recordings, then offloads computation to a server co-located with a home access point, no data is used for the back-haul connection to the cloud or for a cellular plan. We believe it is reasonable to think that edge infrastructure could evolve to include similar network/compute co-location at roadside and related locations.

Gremlin currently relies on a privileged user such as the vehicle manufacturer to specify which interactions are high priority, and it does not have a mechanism to enforce its decisions for all applications. Current vehicle HMIs must already deal with these issues, so industry standards for deciding which applications can be installed and enforcing interface restrictions would help both current HMIs and Gremlin.

Our comparison of Gremlin's decisions with expert opinion is an end-to-end check that Gremlin is making reasonable decisions. However, different experts could express different opinions. Ultimately, extensive user testing is required to deploy a system like Gremlin in a vehicle.

6 RELATED WORK

It has long been recognized that user attention is limited in mobile computing. Consequently, there has been much past work to understand the impact of notifications and to schedule when such interruptions occur. Gremlin differs significantly from this past work in that it attempts to schedule not just a notification, but rather the entire interaction that is initiated by that notification. This leads to its unique approach of quantifying attention demand and building models to predict the demand of future interactions, then comparing these predictions to dynamic estimates of attention supply.

Many researchers have studied the detrimental effect of poorly-timed notifications in desktop environments [4, 10, 26]. Subsequent work [27] has found that importance matters: users are willing to tolerate some disruption in return for receiving valuable notifications. Although these studies target a different computing environment, they support Gremlin's decision to schedule based on priority, as well as on attention supply and demand.

Other researchers have attempted to determine the best time to interrupt users to deliver notifications [1, 22, 23, 25]. For instance, PRIORITIES [23] is a desktop e-mail notification system that uses a Bayesian model to infer the user's available attention level and compute the expected cost of interruption and deferring alerts. Instead of inferring user activity, Gremlin observes it directly using vehicle sensors. Gremlin also employs controlled user studies and road data to quantify how sensor data correlates with attention supply.

Kim et al. [31] use body-worn sensors and vehicular data to determine the opportune moment to interrupt the driver. Their work assumes that the drivers are the best judge of when they should be interrupted, and so learns from their past behavior. In contrast, Gremlin quantifies attention demand and supply from the sensor data and observations of past interactions. Unlike the work of Kim et al., Gremlin considers the entire interaction that follows the interruption in its scheduling decision, rather than just the interruption itself. Other recent studies have focused on measuring changes in driver's workload based on measurable factors, such as pupil dilation [48], heart-rate variability [45], and driver movement changes on a seat [7]. However, these studies do not quantify available attention or match it to the demand of secondary tasks as Gremlin does.

Similarly, recent research attempts to determine proper task break points for mobile devices in non-vehicular settings. Fischer

et al. [14] determine the end of mobile device interactions to deliver notifications at such instances. Okoshi et al. [42] determine accurate application-specific break points, during which the user can be interrupted while using an application. Ho et al. [21] determine when the user is transitioning from one physical activity to another. Kern et al. [30] sense the user's context to use socially acceptable notifications modalities.

AMC [33] checks vehicular applications to determine whether they conform to best practice guidelines. AMC performs static checking of screens and task lengths, whereas Gremlin schedules dynamic behavior (i.e., interactions). AMC determines only whether an interface is acceptable or not in standard driving conditions, whereas Gremlin considers the changing availability of the driver's attention.

Green [18] originally pointed out the need for a workload manager that regulates the flow of information to drivers. In a workshop paper [34], we previously made the case for managing user attention in the operating system. Although our design agrees with the general approach in these papers, neither of these proposals included a specific implementation such as Gremlin.

7 CONCLUSION

We have described a methodology for scheduling interactions initiated by vehicular applications and a prototype, called Gremlin, that demonstrates this methodology. In the future, we hope to extend our work to other attention-limited situations, such as when a user is walking, working at the office, or engaging in social activities. Scheduling in these situations is more complex. In vehicular computing, Gremlin knows that the primary task of driving is very important. The complexity of this task has been measured in numerous studies. Further, the vehicle has a wide array of sensors that can assess the context in which the driver is operating. Other attention-limited situations have a wider range of primary activities with varying importance, and they have a comparative lack of sensors. We believe Gremlin is an important first step toward scheduling interactions in more general attention-limited situations, and we look forward to tackling the research challenges that remain.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their thoughtful comments. This work has been partially supported by the National Science Foundation under grant CNS-1717064 and by Ford Motor Company. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, Ford, or the University of Michigan.

