Cabernet: Vehicular Content Delivery Using WiFi

Jakob Eriksson, Hari Balakrishnan, Samuel Madden
MIT Computer Science and Artificial Intelligence Laboratory

ABSTRACT

Cabernet is a system for delivering data to and from moving vehicles using open 802.11 (WiFi) access points encountered opportunistically during travel. Using open WiFi access from the road can be challenging. Network connectivity in Cabernet is both fleeting (access points are typically within range for a few seconds) and intermittent (because the access points do not provide continuous coverage), and suffers from high packet loss rates over the wireless channel. On the positive side, WiFi data transfers, when available, can occur at broadband speeds.

In this paper, we introduce two new components for improving open WiFi data delivery to moving vehicles: The first, QuickWiFi, is a streamlined client-side process to establish end-to-end connectivity, reducing mean connection time to less than 400 ms, from over 10 seconds when using standard wireless networking software. The second part, CTP, is a transport protocol that distinguishes congestion on the wired portion of the path from losses over the wireless link, resulting in a 2× throughput improvement over TCP. To characterize the amount of open WiFi capacity available to vehicular users, we deployed Cabernet on a fleet of 10 taxis in the Boston area. The long-term average transfer rate achieved was approximately 38 Mbytes/hour per car (86 kbit/s), making Cabernet a viable system for a number of non-interactive applications.

Categories and Subject Descriptors

C.2.1 - Wireless communication

General Terms

Design, Experimentation, Measurement

Keywords

Vehicular networks, vehicle-to-infrastructure, open WiFi access

1 Introduction

This paper describes the design, implementation, and experimental evaluation of Cabernet, a content delivery network for vehicles moving in and around cities. Cabernet delivers data to and from cars using open 802.11b/g (WiFi) access points (APs) that the cars connect to opportunistically while they travel. Cabernet is well-suited for applications that deliver messages (e.g., traffic updates, parking information, event and store information, email) and files (e.g., maps, software updates, documents, web objects, songs, movie clips, etc.) to users in cars, as well as for applications that deliver messages and data from devices and sensors on cars to Internet hosts [16, 22]. These applications do not require interactive end-to-end connectivity between a sender and receiver.

When a car connects via a WiFi AP, it can potentially transfer data at the same rates as static clients connected to the same network. However, as cars move, their connectivity is both *fleeting*, usually lasting only a few seconds at urban speeds, and *intermittent*, with gaps from dozens of seconds up to several minutes before the next time they obtain connectivity. In addition, we observe that packet loss rates over the wireless channel are both high (often 20%) and vary over the duration of a single AP association. The primary goal of Cabernet is to develop techniques that allow moving cars to obtain high data transfer throughput, despite these adverse conditions.

Stock implementations of wireless networking protocols are not well suited to vehicular applications for three reasons. First, current client implementations take too long—several seconds—to scan and associate with an AP, acquire an IP address, and establish end-to-end connectivity in the face of the high packet loss rates in this environment. Second, current end-to-end data transfer and congestion control protocols do not work well when connectivity is only a few seconds long and intermittent, and wireless loss rates are high (as is the case for moving cars). Third, default wireless bit-rate selection algorithms are tuned to non-moving users, resulting in suboptimal performance in vehicular applications.

Cabernet incorporates three techniques to mitigate these problems:

1. To reduce the time between when the wireless channel to an AP is usable and when Internet connectivity through the AP is actually achieved, we have developed *QuickWiFi, a streamlined process that combines all the different protocols involved in obtaining connectivity (across all layers) into a single process, including a new optimal channel scanning policy.*
2. *To improve end-to-end throughput over lossy wireless links, we have developed the* Cabernet Transport Protocol *(CTP), which outperforms TCP over opportunistic WiFi networks by not confusing WiFi losses for network congestion. Unlike most previous work on efficient wireless transport protocols, CTP does not require modifications to APs (which are not under our control); instead, it uses a lightweight probing scheme to determine the loss rate from Internet hosts to an AP.*
3. *To improve link rates, we study the impact of bit-rate selection in vehicular WiFi. Based on our results, we adopt a static 11 Mbit/s WiFi bit-rate for transfers from the car. This optimization could not be applied for downloads from Internet hosts, as the AP controls the bit-rate in this direction.*

The Cabernet design and protocols presented in this paper have been fully implemented. The system is currently deployed in 10 taxis running in the Boston area. The results reported in this paper are from the real-world operation of this testbed, with the in-car nodes running QuickWiFi and the system running CTP. Our key experimental findings are as follows:

Mobicom '08, September 14–19, 2008, San Francisco, California, USA.
Copyright 2008 ACM 978-1-60558-096-8/08/09 ...$5.00.

Many short connections (Section 4): QuickWiFi enables the use of many brief connection opportunities, resulting in a relatively short mean connection time of 10 seconds (median 4 seconds) compared to previous work []. Correspondingly, the mean time between successful encounters is also lower (e.g., 120 seconds, vs. 260 seconds [8], also obtained in the Boston area).

Fast connection establishment (Section 5): QuickWiFi reduces the time to associate with a wireless network from a previously reported average of 12-13 seconds [8, 15] to 366 ms. We find that 70% of connection opportunities last less than 10 seconds. Given the large number of brief connection opportunities, QuickWiFi provides a four-fold increase in the number of connections that transfer data during drives, compared to a system without QuickWiFi.

Improved transport performance (Section 6): By not reacting to non-congestion losses over the wireless link, CTP improves data delivery rate by a factor of two over TCP.

Fixed bit-rate works best (Section 7): Experiments over 2300 connections showed minor differences in loss rates between 1 Mbit/s and 11 Mbit/s rates, and a dramatic increase in losses when using 802.11g (OFDM) rates.

End-to-end performance attained (Section 8): Each car in our testbed was able to download 38 MByte per hour (86 kbit/s) while moving, averaged over periods with and without connectivity. Response times can be high, due to intermittent, bursty connectivity. For example, the observed mean time from request to receiving a 1 MByte response was 9 minutes. These results suggest that Cabernet is well suited to a large class of non-interactive applications.

Of course, there are non-technical issues surrounding both the large-scale use and deployment of open access points. We address these issues briefly in Section 10.

2 Related Work

Cellular networks. Today, delivery of information to moving cars is done mainly using wide-area cellular data networks such as GPRS, 3G, EVDO, etc. In general, these services cost tens of dollars per month. We envision Cabernet as an alternative for "lower end" and embedded mobile information services, which do not justify the cost of a cellular data plan subscription. For example, in our testbed the price of an EVDO cellular data plan is approximately $720/year per car, making it prohibitively expensive for many users and services. It is likely, however, that many users will eventually have cellular data service available on their cell phones. Such connectivity could conceivably be shared with embedded devices in the car, given that these could be properly authenticated. Compared to such solutions, Cabernet makes no assumptions about drivers' other data services, and requires no prior configuration.

With the proliferation of municipal and public WiFi initiatives, in addition to major commercial initiatives, Cabernet is likely to be sufficient for many applications. Even if the cost were not an issue, the *fundamental* capacity of Cabernet is much higher than current cellular networks: given enough access points, the high link speed and intense spatial reuse of WiFi is unlikely to be matched by any cellular system. Because Cabernet does not provide continuous connectivity, however, our goal is to complement cellular data services, not to replace them.

Vehicular networks. The Drive-Through Internet [25], and Infostations [13, 20, 17, 27] projects propose architectures similar to Cabernet. The main difference is that Cabernet uses existing, unmodified APs, and has been evaluated extensively under real-world conditions on a taxi-based testbed. Several projects have studied the problem of intermittently connecting from cars or other mobile devices to the Internet. Our previous work in the CarTel project investigated general architectures for vehicular sensor networks [16], and characterized the extent to which wireless access points deployed in cities can be used as an uplink network for moving cars [8]. This work did not address network performance issues, such as optimized association, scanning, data transport protocols or rate selection.

The UMass DieselNet is a delay-tolerant network running on 40 buses in Amherst. They have studied protocols for bus-to-bus routing in delay tolerant networks [7] as well as measurements and models of bus-to-bus connectivity [28]. In [4], a set of techniques are proposed to enable web search from a bus, using only WiFi connectivity. The authors report an average meeting duration of 55-58 seconds, significantly higher than the 10 seconds we report. This result is a side-effect of using QuickWifi: due to the long connection establishment delays incurred using standard techniques, DieselNet was likely unable to take advantage of connections shorter than 10-15 seconds (>70% of connections in our experiments). They report a median time of 5-8 minutes between successful connections vs. 32 seconds for Cabernet. However, due to the difference in mobility (bus vs. car), and setting (Amherst vs. Boston), these numbers are not directly comparable.

