Spider: Improving Mobile Networking with Concurrent Wi-Fi Connections

Hamed Soroush* Peter Gilbert† Nilanjan Banerjee△
Mark Corner* Brian Neil Levine* Landon Cox †

*Dept. of Computer Science, Univ. of Massachusetts Amherst, {*hamed,brian,mcorner*}*@cs.umass.edu*

△ University of Arkansas, Fayetteville *nilanb@uark.edu*

† Dept. of Computer Science, Duke Univ., {*gilbert,lpcox*}*@cs.duke.edu*

ABSTRACT

We present an in-depth analysis of the performance of attempting concurrent AP connections from highly mobile clients. Previous solutions for concurrent Wi-Fi are limited to stationary wireless clients and do not take into account a myriad of mobile factors. Through an analytical model, optimization framework, and numerous outdoor experiments, we show that connection duration, AP response times, channel scheduling, available and offered bandwidth, node speed, and `dhcp` *joins all affect performance. Building on these results, we design, implement, and evaluate a system, Spider, that establishes and maintains concurrent connections to 802.11 APs in a mobile environment. The system uses multi-AP selection, channel-based scheduling and opportunistic scanning to maximize throughput while mitigating the overhead of association and* `dhcp`*. While Spider can manage multiple channels, we empirically demonstrate that it achieves maximum throughput when using multiple APs on a single channel. Our evaluation shows that Spider provides a 400% improvement in throughput and 54% improvement in connectivity over stock Wi-Fi implementations.*

1. INTRODUCTION

As wide-area cellular networks continue to strain under the heavy load of data access, they are increasingly reliant on offloading traffic to private, public, and openly shared Wi-Fi networks. Most smart-phones are equipped with both Wi-Fi and cellular data and prioritize Wi-Fi over cellular to take advantage of high bandwidth, lower latency, and higher per-bit energy efficiency. While offloading traffic has been used by stationary users, recent measurement work has shown that it is also beneficial to offload mobile users' data [2].

To realize the fullest extent of these gains, Wi-Fi systems can aggregate a large number of access points *concurrently* to achieve improved network characteristics, unlike cellular, where devices are relegated to a single access point assignment. Recent *virtualized* Wi-Fi systems, such as Chandra et al. [5], FatVAP [11], and Juggler [18], have shown that stationary users connected to multiple access points can achieve up to $3x$ greater bandwidth than users connected to a single access point [11]. These systems work by switching between access points on *multiple channels* rapidly, aggregating bandwidth at the client.

Unfortunately, such multi-AP systems were built to support only nomadic access and not truly mobile use. These systems employ scheduling policies that assume that the `dhcp` join has already completed and that the user's connection to the AP is without end. Thus, they perform poorly in a mobile context, producing a schedule that results in significantly lower performance.

The `dhcp` delays in mobile systems are relatively long and not handled by power-save mode, requiring any virtualized Wi-Fi system to dwell on a channel to wait for `dhcp` responses. As we show in this paper, through analytical study, measurement, and experimentation, the dwell times required to obtain `dhcp` leases from APs can swamp TCP round trip times, creating TCP timeouts, resulting in greatly reduced bandwidth. Our results show that at higher speeds, mobile users receive better performance by connecting to multiple APs only if they appear on the same channel. Only at lower speeds can mobile users recover from the throughput loss resulting from `dhcp` joins to APs on separate channels. While these two extremes are clear, the breaking point between them is not predicted easily. One of our contributions is a general model of this problem that isolates the critical factors that determine an optimal schedule using one or more channels. These factors include the user's speed, the AP's `dhcp` response time, the AP's offered bandwidth, and the attained bandwidth. For example, in a typical environment, our model suggests that users that travel with an average speed of 10 m/s (∼22 mph) or faster should form concurrent Wi-Fi connections only within a single channel. We also show empirically that link-layer association, `dhcp` and, TCP performance are affected negatively by multi-channel solutions.

