# First Experience with CarTorrent in a Real Vehicular Ad Hoc Network Testbed

Kevin C. Lee, Seung-Hoon Lee, Ryan Cheung, Uichin Lee, Mario Gerla
University of California, Los Angeles
4732 Boelter Hall
Los Angeles , CA 90095
{kclee,shlee,uclee,gerla}@cs.ucla.edu
rcheung2@ucla.edu

## ABSTRACT

Content sharing using cooperative peer-to-peer model has become increasingly more popular in a vehicular ad hoc network (VANET). The small transmission window from a vehicle to an access point (AP), high mobility of vehicles, and intermittent and short-lived connectivity to an AP provide incentives for vehicles to cooperate with one another to obtain information from the Internet. These characteristics of VANETs naturally stipulate the use of cooperative peer-to-peer paradigm and motivate related content sharing application such as CarTorrent.

Building upon previous research on SPAWN[6, 1], we have implemented CarTorrent and deployed it on a real VANET. We have run extensive field tests to affirm the feasibility of the peer-to-peer file sharing application tailored to VANET. To the best of our knowledge, the deployment of such a content sharing application on a real vehicular ad hoc testbed is the first of its kind.

## Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

## General Terms

Design, Experimentation, Measurement

## Keywords

Data Mule, BitTorrent

## 1. INTRODUCTION

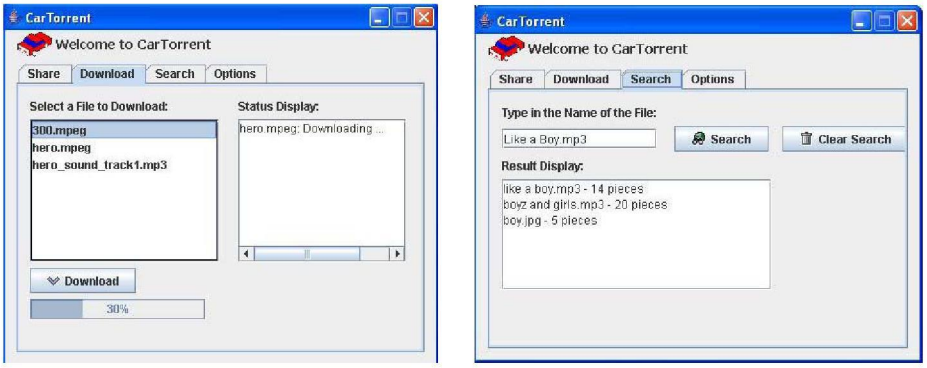
Navigation safety requirements have propelled the development and deployment of VANET. Beyond the safety navigation, new types of applications have emerged such as office-on-wheels and on-car entertainment. Among these, file sharing is gaining its momentum: People want to download not only music and movie trailers while driving, but also location-cognizant data such as virtual hotel tour clips.

People can download files from road-side access points (APs) that provide Internet connections, which is known as Wardriving [8]. The conventional client-server model will not work neither scale well for the following reasons. First, due to the high mobility, the actual contact time to an AP is short. For example, assuming that the WiFi range is 300m, when driving at the speed of 45mph, we can have 30 seconds of contact period. With the overhead of association, DHCP, and Internet connections, the actual contact period is shorter than 30 seconds. Second, in real environments signal strength is mainly a function of distance; i.e., as the distance from the AP increases, the signal strength decreases. This increases the packet error rate; consequently, the effective throughput that one can achieve is much less than expected. Third, it is neither practical to install APs every 300 meters, nor feasible to stop in the middle of roads to download a file. Thus, we conclude that in reality, the contact period is short, and its goodput is low. To effectively handle this situation, we advocate the use of peer-to-peer file swarming in which users out of AP range can still download parts of files from others.