Hadaller et al. [15] have done a detailed study of WiFi connectivity from a vehicle by repeatedly driving past a single access point in an isolated area without other interference or obstructions. They find (as did Gass et al.. [11], Ott and Kutscher [25], and Bychkovsky et al. [8]) that connection establishment times can be quite long (12-13 seconds). They recommend not initiating connections until cars are in a zone of "good" connectivity (approximately the middle 30 seconds of each connection). In Cabernet however, the mean connection only lasts about 19 seconds, and many connections never achieve "good" connectivity; we find packet loss rates in excess of 10% throughout many connections. They suggest tuning timeouts in the wireless stack to improve performance, but do not experimentally evaluate the effects of such changes. We have implemented a number of optimizations in QuickWifi, including various timeout optimizations. Finally, unlike in our experiments, they find that bit rates above 11 Mbit/sec are usable and TCP losses are low. We attribute the significant differences between our respective findings to the fact that their experiments were conducted in a relatively controlled and benign setting.

Mahajan et al. [23] study signal quality between two vehicles and several base stations. They find that harsh fading occurs throughout the duration of a vehicular WiFi connection, and that such periods of poor connectivity can be predicted based on past history. Our results corroborate these, and an interesting direction for future work would be to incorporate their AP prediction algorithms into Cabernet.

Delay tolerant networks. A variety of work on delay tolerant networks [10, 18, 19, 26, 29, 5, 21] has studied architectures, routing, forwarding, and analysis of intermittent and high-delay networks. In contrast, our focus is not on routing—we assume that mobile nodes are just a single hop from well-connected, fixed infrastructure—but on maximizing the utility of that single-hop path.

3 Experimental Setting

Before describing the components of the system in more detail, we briefly summarize the characteristics of our experimental testbed, the equipment used, and the data gathered. We have deployed a mobile vehicular testbed, hosted in taxis in the Boston area. It consists of 25 nodes, each a Soekris 4801 computer, equipped with a Ubiquity

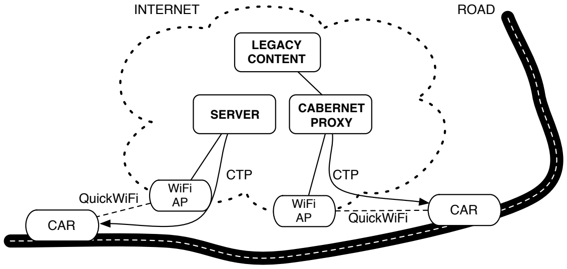

Figure 1: Cabernet architecture. Cabernet-equipped cars briefly connect through open WiFi access points as they move.

Networks SR2 (Atheros chipset) 802.11b/g radio, a small 3 dBi omnidirectional antenna mounted inside the passenger compartment, and a GPS receiver. We allocated 10 of the testbed nodes for use in the Cabernet experiments.

The results reported in this paper were derived from a variety of experiments with varying duration and overlap. In total, we have successfully transferred data using 26000 open APs. This is out of several hundred thousand observed APs all together, the vast majority of which were encrypted. However, except where noted otherwise, most results were computed using 124 hours of actual mobility, defined as driving periods with a maximum stop of 2 minutes. Note that we had no control over vehicle mobility—these measurements were truly gathered "in the wild". That said, it is likely that some of the results reported here may look different when compared to results from private cars or public transport, both of which may have less diverse travel patterns.

4 Design Overview and Goals

This section describes the architecture of Cabernet, illustrated in Figure 1. As a car drives down the road, the onboard embedded computer repeatedly scans for, and attempts to associate with, open APs. It then attempts to establish end-to-end connectivity with a Cabernet enabled host, to retrieve or upload data. Software running on the embedded computer uses QuickWiFi to connect as quickly as possible, and CTP to deliver data (instead of TCP, though TCP could also be used). Finally, communication with legacy Internet hosts is done using a proxy that serves two functions: first, it hides intermittent connectivity from protocols running on fixed Internet hosts and shields them from changing IP addresses, and second, it translates between CTP and popular protocols like HTTP and TCP.

At a high level, Cabernet's design can be described by the following three objectives:

1. Establish connectivity quickly to take advantage of brief connection opportunities.
2. Split congestion control between wired and wireless portions of the path to cope with high non-congestive wireless loss rates.
3. Expose the appearance and disappearance of connectivity to Cabernet-enabled applications on the mobile node.

We motivate and discuss each of these in turn below.

4.1 Establish Connectivity Quickly

When a vehicle is in motion, available WiFi connections are typically brief. Figure 2 shows the CDF of the duration of over 5529 *encounters* (defined below) with 1396 unique APs, when the cars were in motion. This data was collected using a single WiFi radio on each car. The radio scanned continuously for open APs until one was found. It then associated with the AP and maintained the connection until the access point went out of range. Only APs that provided end-to-end connectivity are reported here. We define an encounter as the interval between when the first beacon was heard from an AP and the time at which the last packet was heard from the AP. This duration is an upper bound on the time that could profitably be spent communicating, because it does not include the time it might take for a client to actually achieve end-to-end connectivity. In our experiments, the median duration of an encounter is 4 seconds, with a mean of 10 seconds and standard deviation of 0.4 seconds. 70% of the observed encounters last less than 10 seconds.

Stock implementations of the IEEE 802.11 and Internet protocols typically require several seconds to establish a connection (e.g., Bychkovsky et al. [8] report establishment takes 12.9 seconds). By considerably shortening this time, QuickWiFi is able to exploit the bulk of these fleeting opportunities. We discuss QuickWiFi in more detail in Section 5.

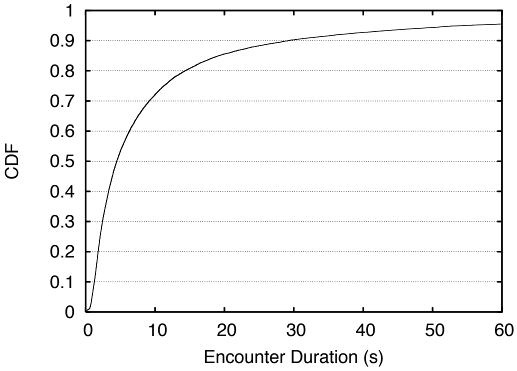

Figure 2: CDF of encounter duration. The median is 4 seconds, mean is 10 seconds. 99th percentile was 250 seconds.

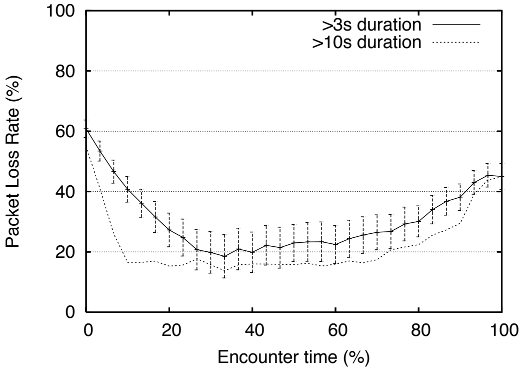

Figure 3: Packet loss rate vs. Encounter time. High losses are experienced throughout an encounter. Error bars show standard deviation, >10s error bars omitted for readability.

4.2 Handle Non-Congestion WiFi Losses

Packet losses over the wireless link are common in our observed encounters, with a mean packet loss rate in excess of 20%. Figure 3 shows how the packet loss rate varies with time from the beginning of each encounter to the end. To collect this data, the cars spent the

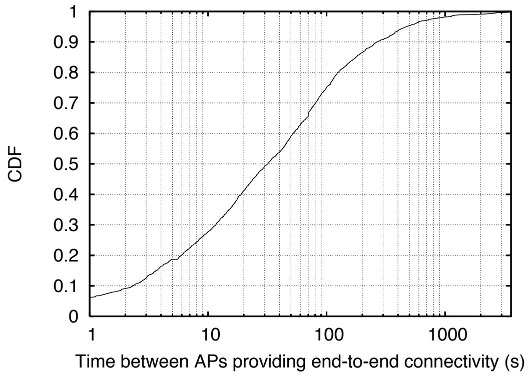

Figure 4: CDF of time between APs providing end-to-end connectivity.

duration of 850 random individual encounters sending bursts of back-to-back packets to the AP, with link-layer retries enabled, at the 11 Mbit/s bit-rate. MAC layer ACKs were used to verify transmission success or failure. The results were essentially identical for lower bit-rates, as we discuss in more detail in Section 7.