REFERENCES

[1] Adamczyk, P. D., and Bailey, B. P. If not now, when?: The effects of interruption at different moments within task execution. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria, April 2004), CHI '04, pp. 271–278.

[2] Allport, D. A., Antonis, B., and Reynolds, P. On the division of attention: A disproof of the single channel hypothesis. *Quarterly Journal of Experimental Psychology 24*, 2 (1972), 225–235.

[3] Smartphone use behind the wheel survey. http://about.att.com/content/dam/snrdocs/2015%20It%20Can%20Wait%20Report_Smartphone%20Use%20Behind%20the%20Wheel%20.pdf.

[4] Bailey, B. P., and Konstan, J. A. On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. *Computers in Human Behavior 22*, 4 (2006), 685–708.

[5] Baumann, A., Peinado, M., and Hunt, G. Shielding applications from an untrusted cloud with haven. In *Proceedings of the 11th Symposium on Operating Systems Design and Implementation* (Broomfield, CO, October 2014).

[6] Bhardwaj, K., Shih, M.-W., Agarwal, P., Kim, T., and Schwan, K. Fast, scalable, and secure onloading of edge functions using airbox. In *Proceedings of the First IEEE/ACM Symposium on Edge Computing* (Washington, DC, October 2016).

[7] Braun, A., Frank, S., Majewski, M., and Wang, X. Capseat: Capacitive proximity sensing for automotive activity recognition. In *Proceedings of the 7th International Conference on Automotive User Interfaces and Interactive Vehicular Applications* (Nottingham, United Kingdom, 2015), pp. 225–232.

[8] CMU Sphinx. *PocketSphinx*. http://cmusphinx.sourceforge.net/.

[9] Courage, C., Milgram, P., and Smiley, A. An investigation of attentional demand in a simulated driving environment. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* (San Diego, CA, July 2000).

[10] Czerwinski, M., Horvitz, E., and Wilhite, S. A diary study of task switching and interruptions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vienna, Austria, April 2004), CHI '04, pp. 175–182.

[11] Driver Focus-Telematics Working Group. Statement of principles, criteria and verification procedures on driver interactions with advanced in-vehicle information and communication systems. Tech. rep., Alliance of Automobile Manufacturers, June 2003.

[12] Enck, W., Gilbert, P., gon Chun, B., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *Proceedings of the 9th Symposium on Operating Systems Design and Implementation* (Vancouver, BC, October 2010).

[13] Engstrom, J., Johansson, E., and Ostlund, J. Effects of visual and cognitive load in real and simulated motorway driving. *Transportation Research Part F: Traffic Psychology and Behaviour 8*, 2 (2005), 97–120.

[14] Fischer, J. E., Greenhalgh, C., and Benford, S. Investigating episodes of mobile phone activity as indicators of opportune moments to deliver notifications. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services* (Stockholm, Sweden, August 2011), pp. 181–190.

[15] Flautner, K., and Mudge, T. Vertigo: Automatic performance-setting for Linux. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, December 2002), pp. 105–116.

[16] General Motors LLC. 2015 Chevrolet MyLink details book. https://my.gm.com/.

[17] Gomez, L., Neamtiu, I., Azim, T., and Millstein, T. Reran: Timing- and touch-sensitive record and replay for android. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, May 2013), pp. 72–81.

[18] Green, P. Driver distraction, telematics design, and workload managers: Safety issues and solutions. In *SAE Convergence* (2004), Society of Automotive Engineers.

[19] Green, P., Levison, W., Paelke, G., and Serafin, C. Suggested human factors design guidelines for driver information systems. Tech. rep., The University of Michigan Transportation Research Institute (UMTRI), August 1994.

[20] Harms, L., and Patten, C. Peripheral detection as a measure of driver distraction. a study of memory-based versus system-based navigation in a built-up area. *Transportation Research Part F: Traffic Psychology and Behaviour 6*, 1 (2003), 23–36.

[21] Ho, J., and Intille, S. S. Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Portland, Oregon, April 2005), CHI '05, pp. 909–918.