In P2P file swarming such as BitTorrent, a file is divided into the same size pieces, and peers with fractions of a file can exchange whatever pieces available by forming an overlay network. This not only sheds the load of the server, but also increases the availability of pieces, thus expediting the downloading process. However, BitTorrent cannot be directly ported to wireless environments because of the discrepancy between a logical overlay topology and a physical topology of mobile nodes. For instance, a peer who is one hop away in a logical overlay could be located five hops away physically. To maximize the available wireless resources by localizing traffic (i.e., which increases the spatial diversity and reduces the routing overheads), researchers thus far have focused on mapping the logical overlay to physical topology [3, 7, 1]. In particular, SPAWN uses the proximity-driven piece selection strategy, thus further reducing the average hop count of multi-hop pulling. It is known that the proximity-driven piece selection outperforms the conventional "rarest first" piece selection.

In this paper, we propose CarTorrent, a BitTorrent-style file swarming protocol in the vehicular environment, by extending SPAWN [1]. For a given file, CarTorrent clients disseminate their piece availability information via gossip-

978-1-4244-1690-5/07/$25.00 ©2007 IEEE.




(a) File Downloading (b) File Searching

**Figure 1: CarTorrent GUI**

ing (i.e., by $k$-hop limited scope broadcasting). Each gossip message is forwarded until it reaches to nodes located $k$-hop away from the originator. Thus, peers can gather statistics such as local topology and piece availability. Statistics are then used to select a piece/peer that is preferably close in proximity. In other words, given that two peers $A$ and $B$ own a rarest piece that $C$ desires, $C$ would choose $A$ because $A$ has a shorter hop count to $C$ than $B$ does. CarTorrent users can send queries to other clients; the query is delay tolerant such that whenever the connectivity is available the query is sent out and resolved. Note that the craving for downloading information, e.g., sightseeing landmarks, movie previews, outweighs the penchant for keeping to oneself and thereby provides incentives for cooperative content sharing.

The goal of this paper is to test the feasibility of in-vehicle content sharing (i.e., CarTorrent). Toward this goal we implement CarTorrent and measure its performance in a real VANET testbed. The use of the peer-to-peer application on a real VANET testbed is the first of its kind. We show that peers can utilize the gossip mechanism to recognize one another's presence and employ the piece-selection strategy to optimally download files from one another. We run extensive field tests and obtain performance measurements in a real VANET testbed. We demonstrate performance comparisons between baseline static parking lot and real road mobile scenarios. We believe that many lessons learned and technologies picked up to set up a testbed of this size are invaluable to VANET research.

The rest of this paper is organized as follows. Section 2 illustrates CarTorrent's architecture and implementation details. Section 3 presents our experiment setup and results. Section 4 shows the related work. Finally, Section 5 draws conclusion of the paper, and present the possible future work.

# 2. CARTORRENT OVERVIEW AND ARCHITECTURE

The rightful progression to cooperative peer-to-peer sharing has led to the concept of CarTorrent, a BitTorrent-like swarming protocol that exploits the broadcast nature of the wireless medium and node proximity by the use of a gossip mechanism and a novel piece selection strategy, respectively. In this section, we describe CarTorrent's overview and architecture.

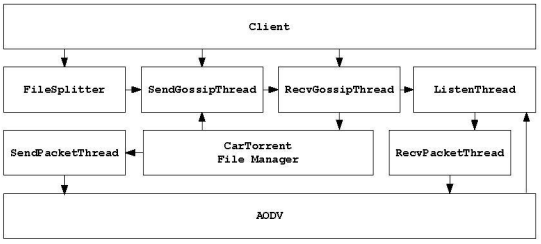



**Figure 2: CarTorrent Architecture**

## 2.1 CarTorrent Overview

Figures 1(a) and 1(b) show the GUI version of the CarTorrent. A typical use case is described as follows: Client $A$ wishes to share a file $F$. $F$ is split into pieces by `FileSplitter`. `SendGossipThread` periodically sends gossips that contains the originator of a file, a sequence number, a filename, a piece availability bit vector, a time-to-live (TTL), and hop count. Upon sharing $F$, `SendGossipThread` sends gossips regarding the existence of file $F$.

Client $B$ sees the file $F$ being shared as its `RcvGossipThread` receives gossips about file $F$. The gossips are then fed to `CarTorrent File Manager` which runs the engine of the piece selection algorithm. Based on the algorithm, `SendPacketThread` requests a particular piece from Client $A$ through `AODV`.