As one might expect, the beginning and end of an encounter experience high loss rates, but even the middle of an encounter is prone to losses. Most of these losses are unlikely to be due to congestion and are instead caused by a combination of marginal links and mobility.

Given these packet loss rates, a congestion control mechanism such as TCP's that treats all end-to-end packet losses as congestion is sub-optimal. Unlike in much previous work on improving wireless transport performance, our environment has legacy APs that aren't under our control. We solve this problem in CTP by implementing a lightweight probing protocol to independently detect congestive losses on the (wired) path to the AP. In addition, we investigate wireless bit-rate selection in the Cabernet setting to improve throughput. We discuss CTP in more detail in Section 6 and bit-rate selection in Section 7.

4.3 Managing Intermittent Connectivity

The distribution of times between encounters depends on the density of open APs in the area and the speed of travel. Figure 4 shows the CDF of the time between AP encounters that provided end-to-end connectivity, based on experiments spanning 124 hours of driving. The observed median time between successful encounters was 32 seconds and the mean was 126 seconds, heavily skewed by a small number of longer intervals. With 90% probability, a successful encounter will happen within just five minutes after the end of the last encounter.

Given the short encounter durations, most encounters will not be long enough to complete a requested transfer. Because most current applications and protocols do not tolerate intermittency, we either need new delay-tolerant applications, or we need proxies that hide intermittent connectivity. Cabernet provides support for both types of use. For new applications, the CTP API provides prompt feedback (through OS signals) whenever end-to-end connectivity disappears and reappears. These notifications allow applications to begin transfers when connectivity appears and avoid having to implement complex application-specific timeouts or connection polling techniques to determine if connectivity is available.

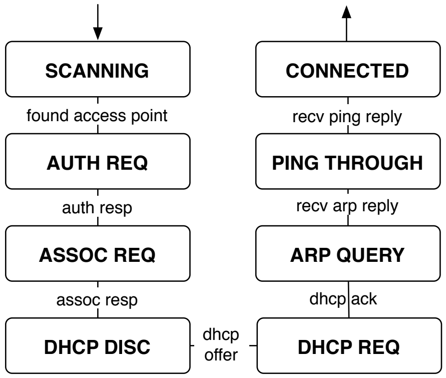

Figure 5: Steps involved in connection establishment. Establishing a connection requires a minimum of 13 frames.

Legacy applications, which do not implement CTP or which may not be well suited to conditions of intermittent connectivity, are supported by a combination of client-side and Internet-based proxies, as proposed by Ott and Kutscher [25].

5 Establishing Connectivity

Establishing an Internet connection over a wireless AP involves several steps which, using stock implementations of various protocols, can take significant time to complete. The situation is exacerbated in vehicular environments because of the high packet loss rates and short encounter durations.

Figure 5 illustrates the connection establishment process. In general, each step involves a request, followed by a response. If no response is received within some specified time, the request is retransmitted. In our case, an additional "ping-through" step is used to verify that an end-to-end connection is available.

5.1 Problems with Stock Implementations

Below, we list several reasons why it can be difficult to use stock tools for connection establishment in a vehicular setting, all of which are addressed by QuickWiFi.

Suboptimal scanning. Scanning by stock implementations typically assumes that the channels are occupied by APs with equal probability, and that APs, once discovered, are unlikely to disappear in the next several seconds.

Inappropriate timeouts. At high packet loss rates it is crucial to appropriately tune retry and timeout values: 13 frames need to be received over the wireless channel before a verified end-to-end connection is available.

Sequential, yet asynchronous. The steps required in establishing a connection are sequentially executed, but usually by separate processes. There is often no explicit notification between processes, causing extra delays, and lost opportunities for parallelism.

Manual intervention required. Stock implementations typically rely on users to choose what APs to connect to, or when to give up and try connecting to a different AP.

Always-on connection model. Current client stacks are not designed for the on-again, off-again connectivity available through moving vehicles. While Mobile IP could be used to handle session migration, applications need to be explicitly notified when connectivity is available, in order to make full use of it.

5.2 QuickWiFi Operation

QuickWiFi incorporates fully automatic scanning, AP selection, association, DHCP negotiation, address resolution, and verification of end-to-end connectivity, as well as detection of the loss of connectivity. It is implemented as a single state machine, running in one process. The result is a tight integration between steps: each step is executed immediately after the previous step has completed, and parallelism is exploited whenever possible. We now describe some of the optimizations incorporated in QuickWiFi. The optimized channel scanning technique used in QuickWiFi is described separately in Section 5.3.

In the current implementation, QuickWiFi attempts to associate with the first open access point it encounters as it scans through the wireless channels. After a connection is complete, it resumes scanning at the point where it left off. In practice, this policy results in a quick way to select a random AP, since any AP beacon may be the first one heard when a scan begins. As future work, it would be interesting to explore schemes that cache historical AP connectivity information so that APs with better connectivity are selected with higher probability. Such schemes are tricky to implement as they require waiting for a scan to complete (decreasing usable connection times) and can also decrease the chances that new APs are discovered and used.

Tuned for vehicular WiFi. Authentication, association, DHCP, and ARP all include timeout/retry protocols. Retries are very likely, particularly in DHCP discovery and ARP request phases when broadcast messages (without link-layer ACKs) are used. By reducing timeouts to hundreds of milliseconds (which is far longer than the typical wireless channel round trip time), rather than seconds (as in most stock implementations), we can dramatically reduce the mean connection establishment time. In QuickWiFi, the timeout period in each phase is set to 100 ms, after which the request is sent again, up to 5 times. If the 5th request fails as well, the process starts over with the scanning phase. The choice of 100 ms here is somewhat arbitrary. Further improvements may be possible by tuning the timeout interval depending on the message type.

Back-to-back authentication/association. Even open APs require clients to authenticate, but this step is always successful. Thus, there is no need to wait for the response to our authentication request before sending the association request. Should the authentication request be lost in the meantime, the subsequent association request will fail, and QuickWiFi will automatically restart the process.

QuickWiFi includes two additional functions to monitor and report the status of an end-to-end connection.

Ping-through. Many seemingly open access points do not provide end-to-end Internet connectivity (because, for example, they require the user to register before granting end-to-end connectivity). QuickWiFi uses a quick request-response exchange with a central server to verify end-to-end connectivity. Upon success, QuickWiFi explicitly notifies applications that Internet connectivity is available, through an OS signal. Should the end-to-end connectivity test fail, the connection is torn down, and scanning is resumed.

Connection loss monitoring. We need to quickly discover when a connection is lost, and return to scanning for new APs. In a vehicular scenario, if we have not seen any transmissions (including beacon frames) for 500 milliseconds, it is likely that the car has moved out of the range of the AP. In this case, scanning is resumed and running applications are notified.

Finally, we have implemented an optimized scanning strategy, as described in the following section.

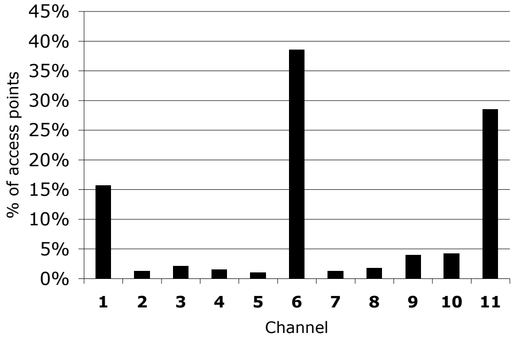

Figure 6: Distribution of APs across channels.

5.3 Optimal Scanning Strategy

In IEEE 802.11b as specified for the U.S., there are 11 (partially overlapping) channels. Scanning can be done either passively or actively. Typically, a *passive* client scans by tuning to each of the available channels and lingering for at least 100 ms on each channel to listen for beacons from nearby access points. With 11 channels, this approach results in a mean delay of 600 ms between entering the coverage range of an access point and discovering it. An *active* client transmits a broadcast probe request, which APs optionally respond to. In order to discover APs that do not respond to such probes, a linger time in excess of 100 ms is needed. Scanning type is a configuration option in QuickWiFi, and for the experiments reported here, we used passive scanning.