[22] Horvitz, E., and Apacible, J. Learning and reasoning about interruption. In *Proceedings of the 5th International Conference on Multimodal Interfaces* (Vancouver, Canada, November 2003), ICMI '03, pp. 20–27.

[23] Horvitz, E., Jacobs, A., and Hovel, D. Attention-sensitive alerting. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence* (Stockholm, Sweden, July 1999), UAI'99, pp. 305–313.

[24] Hunt, T., Zhu, Z., Xu, Y., Peter, S., and Witchel, E. Ryoan: A distributed sandbox for untrusted computation on secret data. In *Proceedings of the 12th Symposium on Operating Systems Design and Implementation* (Savannah, GA, November 2016).

[25] Iqbal, S. T., and Bailey, B. P. Understanding and developing models for detecting and differentiating breakpoints during interactive tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, April 2007), CHI '07, pp. 697–706.

[26] Iqbal, S. T., and Horvitz, E. Disruption and recovery of computing tasks: Field study, analysis, and directions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (San Jose, California, April 2007), CHI '07, pp. 677–686.

[27] Iqbal, S. T., and Horvitz, E. Notifications and awareness: A field study of alert usage and preferences. In *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work* (Savannah, GA, February 2010), CSCW '10, pp. 27–30.

[28] Jahn, G., Oehme, A., Krems, J. F., and Gelau, C. Peripheral detection as a workload measure in driving: Effects of traffic complexity and route guidance

system use in a driving study. *Transportation Research Part F: Traffic Psychology and Behaviour 8*, 3 (2005), 255–275.

[29] Jamson, A. H., and Merat, N. Surrogate in-vehicle information systems and driver behaviour: Effects of visual and cognitive load in simulated rural driving. *Transportation Research Part F: Traffic Psychology and Behaviour 8*, 2 (2005), 79–96.

[30] Kern, N., and Schiele, B. Context-aware notification for wearable computing. In *Proceedings of the 7th IEEE International Symposium on Wearable Computers* (Washington, DC, October 2003), pp. 223–230.

[31] Kim, S., Chun, J., and Dey, A. K. Sensors know when to interrupt you in the car: Detecting driver interruptibility through monitoring of peripheral interactions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)* (Seoul, Republic of Korea, April 2015), pp. 487–496.

[32] Kincaid, J. P., Fishburne Jr, R. P., Rogers, R. L., and Chissom, B. S. Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Tech. rep., DTIC Document, Feb 1975.

[33] Lee, K., Flinn, J., Giuli, T., Noble, B., and Peplin, C. AMC: Verifying user interface properties for vehicular applications. In *Proceedings of the 11th International Conference on Mobile Systems, Applications and Services* (Taipei, Taiwan, June 2013), pp. 1–12.

[34] Lee, K., Flinn, J., and Noble, B. The case for operating system management of user attention. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile)* (Santa Fe, NM, February 2015).

[35] Lin, C.-C. Effects of contrast ratio and text color on visual performance with tft-lcd. *International Journal of Industrial Ergonomics 31*, 2 (2003), 65–72.

[36] Liu, P., Willis, D., and Banerjee, S. Paradrop: Enabling lightweight multi-tenancy at the network's extreme edge. In *Proceedings of the First IEEE/ACM Symposium on Edge Computing* (Washington, DC, October 2016).

[37] Martens, M., and van Winsum, W. Measuring distraction: the peripheral detection task. Tech. rep., National Highway Traffic Safety Administration, Jun 2000.

[38] Mazda Motor Company. MZD Connect. http://infotainment.mazdahandsfree.com/communication-sms?language=en-RW.

[39] Nakamura, Y. JAMA guideline for in-vehicle display systems. Tech. rep., Japan Automobile Manufacturers Association, Oct 2008.

[40] Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles* (Saint-Malo, France, October 1997), pp. 276–287.

[41] of Automobile Manufacturers, A. Comments received from the alliance of automobile manufacturers. Accessed at www.regulations.gov, Docket NHTSA-2010-0053, Document Number 0104.

[42] Okoshi, T., Tokuda, H., and Nakazawa, J. Attelia: Sensing user's attention status on smart phones. In *16th International Conference on Ubiquitous Computing* (Seattle, Washington, September 2014), pp. 139–142.