Client $A$'s `ListenThread` serves the incoming requests. Upon receiving a request for a particular piece, the connection is immediately processed by `ReceivePacketThread`. It retrieves the particular piece from its stable storage and sends the piece through `AODV`. The difference between `SendPacketThread` and `ReceivePacketThread` is `SendPacketThread` only sends out requests and `ReceivePacketThread` receives requests and pieces for the requests.

## 2.2 CarTorrent Implementation

We describe the five major components of CarTorrent in Figure 2 in the remaining section.

### *2.2.1 CarTorrent File Manager*

Each file is managed by the CarTorrent File Manager. It is responsible for keeping track of the status of each piece

978-1-4244-1690-5/07/$25.00 ©2007 IEEE.

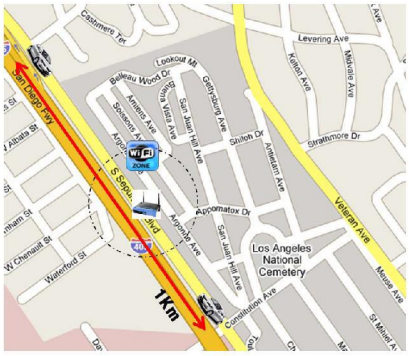



**Figure 3: Straight road scenario**

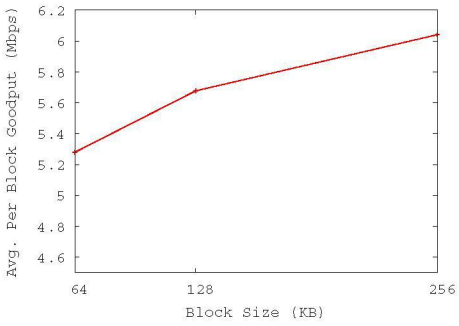



**Figure 4: Per-piece goodput vs. block size**

for the file. In addition, it maintains a list of peers and their hop count with respect to each piece. The status information is updated whenever a piece is received. The list of peers and their hop count are updated whenever a client receives a gossip. The retrieval of closest rarest piece and the peer that owns the piece are conveniently associated with each file. It provides a quick way to determine which piece to download and who the piece should be downloaded from.

#### 2.2.2 *SendGossipThread*

SendGossipThread component is responsible for sending out gossips periodically. There are two types of gossips that it is responsible for. One is from the node itself. The other is from the queue where gossips from the other nodes are kept. Gossips are sent out with different frequencies based on probabilities parameterized in the node's program initially. Gossips that are of the node's interest are sent out with higher probability, therefore, higher frequency than gossips that are not of the node's interest. The interest is determined by whether the node is interested in the file gossiped by the other nodes.

#### 2.2.3 *ReceiveGossipThread*

This component is responsible for receiving gossips. The thread unblocks whenever a gossip is received. The gossip is discarded if the received gossip is from the node itself. If the gossip message is not discarded, it is forwarded to the CarTorrent File Manager for further processing and is kept in a gossip queue to be sent out by its SendGossipThread component.

#### 2.2.4 *SendPacketThread*

This component is responsible for sending out requests periodically. Request for a particular file stops when the client has downloaded all pieces of the file.

#### 2.2.5 *ListenThread and ReceivePacketThread*

When the CarTorrent application starts, the ListenThread component is created for incoming connections. Each incoming connection is then handed off to ReceivePacketThread for further processing. Currently, the system creates three ReceivePacketThreads that will process incoming connections in a round-robin fashion. If the number of incoming connections exceeds three, there will be a delay in processing those requests.

There are two types of incoming packets. The first kind is data request. The ReceivePacketThread will process the data request packet by sending out the piece that the other node is requesting. The second kind is data. The incoming data is the file piece that the node requested. It is saved in the node's local space and later combined when all pieces are gathered.

## 3. EXPERIMENT

In this section, we describe two different experimental scenarios. The first scenario is static communication between two laptops in a parking lot. It serves as a baseline performance comparison with the second scenario in which two moving vehicles download pieces from each other and the AP in a straight road.