Only three channels in IEEE 802.11b/g are considered "orthogonal" (free of co-channel interference): 1, 6 and 11. Users (or perhaps device manufacturers) take this information into account in practice; Figure 6 shows that the distribution of AP channels in our data set is non-uniform and biased toward these channels. 83% of unique access points observed are assigned to one of the three orthogonal channels; 38.5% use channel 6.

Given this information, our goal is to find a scanning strategy that minimizes the time it takes to find a channel with an open AP present. Let N be the number of channels, and $\lambda_k, 1 \leq k \leq N, \sum \lambda_k = 1$ be the probability that an arbitrary access point is assigned to channel k.

We are interested in obtaining an optimal scanning strategy for a moving car that is continually scanning for a usable AP, where the APs are configured independently at random to channels according to the λ probability distribution. We seek a scanning strategy that scans channel k at a rate f_k; i.e., given a large number of total scans S scans, channel k gets scanned at regular intervals a total of Sf_k times, and $\sum_k f_k = 1$.

Consider any strategy that scans each channel k at a rate proportional to f_k (e.g., weighted round-robin scheduling). Here, the expected number of scans before a given channel k is scanned is proportional to $1/f_k$. A fixed schedule of this sort is preferable to simply randomly sampling channels and selecting channel k with probability f_k because random sampling (with replacement) may result in it occasionally taking a very long time to scan a channel with low f_k. Assuming the car is able to complete a scan before it moves out of range of any access point, it is preferable to scan such rare channels before re-scanning a channel that has already been scanned. Using a fixed scanning strategy, the expected number of scans before the car successfully finds a single point on a random channel is therefore proportional to $\sum_k \frac{\lambda_k}{f_k}$.

Our goal is to find an assignment to $f_k \forall k$ that will minimize this quantity, subject to $\sum_k f_k = 1$. We now show that this quantity is minimized when $f_k \propto \sqrt{\lambda_k}$. Using a Lagrange multiplier, μ, define

$$h = \sum_k \frac{\lambda_k}{f_k} - \mu(1 - \sum_k f_k).$$

Taking the partial derivative with respect to f_k and setting to 0 yields

$$\frac{\partial h}{\partial f_k} = \frac{-\lambda_k}{f_k^2} + \mu = 0,$$

and thus $f_k \propto \sqrt{\lambda_k}$. Applying the constraint $\sum_k f_k = 1$, we obtain $f_k = \sqrt{\lambda_k} / \sum_i \sqrt{\lambda_i}$.

Using packetized processor sharing, we can now generate a feasible scanning schedule, giving a weight to each channel k equal to $\sqrt{\lambda_k}$. The scan order is determined by the schedule produced. For example, with λ_k as in Figure 6, the following schedule achieves an expected number of scans of 3.64: $6, 11, 9, 2, 6, 1, 11, 10, 6, 3, 1, 11, 6, 4, 9, 11, 6, 1, 5, 10, 6, 11, 7, 1, 8$, about 40% better than the naive scanning approach.

5.4 Open AP Co-occurrence and AP Selection

It is common for a node to receive beacons from more than one open AP. For intervals of 1 second, multiple open APs were observed in 50% of cases. QuickWiFi does not explicitly address this scenario. Instead, it immediately initiates an association attempt as soon as an open AP is observed. Assuming that APs send beacon frames at similar rates, this strategy results in a random selection between the available APs, weighted by the packet delivery probability from each AP to the car.

More carefully selecting which AP to associate with may improve performance. However, several issues make this a nontrivial task. For example, many seemingly open APs do not actually provide end-to-end connectivity. If, in some areas, such APs have consistently higher signal strength, any signal-strength based strategy would likely degrade performance. Similarly, focusing on APs with a good track-record may unnecessarily avoid recently added APs, or recently reconfigured ones. Finally, a trade-off exists between spending time scanning for the "ideal" AP, and aggressively associating as soon as an opportunity arises. We leave this opportunity to future work.

5.5 QuickWiFi Performance

We now turn to the performance of the connection establishment process. For each successful connection, we measure the time between when the car discovered an open access point, and when QuickWiFi reported a successful connection to the application. Figure 7 shows the CDF of connection establishment times, based on a total of 16800 connections to 2700 unique APs. With a mean connection establishment time of 366 ms, and a median of 287 ms, QuickWiFi achieves about a 35-45× speedup over the previously reported median time of 12-13 seconds [8, 15].

Dissecting QuickWifi latency. QuickWiFi spends 366 ms on establishing a connection, but how is this time spent? Figure 8 shows the mean time spent in each of the phases, for all connections that eventually succeeded. DHCP dominates the delay incurred, with a total of 197 ms spent on the two DHCP phases. Part of the the reason for this delay is that both the DHCP discover and the DHCP request packets are sent to the broadcast address. Such packets do not benefit from link-layer ACKs and retransmissions, which means that this stage takes a long time to complete. Another reason is that most (85% in our experiments) DHCP servers send an ARP request prior to responding to a DHCP Discovery message, in order to ensure that the IP address to be sent in the DHCP Response is not already in use. This appears as part of the DHCP Discovery time. An overheard ARP request from the DHCP server can be used to accelerate the DHCP address acquisition process, as described in [12]. QuickWiFi supports this option, but it is turned off by default out of caution: using an IP address before it is granted by the DHCP server may disrupt a client already using the address.

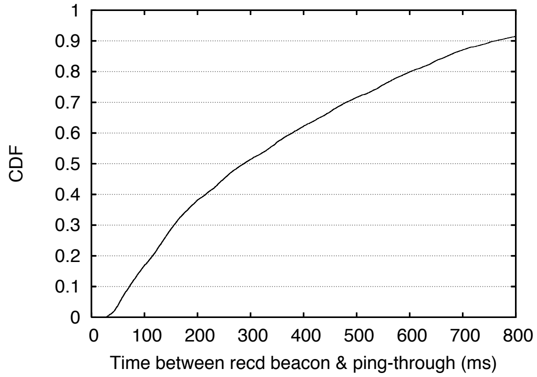

Figure 7: QuickWiFi speeds up connection establishment. CDF of connection delay when using QuickWiFi. Median = 287 ms, mean = 366 ms. 95th percentile = 920 ms, 99th percentile 1230 ms.

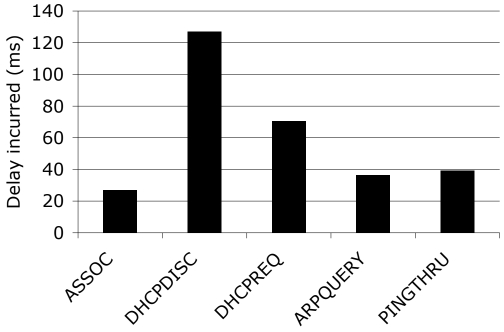

Figure 8: Mean time (s) spent in the various phases of connection establishment.

ARP queries from the client are also sent to the broadcast address. We believe, however, that the shorter time spent on ARP queries can be explained by survivorship bias: once both DHCP phases have completed, the connection quality is significantly improved given success in the two DHCP phases. In addition, ARP queries are short compared to DHCP packets (< 30 bytes of payload and no IP header, compared to 200 bytes), resulting in a lower probability of packet loss. The relatively short time taken for ping-through (40 ms) is consistent with the round-trip time to our server at MIT from locations in the Boston area.

How do connection attempts fail? The shorter timeouts used in QuickWiFi lead to faster connection establishment times. However, several attempts fail at different stages. Figure 9 shows the distribution of connection failures, for all access points that allowed QuickWiFi to associate. CONNLOSS indicates those connections

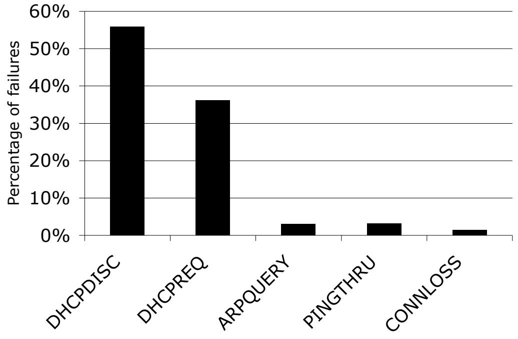

Figure 9: Connection failures (percent) in QuickWiFi. Bars indicate percentage of connection failures at various phases.

that were successfully established, and subsequently failed as the car moved out of range of the AP.