Building from the results of our analytical study, we design and implement a practical version of a mobile, virtualized Wi-Fi system called Spider that is designed

for high-speed mobile users. Spider solves the practical issues of access point discovery and `dhcp` lease acquisition while optimizing bandwidth using a single wireless channel. Through a set of extensive outdoor experiments we show that we can maximize bandwidth using multiple APs on a single wireless channel, achieving 122KBps, which is more than 400% improvement over a multi-channel approach. These results present Spider as an effective supplement to cellular networks for highly mobile clients.

We also investigate empirically tradeoffs between throughput and connectivity. Spider can be used to manage and schedule joins to APs on multiple channels, and at a serious penalty to achieved bandwidth as our model predicts. We demonstrate that if connectivity is a priority, then joining to multiple APs on multiple channels is best: Spider achieves 44% connectivity (compared to 35% in the single-channel case), lowering bandwidth to 28KBps.

2. CHALLENGES OF MOBILE MULTI-AP CONNECTIONS

Past research has shown the benefits of both mobile Wi-Fi systems [2,8] and virtualized multi-AP bandwidth aggregation schemes in static wireless scenarios [5,11,18]. In this work, we investigate whether a multi-AP solution is possible and beneficial in *truly* mobile scenarios where Wi-Fi connections are both fleeting and intermittent.

An obvious starting point would be to take one of the existing virtualized multi-AP solutions and apply it in a mobile setting. However, a closer look at the dynamics of both static multi-AP solutions and mobile Wi-Fi networks shows that the integration of the two is not straightforward.

Since APs can be instructed by the client to buffer packets, concurrent connections between a static client and multiple APs [5,11,18] are possible for Wi-Fi. The client falsely claims it is entering *power-save mode* (PSM), implicitly asking the AP to queue the incoming packets, and then communicates with another AP. Given that the backhaul bandwidth is typically smaller than the wireless bandwidth, such a scheme results in higher aggregate throughput if switching delays are kept very short. Static multi-AP solutions are not concerned with the delay incurred by the process of joining to the APs, and they do not need to be. In a static scenario joining process (association and `dhcp`) happens once, and its duration is negligible compared to the total connection time.

In a mobile Wi-Fi environment, on the other hand, clients must continuously associate and obtain `dhcp` leases from APs as they become available. In addition, the packets associated with the join process cannot be buffered by the PSM request, and therefore, the client cannot switch away without reducing its chances of getting a `dhcp` lease.

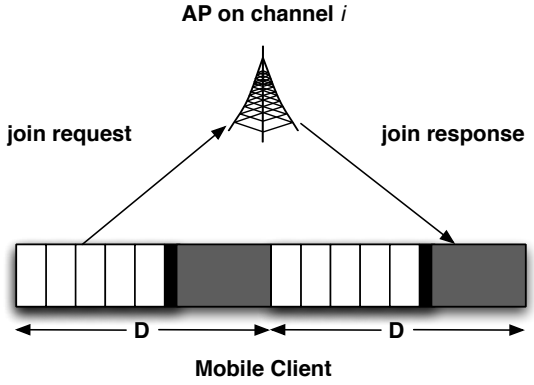

Figure 1: Depiction of a join failure due to untimely arrival of the join response message in our simplified scenario. White regions represent the fraction (f_i) of the scheduling period (D) spent on channel i to communicate with the AP. Gray regions depict the rest of the scheduling period spent away from the AP on another channel. Switching delay (w) is shown in black. For simplicity, the model assumes that join requests are sent in the beginning of intervals of length $\lceil \frac{Df_i}{c} \rceil$ where c is a constant. Join process becomes more complicated in reality, where it involves multiple steps to associate and obtain a `dhcp` lease instead of one.

To understand the effect of mobility on the performance of multi-AP systems, we design and validate a model that predicts the probability of obtaining a `dhcp` lease from an AP as a function of the amount of time spent in range. Based on this model, we then formulate an optimization framework to determine schedules that maximize aggregate throughput.