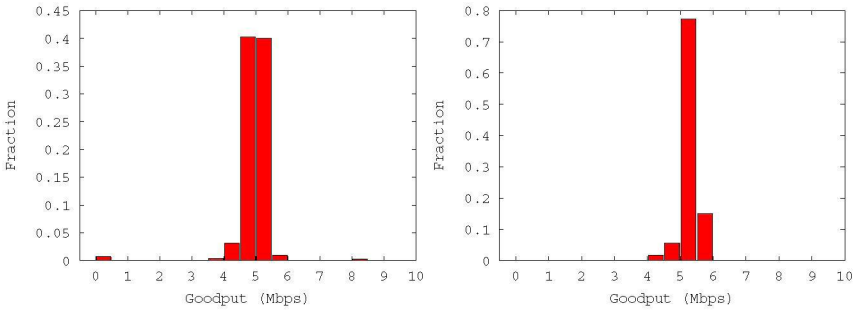
We present results of baseline and straight road. We first present the baseline per-piece goodput versus block size and distribution of the per-piece goodput. We then present the straight road distribution of downloaded pieces over time with respect to the link quality between peers themselves and between peers and the AP and distribution of the per-piece goodput.

### 3.1 Scenarios

We equipped each vehicle with a laptop that has two 802.11b wireless interface cards. One interface card is responsible for communication among vehicles in the ad-hoc mode. The other interface card is responsible for communication between vehicles and the AP in the infrastructure mode. When a vehicle approaches the AP, it receives gossips and requests pieces from the AP on the interface where the gossips are received from. After some time to associate with the AP, the AP will obtain requests and send out the requested pieces. At the same time, the vehicle also receives gossips and requests pieces from its peers on the other interface. Since CarTorrent uses threads to receive incoming gossips, it can take care of simultaneous gossips from both interfaces. Requests are stamped with the originator. This allows the system to identify which interface data should be sent out from. To avoid interference between the two interface cards, we set one card to channel 1 and the other card to channel 11. We used TCP as the transport protocol. We specified a timeout value of 0.65 sec to avoid stalling on the blocking call so that ReceivePacketThread's can continuously serve incoming requests. We tested AODV and incorporated it into CarTorrent.

We performed two scenarios of the experiment. The first scenario was done in an underground parking lot to eliminate interference from other wireless signals. We transferred

978-1-4244-1690-5/07/$25.00 ©2007 IEEE.




(a) 64KB (b) 128KB

**Figure 5: Distribution of the per-piece goodput (Parking Lot Scenario)**

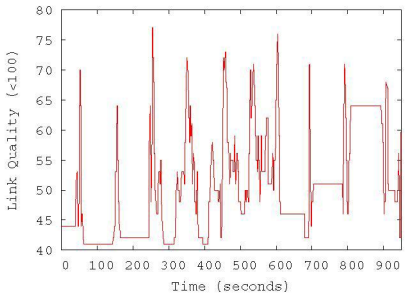



**Figure 6: Link Quality between peers in the ad-hoc mode**

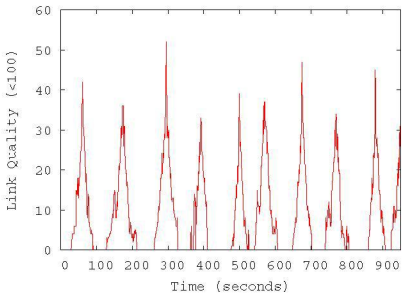



**Figure 7: Link Quality between peers and the AP in the infrastructure mode**

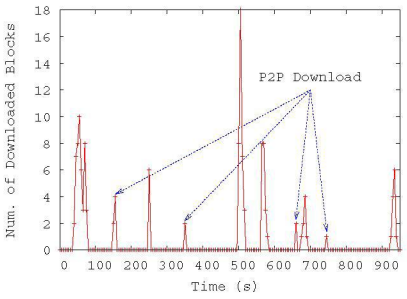



**Figure 8: Distribution of downloaded pieces over time**

a file of 25MB from one laptop to another with piece sizes of 64KB, 128KB, and 256KB, respectively to obtain the baseline download progress over time and per-piece goodput. We then conducted our real VANET experiment on a straight road about 1KM long (Figure 3). We placed the AP in the middle of the road. Two vehicles started from both ends of the street. We observed that cars downloaded pieces from each other and from the AP successfully using CarTorrent's gossips and piece-selection strategy. We show our results in the next section.