We note that the vast majority of failures happen in the DHCP phases. There are several possible contributing factors to this. Poor channel quality is in all likelihood the biggest culprit. Due to the broadcast nature of DHCP messages, there are no MAC layer retries, with a corresponding increase in packet losses. However, some DHCP servers may be unable to respond before QuickWiFi times out and restarts the process. An increased number of retry attempts for DHCP discovery would help address this. In the balance lies wasted opportunities to attempt association with other APs. Every second spent trying to get a DHCP response from an AP that doesn't produce one is potentially a second of connectivity wasted. Additional experimentation with varying DHCP timeouts and retries in the DHCP steps may offer further performance gains.

6 Cabernet Transport Protocol

As discussed in Section 4, Cabernet uses a proxy to mediate between unmodified Internet hosts (e.g., servers) and the node in the car. This proxy shields the car's intermittent connectivity and changing IP address from legacy hosts. In our initial deployment and experiments, the proxy used TCP to communicate with the in-car nodes. Despite the widespread use of 802.11's link-layer acknowledgment and retransmissions, we found that the IP-layer loss rates were quite high, and that TCP performance was disappointing.

At first blush, the large amount of research done on wireless TCP improvements would seem to be directly useful in our context [3, 2, 1, 6, 9]. Unfortunately, most of these schemes involve modifications to the access point (or to a host very close to it). In Cabernet, because we don't control the APs, we need an end-to-end solution that can run at only the proxy and in-car node. Although the number of previous protocols that operate under this constraint is small, two are potentially applicable: TCP with Explicit Loss Notification (e.g., E2E-ELN and E2E-ELN-RXMT from [3]), and TCP Westwood/Westwood+ [24, 14]. We consider each in turn here.

TCP with ELN may be used end-to-end if the receiver's link layer is modified to deliver erroneous frames to the TCP receiver, so that the receiver can send ELN messages to the sender. For this scheme to work, the TCP header must be separately protected with an error detection code, since otherwise an ELN could end up being sent to the wrong connection and possibly disrupt the state of that connection (e.g., imagine the sequence or port number fields being corrupted). More importantly, because many non-congestive losses corrupt the preamble, those link-layer frames would never get decoded by the receiver, implying that many such losses would never trigger an ELN. The resulting throughput would not be good. In [2], this problem was dealt with by having the nodes at the ends of the wireless link maintain state to detect holes in the TCP sequence space, but such modifications require changes to APs.

A TCP Westwood/Westwood+ sender does not cut its congestion window by half on a loss, but instead sets the congestion window to the product of the "bandwidth estimate" and the minimum round-trip time (RTT) for the connection, divided by the segment size. The bandwidth estimate depends on the rate at which acknowledgments arrive, with Westwood+ implementing a method to avoid over-estimating when ACK compression occurs. On a timeout, the congestion window is set to 1 segment.

We didn't use Westwood+ because our measurements showed loss rates of 20% or more, much higher than the random loss rates at which the Westwood papers showed benefit (5% or less). Because Westwood+ does not actually try to distinguish between wired and wireless losses, but is only less conservative about setting the congestion window and slow start thresholds, it is unlikely to perform well over links with high and bursty losses, as in our system. Therefore, we developed a new protocol.[1]

CTP is an end-to-end, reliable, stream-oriented transport protocol that aims to achieve reasonable throughout for data transfers over a series of short and loss-prone wireless connections. It hides network mobility (changing addresses and underlying connections) from the application. It incorporates a different congestion control mechanism from the many previously proposed TCP variants to perform better over networks with high non-congestive loss rates and it does so while running over legacy wireless APs.

6.1 Reliability and Mobility

CTP exposes a reliable socket API to the application. From the application's point of view, the main difference between CTP and a normal TCP socket is that a CTP session does not break when the underlying IP address changes or path disappears. Instead, CTP end hosts carry unique network-independent identifiers, allowing CTP sessions to migrate seamlessly across changing IP addresses and APs.

On an address change, the client reconnects to the other end (assumed to be a permanently connected, CTP-aware Internet host) and securely identies itself using its unique identifier. CTP delivers data reliably in both directions using large send and receive buffers that can hide outages that could last several minutes. An outgoing message is divided into packet-sized *chunks*, each of which is assigned a chunk sequence number. In order to reduce wireless contention, ACKs are aggregated and sent at a moderate rate (once every 100 ms). Each ACK contains a bitmap indicating what chunks are still missing. The CTP sender retransmits chunks if they show up as missing more than a threshold number of times in the bitmap or after a timeout.

6.2 CTP congestion control

From an end-to-end perspective, non-congestive wireless losses are indistinguishable from congestion, so protocols like TCP react by reducing their windows and increasing the likelihood of a timeout. CTP improves throughput by reacting only to congestive losses,

[1] That said, comparing CTP with Westwood and TCP-ELN would be an interesting area for future work.

which are assumed to occur only on the path from the Internet sender to the AP.

We believe this assumption is likely to be valid in the case where the car is downloading from the Internet. Here, concurrent transmissions are handled by the 802.11 MAC protocol running on the AP. In the case of a concurrent upload (e.g., another wireless client sending to the Internet while the car is downloading), there are no hidden terminals with respect to the AP, and thus sharing should be adequately handled by the MAC protocol. In the case of concurrent downloads (sending from the Internet to other wireless clients), sharing the wireless medium is not an issue, as the AP is doing the majority of the sending.

For the case when we are sending from cars to the Internet, which is not the focus of this paper, hidden terminals may be an issue under some conditions, as the sender may not be able to detect other concurrent transmissions.

For transfers from a car to an Internet host, the CTP sender has an easy task: it receives immediate feedback from the absence of link-layer ACKs; propagating this information to the CTP layer allows the sender to retransmit data without reducing the CTP transmission rate. Our main focus in this section, however, is on the more difficult case of transfers from an Internet host to a car. The challenge is to detect congestion on the wired part of the path without getting confused by wireless losses. To achieve this goal, the CTP sender transmits probe packets periodically to the AP in question. A lack of response to such probe packets is interpreted as a sign of congestion. This information is combined with the end-to-end RTT and end-to-end packet loss rate to calculate the transmission rate.

6.2.1 Probing Strategy

Because the APs aren't under our control, the choice of a suitable probe packet depends on its ability to elicit responses from the AP. A good scheme must have low overhead and work in the presence of NATs and common firewall configurations. Below, we describe the three methods used by CTP, in order of preference, and named according to the expected response from the AP:

1. **TCP RST.** Send a large (1400-1500 bytes) packet that looks like a TCP segment to a randomly selected, high-numbered port on the AP. Because the AP does not have a record of a corresponding TCP session, it will respond with a small TCP RST message.
2. **ICMP TIMXCEED.** Send a packet that is identical to a regular payload packet belonging to the on-going CTP connection, but with the IP header's time-to-live (TTL) field set to one less than the number of hops to the car. The CTP receiver continuously updates the sender's TTL estimate through a field in the ACK. Assuming the TTL estimate is correct, the packet will expire when it reaches the AP, which will respond with a small ICMP TIMXCEED message.
3. **ICMP ECHO.** Send a 1500-byte ICMP ECHO (ping) packet to the AP as the probe. If the AP is ICMP-enabled, it will respond with a 1500-byte ICMP response. This scheme is somewhat wasteful of uplink capacity, but acceptable given that probe packets make up a small fraction of the overall data transfer.

In all cases, it is necessary to send a large probe packet, in order to accurately measure congestion losses for large payload packets. Out of the three methods listed, TCP RST is preferred because APs that do provide a TCP RST response tend to do so without rate limitation. ICMP probes, however, tend to be rate limited to 1-2 per second. Unfortunately, most APs (more than 80%) do not provide TCP RST responses.

If none of these techniques produces a response from the AP, CTP relies on observed end-to-end packet losses for rate control, and adjusts the rate on each end-to-end ACK, essentially reverting to TCP behavior.

6.2.2 Rate adjustment

Using the feedback provided by end-to-end ACKs and periodic probe packets, CTP continuously adjusts the transmission rate for the connection. The objective of the rate control algorithm is to emulate the throughput that a bulk TCP transfer would achieve if it went over a path that had the same RTT as the end-to-end CTP connection, but the loss rate of the path from the sender to the AP. Such a rate control scheme would interact reasonably well with other TCP traffic on the same path, but achieve higher throughput than a connection that used the end-to-end loss rate (which in general would include a large non-congestive component). After some experimentation, we arrived at the following rate adaptation rules.