[43] Olsson, S., and Burns, P. Measuring driver visual distraction with a peripheral detection task. Tech. rep., National Highway Traffic Safety Administration, Feb 2008.

[44] https://www.openstreetmap.org.

[45] Park, J. W., Kim, S., and Dey, A. Integrated driving aware system in the real-world: Sensing, computing and feedback. In *Proceedings of the 2016 CHI Conference: Extended Abstracts on Human Factors in Computing Systems* (Santa Clara, CA, 2016), pp. 1591–1597.

[46] Patten, C. J., Kircher, A., Ostlund, J., and Nilsson, L. Using mobile telephones: cognitive workload and attention resource allocation. *Accident Analysis and Prevention 36*, 3 (2004), 341–350.

[47] Patten, C. J., Kircher, A., Ostlund, J., Nilsson, L., and Ola, S. Driver experience and cognitive workload in different traffic environments. *Accident Analysis and Prevention 38*, 5 (2006), 887–894.

[48] Rajan, R., Selker, T., and Lane, I. Task load estimation and mediation using psycho-physiological measures. In *Proceedings of the 21st International Conference on Intelligent User Interfaces* (Sonoma, CA, 2016), pp. 48–59.

[49] Richtel, M. Phone makers could cut off drivers. so why don't they? *The New York Times* (2016).

[50] Senders, J., Kristofferson, A., Levison, W., Dietrich, C., and J.L., W. The attentional demand of automobile driving. *Highway Research Record*, 195 (1967), 15–33.

[51] Stevens, A., Quimby, A., Board, A., Kersloot, T., and Burns, P. Design guidelines for safety of in-vehicle information systems. Tech. rep., Transport Research Laboratory, Feb 2002.

[52] Strayer, D. L., Cooper, J. M., Turrill, J., Coleman, J., MedeirosWard, N., and Biondi, F. Measuring cognitive distraction in the automobile. Tech. rep., AAA Foundation for Traffic Safety, Jun 2013.

[53] Strayer, D. L., Turrill, J., Coleman, J. R., Ortiz, E. V., and Cooper, J. M. Measuring cognitive distraction in the automobile ii: Assessing in-vehicle voice-based interactive technologies. Tech. rep., AAA Foundation for Traffic Safety, Oct 2014.

[54] Sun, X., Plocher, T., and Qu, W. An empirical study on the smallest comfortable button/icon size on touch screen. In *Proceedings of the 2nd International Conference on Usability and Internationalizationi (UI-HCII)* (Beijing, China, July 2007), pp. 615–621.

[55] Task Force HMI. European statement of principles on human machine interface for in-vehicle information and communication systems. Tech. rep., Commission of the European Communities, December 1998.

[56] Tornros, J. E., and Bolling, A. K. Mobile phone useâĂŤeffects of handheld and handsfree phones on driving performance. *Accident Analysis & Prevention 37*, 5 (2005), 902–909.

[57] Toyota Motor Company. Entune system quick reference guide. http://www.toyota.com/t3Portal/document/om-s/OM16QTQRG/pdf/OM16QTQRG.pdf.

[58] Tsimhoni, O., Yoo, H., and Green, P. Effects of visual demand and in-vehicle task complexity on driving and task performance as assessed by visual occlusion. Tech. rep., The University of Michigan Transportation Research Institute (UMTRI), December 1999.

[59] Wang, A.-H., and Chen, M.-T. Effects of polarity and luminance contrast on visual performance and vdt display quality. *International Journal of Industrial Ergonomics 25*, 4 (2000), 415–421.

[60] Wickens, C. D. Processing resources and attention. *Multiple-task performance* (1991), 3–34.

[61] Young, R. Revised odds ratio estimates of secondary tasks: A re-analysis of the 100-car naturalistic driving study data. Tech. rep., SAE Technical Paper, Jan 2015.

[62] Ziefle, M. Effects of display resolution on visual performance. *Human Factors: The Journal of the Human Factors and Ergonomics Society 40*, 4 (1998), 554–568.

[63] Zuffi, S., Brambilla, C., Beretta, G., and Scala, P. Human computer interaction: Legibility and contrast. In *14th International Conference on Image Analysis and Processing (ICIAP)* (Modena, Italy, September 2007), pp. 241–246.