## 3.2 Experimental Results

### *3.2.1 Parking Lot Scenario*

Figure 4 shows the average piece goodput for piece sizes of 64KB, 128KB, and 256KB. Because pieces are sent using TCP which takes time to set up, it is more efficient to transmit pieces of larger sizes. However, the slope of such an increase in the average per-piece goodput decreases because bigger pieces are more susceptible to retransmission in a mobile wireless network. Moreover, because of the sender and receiver buffer size, larger pieces are subject to fragmentation and more processing. For those pieces that fail to be transmitted in the range of the peers or the AP, they are discarded after disconnection. All these suggest diminishing return on the average per-piece goodput as the piece size increases.

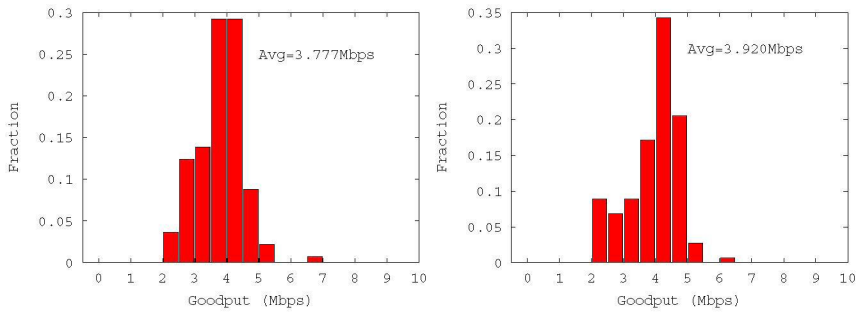
### *3.2.2 Straight Road Scenario*

Figure 5 shows the distribution of per-piece goodput for both 64KB and 128KB pieces. For 64KB pieces, the per-piece goodput is 5.279Mbps. For 128KB pieces, the per-piece goodput is 5.677Mbps. We define the per-piece goodput as the piece size divided by the difference between the time the request is sent for the piece and the time the piece is received.

Figure 6, 7, and 8 show the correlation between the link quality and the number of downloaded pieces in one of the two cars. Peer-to-peer downloaded are indicated by four arrows, denoting four different periods. Because of the space constraint, we only present results from one car. Results from the other car are very similar. Overall, the spikes in link quality correspond well with spikes in numbers of downloaded pieces. In periods 40s-70s and 520s-600s, there were no pieces downloaded from the peer but from the AP. This reflects the way we design the piece-selection strategy. Even though both the peer and seeder of the piece were one hop away from the downloader, the piece-selection strategy favored downloading from the AP. There is no particular reason to program in such a way and we plan to make a change to the strategy.

In period 790s-810s and 890s-910s, even though the link quality between the peers was good, there was no downloading activity. We reason that the node had no pieces that it wanted from its peer. Since both peers owned exactly the same pieces, piece exchange did not take place. This reasoning is backed up by the very last spike we see in 920s-950s; during this period, pieces were all downloaded from the AP. The effect of TCP setup time was seen in 480s-520s. Even though the link quality between the AP and the peer was high, downloading did not start until 510s. The late start

978-1-4244-1690-5/07/$25.00 ©2007 IEEE.




(a) 64KB (b) 128KB

**Figure 9: Distribution of the per-piece goodput (Straight Road Scenario)**

indicates AP association and TCP setup time.

There are two mysterious periods centered around 300s and 400s. During these two periods, the link quality between the AP and the peer was high; however, there was no downloading activity. We compare the same periods with the other peer's piece downloading graph and reason that the "capturing effect" from the other peer requesting for pieces caused the number of threads in the AP busy serving those requests. When the AP was ready to serve requests from the starving peer, the peer had already gone out of range. As discussed earlier, we plan to modify the piece selection strategy so that peers download from each other in the range of AP. We also would like to experiment with increased number of threads in serving incoming requests to minimize starvation.

Figure 9 shows the mean per-piece goodput of 64KB pieces for both peers. The goodput is about 1.5Mbps less than the baseline in the parking lot where there is no interference with other wireless signals.