Rate increase rule. The CTP sender considers increasing its rate each time it gets an ACK. If the CTP ACK indicates that no payload packets were lost during the *ack_interval* period, the new rate for a long-running CTP connection is computed as

$$rate \leftarrow rate + \frac{ack\_interval}{RTT^2}. \qquad (1)$$

This increase rule is similar to TCP's additive increase, where TCP increases the window by 1 packet per RTT (actually, 0.5 packets with delayed acks but we ignore that here). Therefore, in one RTT (measured in seconds), TCP's rate increases from W/RTT packets/s to $(W+1)$/RTT packets/s, where W is the window size in packets. The increment in rate is 1/RTT packets/s, which is made every RTT seconds. Hence, in *ack_interval* seconds, the rate should increase by the amount shown above.

Rate decrease rule. CTP considers reducing its rate every time a new probe packet is sent. If the previous probe packet got no response, the new rate is computed as

$$rate \leftarrow \max(rate \cdot (1 - c \cdot p_{loss}), r_{min}), \qquad (2)$$

where p_{loss} is an exponentially weighted moving average of the loss probability, computed from the ACKs, and c is a tunable constant.

This decrease rule is a heuristic that has to reconcile some real-world problems. The feedback received from probe packets is not frequent enough to accurately estimate congestion loss rates. Instead, we use it as a *binary signal* to determine whether or not to reduce the rate. To estimate the *amount* of reduction that is needed, we still fall back on the end-to-end delivery rate.

Given the high rate of wireless losses in Cabernet, however, simply reducing the rate by half for every dropped payload packet would be a poor strategy. Instead, given the congestion signal received from probing, we conservatively use the end-to-end loss rate estimate as an upper bound on the congestion loss rate. Thus, given a packet loss probability p_{loss}, we need to *at least* reduce our rate by at least this much. However, this rate reduction does not make any provision for fairness. In order to allow TCP sessions to successfully contend for link capacity, we reduce the rate by an additional constant factor c. We empirically determined that TCP competes effectively with CTP when $c = 2$, as we report in Section 6.4.

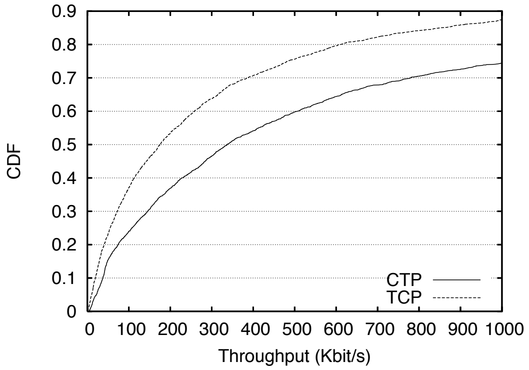

Figure 10: CDF of encounter-averaged throughput achieved using CTP and TCP respectively.

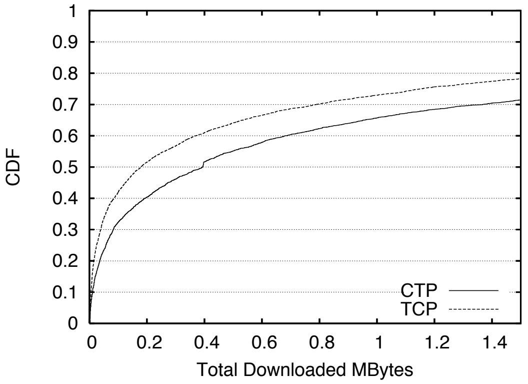

Figure 11: CDF of downloaded bytes per access point encounter.

6.3 CTP vs TCP Performance

To determine the utility of CTP's rate control measures, we compare the throughput achieved by CTP to TCP for downloads to cars. Running CTP and TCP simultaneously on the same AP does not produce the desired result. Instead, we designed our experiment to alternate between CTP and TCP for every established connection. The TCP implementation used was the default settings in Linux 2.6.17.13, with SACK enabled. Figure 10 shows the CDF of the encounter-averaged throughput achieved with CTP and TCP. We note that the measured median throughput achieved by CTP is approximately 2× that of TCP. The mean CTP throughput of 760 kbit/s was somewhat less than 2× the TCP throughput of 408 kbit/s.

The total number of bytes downloaded in an encounter is a function of the throughput as well as the duration of the encounter. Thus, throughput and bytes downloaded need not have a one-to-one correspondence. Figure 11 shows the CDF of the number of bytes downloaded per encounter. We note that the median CTP download is approximately 2× the median TCP download. However, the mean CTP download of 2438 kbytes is only 35% higher than the mean TCP download of 1860 kbytes.

The explanation for this phenomenon is simple: long-duration encounters have significantly lower packet loss rates than short-duration encounters. Figure 12 plots the average packet loss rate vs. encounter duration. It is clear that the majority of the very high loss rates are contributed by relatively short-lived connections, with relatively low (5%) loss rates for encounters 32 seconds and longer. Thus, on long duration encounters, TCP is able to achieve nearly the same throughput as CTP. This combination of long duration and high throughput explains the disparity between the mean throughput and download values.

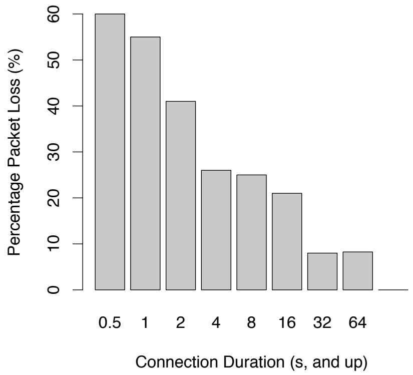

Figure 12: Avg. packet losses vs. encounter duration. Longer encounters on average exhibit lower packet drop rates.

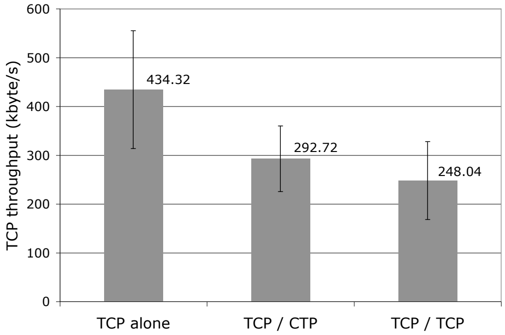

Figure 13: Mean throughput achieved by TCP in three scenarios: alone, competing with a CTP flow, and competing with another TCP flow. Error bars indicate standard deviation.

6.4 Interacting with TCP

To evaluate the "TCP friendliness" of the CTP rate adaptation algorithm, we ran a set of experiments from inside a home. Wireless packet losses were minimized by placing the node near the AP. We then measured the downstream throughput of one TCP flow, under three different conditions: (a) the measured TCP flow was the only active flow on the local network; (b) the measured TCP flow was competing with a downstream CTP flow; (c) the measured TCP flow was competing with a second downstream TCP flow.

In all cases, the measured TCP flow was established 20 times, each performing a 10 MB download. In cases (b) and (c), the competing flows were established before the start of the first measured TCP flow, and ended after the end of the last TCP flow. In case (b), we manually verified that the CTP flow was saturating the link before the start of the first TCP flow.

Figure 13 illustrates the result of these experiments. We note that the measured TCP flow achieves somewhat higher throughput when competing with CTP than when it is competing with another TCP flow. In addition, the standard deviation is lower when the measured

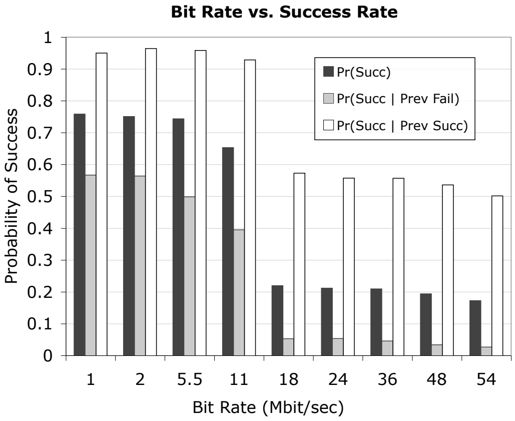

Figure 14: Probability of a successful packet transmission, for varying transmission rates.

TCP flow is competing with CTP than when it is competing with another TCP flow. A more extensive study of CTP interactions with TCP is a subject for future work.

7 Vehicular WiFi Rate Selection

In IEEE 802.11 networks, the sender may choose a bit rate from amongst a list of rates supported by both sender and receiver. Hence, when downloading data to cars in Cabernet, the AP controls the rate. Unfortunately, the bit-rate adaptation protocols used by most APs are optimized for stationary users, and even the best current rate adaptation schemes require multiple seconds to settle on the optimal rate, which makes them unsuitable for Cabernet.

Since we cannot control the rate selection scheme used for downloads, we ran a series of experiments to measure the effect of bit-rate selection on uploads. We believe that these results would apply to downloads as well if APs had support for vehicular clients. Specifically, our experiments measure the loss rate of uploads from moving vehicles transmitting at different bit rates. 802.11b supports 4 wireless transmission rates, 1, 2, 5.5 and 11 Mbit/s, and 802.11g (OFDM) adds several more rates, ranging from 6 through 54 Mbit/s.

During an encounter, we periodically sent bursts of 10 1000-byte frames to the AP, and recorded whether an acknowledgment was received, and after how many retries. Each frame was sent at a rate chosen uniformly at random from the supported rates advertised by the access point. The experiment was run on 10 vehicles, over the course of 2338 encounters with 531 unique access points. After initial experimentation showed that they performed significantly worse than the 11 Mbit/s rate, we did not include the 802.11g rates 6, 9 and 12 Mbit/s.

Figure 14 shows the results of this experiment. For each rate, the overall probability of success is shown, as well as the conditional probability of success given that the previous attempt (at any rate) was a success or failure. It is immediately apparent that 802.11b transmissions were much more likely to succeed than the faster 802.11g rates. However, somewhat surprisingly, the difference in loss rates between 802.11b bit rates was small. These measurements suggest that using 1 Mbit/s is never a good choice in Cabernet[2]. The time required to send a large packet at 1 Mbit/s is almost 10x the time required at 11 Mbit/s, negating any reliability benefit that could be had by switching to the lower rate. Moreover, looking at the conditional success probability given a previous failure (the light grey bar), we see that even after losing packets, reducing the rate would not result in a performance gain. It's unclear if 802.11g rates are ever a good idea—though they succeed 2-3 times less often, rates are high enough that they may still offer an overall performance gain. However, adding the extra overhead of retransmissions, backoff periods etc, it appears that any benefit of using the 802.11g rates would be marginal. It is unclear to us why the OFDM rates of 802.11g perform so much worse than the 802.11b rates in this setting. Signal strength alone does not appear to fully explain the phenomenon, as the 11 Mbps QPSK rate is frequently reported with a very similar receiver sensitivity as the 18 Mbps OFDM rate. The experiments in Section 9 show that adding an external antenna can improve signal quality by around 10 dB. Determining whether an external antenna would allow OFDM to operate reliably in a vehicular setting is a topic of future work.

In summary, current rate adaptation schemes are highly useful for stationary users, where signal quality varies slowly. However, in the case of Cabernet, the combination of rapid mobility and marginal channel quality means that a suitable adaptation scheme must adapt very quickly to realize a performance gain. Because the 11 Mbit/s rate overall performed so much better than the other rates, Cabernet currently uses only that rate. Based on these findings, we implemented a "fixed 11 Mbit/s" rate selection strategy on our taxi testbed. We use this modified rate selection protocol throughout the experiments reported in this paper. The full benefits of this optimization cannot be seen, however, because most of our experiments focus on download performance from the Internet to cars, where the (unmodified) APs control the bit rate.

8 End-to-End Performance

Having established the performance of QuickWiFi and CTP in isolation, we now turn to the overall performance of the implemented system. The experiments described below seek to answer the following key questions and explain observed performance:

1. What is the expected time between a request for information, and the retrieval of the desired response?
2. Given WiFi conditions in the Boston area, what is the expected time-averaged throughput of a Cabernet-equipped vehicle?

8.1 Expected Transaction Time

For applications that react to external stimuli, such as passing a landmark, a timer expiring, or a user action, the expected time taken to complete a network transaction is of interest. We envision a usage scenario where an application requests a download of a given size, at an arbitrary time. We then compute the time it would take Cabernet to complete the download.

Figure 15 shows the expected response time t_r, for a varying transaction size s. The median response time for a small (10 kbyte) request was 2 minutes, and the mean was 5 minutes. Typical requests of this nature may include traffic and weather updates, requesting driving directions, polling RSS feeds, and checking for new email. For larger transfers, such as downloading a full web page, large emails with attachments, or a 1 minute audio clip, we consider a 1 Mbyte request. The median time for such requests was 4 minutes, and the mean time was 9 minutes.

[2] Of course, it is possible that loss rates are not symmetric, but we do not believe that they should be dramatically different.

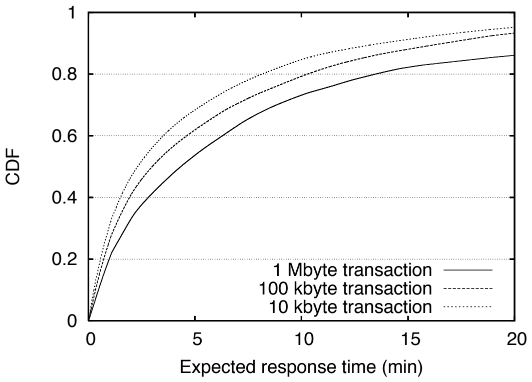

Figure 15: CDF of expected transaction time, given several transaction sizes.

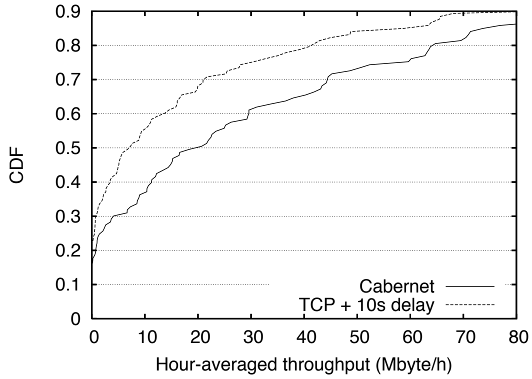

Figure 16: CDF of the expected number of bytes downloaded, per hour of driving, per car, with and without Cabernet.

Thus, while Cabernet is not suitable for a fast-paced browsing session, moderately interactive usage is feasible. Moreover, Cabernet is well suited to applications which perform periodic background updates as often as every few minutes.

8.2 Expected long-term Throughput

For less time-sensitive applications, such as podcasts, prefetching of web pages, or background file synchronization, the metric of interest is total download capacity, rather than transaction time. As we demonstrate below, Cabernet provides ample throughput for such applications.

We divide the driving time of each car into hour-long segments, and compute the total download throughout each time segment. Figure 16 shows the CDF of the available hour-averaged throughput when using Cabernet. For comparison, we also show TCP throughput after reducing (via post-processing) the total download time of each encounter by an *estimated* connection establishment delay of 10 seconds, to account for the improvement in connection establishment time due to QuickWiFi. In conclusion, we find that the mean throughput for Cabernet in the Boston area was 38 Mbyte/hour (86 kbit/s).

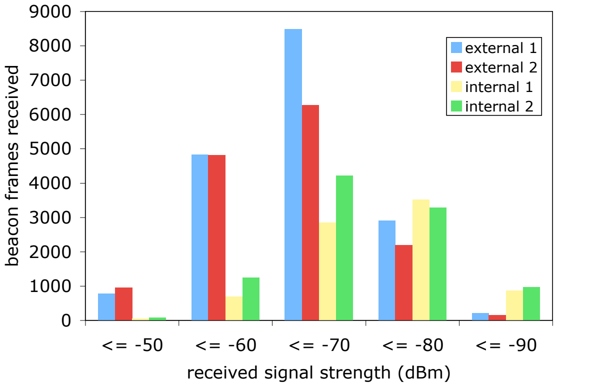

Figure 17: Received signal strength vs. antenna choice. Often difficult to install, an external high-gain antenna can dramatically improve reception.

9 Effect of Antenna Type and Placement

In our deployment, use of antennas on the exterior of the vehicle was not possible, and space constraints restricted the size of the antenna inside the vehicle. Hence, to better understand how the type and placement of the antenna affect Cabernet performance, we performed a small set of controlled experiments, using a single car, driving a well-defined route. The same route was driven four times, twice with the 3 dBi antenna on the dashboard, and twice with an additional 8 dBi antenna on the roof of the car.