## 4. RELATED WORKS

BitTorrent[1] has been a popular file-sharing protocol that accounts for a rather significant portion of Internet traffic. However, BitTorrent does not scale well for mobile ad-hoc networks because it gives equal weight to and therefore does not distinguish rarest pieces that are both far and near. This increases the downloading time as one chooses the longer way to get a rarest piece. Because of node mobility, nodes that are further away are a less reliable downloading medium that nodes that are close.

The characteristics of a VANET call for a different design of BitTorrent. The use of BitTorrent-like protocols for VANETs is not new but has been done primarily in analysis on paper or simulations. [7] studies the mobility model of vehicles and utilizes the Random Waypoint Model. It simulates vehicular mobility in ns-2. SPAWN[1], which this work extends to, is only implemented in simulation on Nab, a network simulator written in Ocaml. We have implemented the torrent-swarming protocol based on SPAWN on an actual VANET testbed. Furthermore, we have incorporated the AODV routing implementation in linux developed by Uppsala University as the bottom layer in performing peer discovery and multi-hop message transfer[5].

CodeTorrent [4] is another peer-to-peer file sharing protocol that exchanges pieces by performing encoding operations that exploit the randomness of coded pieces and thereby reduce download time in a VANET. Unlike CarTorrent, a peer only needs to receive any $k$ encoded pieces to decode the file, where $k$ is the number of pieces of a file. The inherent redundancy of network coding takes care of problems of node departures [2]. In addition, nodes take advantage of overhearing coded blocks even though they are not directly involved in communication. This further increases the collection of $k$ coded blocks and thus speeds up the download time. CodeTorrent's strategy in disseminating pieces is different from CarTorrent's. We plan to implement CodeTorrent in a real VANET and compare its results with CarTorrent.

In summary, our work is unique because of the actual CarTorrent implementation and field measurements on a real VANET testbed.

## 5. CONCLUSION

Vehicular ad-hoc networks are surging in popularity and there have been implementations in the areas of sensor networks and advertisement flooding. In this paper, we focus on tackling the problem of an efficient peer-to-peer content sharing protocol and system for vehicular nodes in a real VANET. We have demonstrated the detection of available files with periodic gossips and the successful downloading of files. The efficient selection of a file piece is possible with the use of the Rarest-Closest First strategy where each node first determines the rarest file piece it needs and looks for the closest node that has it. This exploits the mobility of vehicular nodes in error-prone wireless links.

Finally, we have implemented and deployed CarTorrent on a real vehicular ad hoc testbed. We have shown promises of running the peer-to-peer content sharing application that is tailored to the constraints of VANETs. We plan to study the effects of channel condition by varying AP location and improve gossip mechanism by incorporating direction, speed, and distance information.

## 6. REFERENCES

[1] S. Das, A. Nandan, and G. Pau. Spawn: a swarming protocol for vehicular ad-hoc wireless networks. In *VANET'04*, pages 93–94, 2004.

[2] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *INFOCOM'05*, 2005.

[3] A. Klemm, C. Lindemann, and O. Waldhors. A special-purpose peer-to-peer file sharing system for mobile ad hoc networks. In *VTC'03*, Fall 2003.

978-1-4244-1690-5/07/$25.00 ©2007 IEEE.

[4] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla. Codetorrent: Content distribution using network coding in vanet. *MobiShare'06*, 2006.

[5] H. Lundgren, D. Lundberg, J. Nielsen, E. Nordstrm, and C. Tschudin. A large-scale testbed for reproducible ad hoc protocol evaluations. In *WCNC'02*, 2002.

[6] A. Nandan, S. Das, G. Pau, M. Gerla, and M. Y. Sanadidi. Co-operative downloading in vehicular ad-hoc wireless networks. In *WONS'05*, pages 32–41, January 2005.

[7] A. K. Saha and D. B. Johnson. Modeling mobility for vehicular ad-hoc networks. In *VANET'04*, May 2005.

[8] Wardriving. `http://en.wikipedia.org/wiki/Wardriving`.

978-1-4244-1690-5/07/$25.00 ©2007 IEEE.