All correctly received frames were recorded, and the subset of access points that were observed during all four drives was computed. Figure 17 shows the number of beacon frames received from these access points, in 5 signal strength buckets. We observed that the external antenna configuration consistently received a 10+ dB stronger signal, and also received substantially more packets.

This improved signal quality also resulted in improved end-to-end performance. For these drives, the mean time between encounters was 35 seconds for the external antenna configuration, and 44 seconds for the internal antenna configuration. Median time between encounters were 18 and 26 seconds respectively. The median connection duration changed from 2.75 seconds to 3 seconds, not enough to be significant. Apparently, the median connection duration did not improve with the high-gain external antenna. This phenomenon is explained by the fact that the external antenna configuration could communicate with more APs overall, with the majority of the new encounters (to more remote APs) having a short duration.

Finally, we measured the throughput achieved by CTP over the course of these drives. Here, the both the mean and median transfer per encounter was smaller with the external antenna than without. However, due to the larger total number of encounters, CTP averaged 14 megabytes per drive with the external antenna, compared to slightly less than 8 megabytes without the external antenna.

Despite the small number of drives used to collect this data, we guardedly draw a few conclusions from these experiments. First, an external antenna substantially improved signal quality, by about 10 dBi. The primary consequence of this improved signal quality was that the number of potentially usable APs (open and closed) increased by 40% (from 950 to 1330), accompanied by a corresponding decrease in the time between open AP encounters. Many of these potential new encounters are of short duration and poor quality.

10 Conclusion

This paper described Cabernet, a system for delivering data to moving vehicles. Cabernet uses WiFi access points encountered during drives for network connectivity. This intermittent connectivity presents several challenges, including high connection establishment latency and high WiFi loss rates.

We designed Cabernet to address these issues, evaluating our system on a real-world taxi testbed in the Boston area. To reduce connection establishment time, we built QuickWiFi, an optimized and integrated collection of tools for establishing connections with wireless access points. It is able to connect in just 366 ms on average, improving significantly on previous work. It also notifies applications when connectivity appears or disappears, simplifying application design. To improve throughput, we developed CTP, a transport protocol that handles high non-congestive wireless loss rates by running a lightweight probing protocol between a sender and the access point to isolate congestion events on the Internet path from last-hop losses. CTP is able to achieve double the throughput of TCP over paths with high non-congestion losses, with a mean throughput of 800 kbit/s when connectivity is present. Finally, in an end-to-end performance evaluation, we find that Cabernet is able to achieve an end-to-end throughput of 38 megabytes/hour (86 kbit/s) per car during its drives—ample throughput for a large class of non-interactive vehicular applications.

These performance results are encouraging, but what of wider deployment? Using open APs without the owners' explicit consent can be sensitive, no matter how short the duration of a connection. For wider use, incentives for access point owners to "open up" and opt-in are needed. A recent promising model is Fon's, where users open their access point to all Fon users that also open theirs. This model is becoming popular, and is now supported by established ISPs such as British Telecom, Time Warner, and Neuf[3]. In addition to privately operated access points, emerging wireless mesh networks as well as large-scale WiFi networks such as those operated by T-Mobile and others offer good coverage in many parts of the world. Cabernet provides a beneficial way to use these WiFi networks from moving vehicles.

Acknowledgments

We thank Brad Karp for carefully reading and critiquing multiple drafts of this paper, and the anonymous reviewers for their comments. We thank PlanetTran for their on-going support of our vehicular testbed. This work was supported by the National Science Foundation under grants CNS-0205445 and CNS-0520032, and the T-Party Project, a joint research program between MIT and Quanta Computer Inc., Taiwan.

[3] http://www.techcrunch.com/2007/10/04/fon-inks-deal-with-british-telecom/ and http://blog.wired.com/business/2007/10/fon-partners-wi.html

References

[1] A. Bakre and B. R. Badrinath. Handoff and System Support for Indirect TCP/IP. In *Proc. Second Usenix Symp. on Mobile and Location-Independent Computing*, Apr. 1995.

[2] H. Balakrishnan and R. Katz. Explicit Loss Notification and Wireless Web Performance. In *Proc. 3rd Global Internet Mini-Conference in conjunction with IEEE Globecom*, Nov. 1998.

[3] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. Katz. A Comparison of Mechanisms for Improving TCP Performance over Wireless Links. *IEEE/ACM Transactions on Networking*, 5(6), Dec. 1997.

[4] A. Balasubramanian, Y. Zhou, W. B. Croft, B. N. Levine, and A. Venkataramani. Web search from a bus. In *CHANTS '07: Proceedings of the second workshop on Challenged networks, CHANTS*. ACM, 2007.

[5] N. Bansal and Z. Liu. Capacity, delay and mobility in wireless ad-hoc networks. In *INFOCOM*, 2003.

[6] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks. *ACM Computer Communications Review*, 27(5), Oct. 1997.

[7] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine. MaxProp: Routing for Vehicle-Based Disruption-Tolerant Networks. In *INFOCOM*, 2006.

[8] V. Bychkovsky, B. Hull, A. K. Miu, H. Balakrishnan, and S. Madden. A Measurement Study of Vehicular Internet Access Using In Situ Wi-Fi Networks. In *12th ACM MOBICOM Conf.*, September 2006.

[9] R. Caceres and L. Iftode. Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments. *IEEE Journal on Selected Areas in Communications (J-SAC)*, 13(5), June 1995.

[10] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proc. ACM SIGCOMM*, pages 27–34, 2003.

[11] R. Gass, J. Scott, and C. Diot. Measurements of In-Motion 802.11 Networking. In *Proc. WMCSA*, Apr. 2006.

[12] S. Giordano, D. Lenzarini, A. Puiatti, and S. Vanini. Enhanced DHCP client. In *CHANTS*. ACM, 2007.

[13] D. Goodman, J. Borras, N. Mandayam, and R. Yates. Infostations: A new system model for data and messaging services. *In Proc. IEEE Vehicular Technology Conference*, pages 969–973, May 1997.

[14] L. A. Grieco and S. Mascolo. Performance evaluation and comparison of Westwood+, New Reno and Vegas TCP congestion control. *ACM CCR*, 34(2), Apr. 2004.

[15] D. Hadaller, S. Keshav, T. Brecht, and S. Agarwal. Vehicular Opportunistic Communication Under the Microscope. In *ACM MobiSys*, June 2007. To appear.

[16] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *Proc. ACM SenSys*, Nov. 2006.

[17] A. Iacono and C. Rose. Bounds on file delivery delay in an infostations system. In *Vehicular Technology Conference*, 2000.

[18] S. Jain, K. Fall, and R. Patra. Routing in a Delay Tolerant Network. In *ACM SIGCOMM*, 2004.

[19] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *Proc. Architectural Support for Programming Languages and Operating Systems*, October 2002.

[20] U. Kubach and K. Rothermel. Exploiting location information for infostation-based hoarding. In *MOBICOM*, pages 15–27, 2001.

[21] J. Lebrun, C.-N. Chuah, D. Ghosal, and M. Zhang. Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks. In *IEEE Vehicular Tech. Conf.*, pages 2289–2293, 2005.

[22] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, and A. Corradi. Mobeyes: smart mobs for urban monitoring with a vehicular sensor network. *IEEE Wireless Communications*, 13(5), Oct. 2006.

[23] R. Mahajan, J. Zahorjan, and B. Zill. Understanding WiFi-based connectivity from moving vehicles. In *IMC*, 2007.

[24] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links. In *Proc. ACM Mobicom*, July 2001.

[25] J. Ott and D. Kutscher. A Disconnection-Tolerant Transport for Drive-thru Internet Environments. In *INFOCOM*, 2005.

[26] A. Seth, P. Darragh, S. Liang, Y. Lin, and S. Keshav. An Architecture for Tetherless Communication. In *DTN Workshop*, 2005.

[27] T. Small and Z. J. Haas. The shared wireless infostation model: A new ad hoc networking paradigm (or where there is a whale, there is a way). In *MOBIHOC*, pages 233–244, 2003.

[28] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang. Study of a Bus-Based Disruption Tolerant Network: Mobility Modeling and Impact on Routing. In *Proc. ACM Mobicom*, pages 195–206, September 2007.

[29] W. Zhao, M. H. Ammar, and E. W. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *MobiHoc*, pages 187–198, 2004.