Our analytical framework allows us to find the *dividing speed* above which a mobile multi-AP solution should consider APs only on a single channel for throughput maximization. However, the framework is limited since it does not consider the complexities of the multi-phase algorithms running link-layer association, `dhcp`, and TCP. In order to isolate the effects of multi-channel switching on these protocols, we also perform a series of real experiments (Section 2.2). The bottom line of the model, optimization framework, and empirical experiments is that switching is detrimental to overall throughput for almost any short-lived connection. This conclusion is further supported by a series of extensive experiments in Section 4 using a complete system in an outdoor setting.

2.1 Analytical Framework

We have designed our analytical framework with two key questions in mind: *First, how does channel switching and node speed affect the time to associate and obtain a* `dhcp` *lease?* And *second, at vehicular speeds, when is it desirable to aggregate throughput from APs on multiple*

channels?

2.1.1 Join Model

We model the probability of a mobile node successfully joining an AP operating on channel i during the first t seconds that it is in range. We assume a round robin schedule, where the node spends a fraction of time f_i on channel i from the total scheduling duration of D. To switch channels, the node incurs a delay w during which no packets can be received. We approximate $t \simeq s \times D$, where s is a positive integer reflecting the speed of the mobile node (note that s is larger when the node is moving more quickly). We refer to each of the s time intervals as a *round*. In sum, in each round, a duration of Df_i is spent on channel i. For simplicity, we assume that the mobile node switches to channel i as soon as it enters the range of the AP.

While in practice, a Wi-Fi join event consists of several complimentary steps, for simplicity we assume that association involves a single handshake. Further, we assume that in the absence of losses, the time between the transmission of the single join request and the reception of the single join response in a non-virtualized scenario is uniformly distributed in the interval $[\beta_{min}, \beta_{max}]$. During a round, the mobile node can transmit a maximum of $\lceil \frac{Df_i}{c} \rceil$ join requests to the AP, where c is the time between two consecutive requests. In practice, the duration c is determined by `dhcp` and link-layer timeout values, and we set it to a constant in the model. We let $\beta_{m,k} \in [\beta_{min}, \beta_{max}]$ denote the *join time* corresponding to a request transmitted in segment k of round m, where $1 \leq k \leq \lceil \frac{Df_i}{c} \rceil$. We then compute the probability that a request made at the beginning of segment k in round m leads to a successful join. Note that for a successful join, the response from the AP must be received at the mobile node within a time interval Df_i of the current or later round (see Figure 1). The above constraint can be mathematically formulated as follows.

$$w + (k-1)c + \beta_{m,k} \leq (n-m)D + Df_i \quad (1)$$
$$w + (k-1)c + \beta_{m,k} \geq (n-m)D \quad (2)$$

These can be combined into one equation as

$$(n-m)D + c - w \leq kc + \beta_{m,k} \leq (n-m+f_i)D + c - w \quad (3)$$

where $n \geq m$ is the round in which the response is received. Since $\beta_{m,k} \in [\beta_{min}, \beta_{max}]$, we have

$$kc + \beta_{min} \leq kc + \beta_{m,k} \leq kc + \beta_{max} \quad (4)$$

Using the above equations, we can derive the probability that a request sent during segment k of round m leads to a successful join for a lossless channel as

$$q(m,n,k) = \begin{cases} 0 & \text{if } \delta^{min}_{m,n} > \alpha^{max}_k \\ 0 & \text{if } \delta^{max}_{m,n,fi} < \alpha^{min}_k \\ \frac{\min\{\alpha^{max}_k, \delta^{max}_{m,n,fi}\} - \max\{\alpha^{min}_k, \delta^{min}_{m,n}\}}{\alpha^{max}_k - \alpha^{min}_k} & \text{otherwise} \end{cases} \quad (5)$$

where

$$\begin{aligned} \alpha^{min}_k &= kc + \beta_{min} \\ \alpha^{max}_k &= kc + \beta_{max} \\ \delta^{min}_{m,n} &= (n-m)D + c - w \\ \delta^{max}_{m,n,f_i} &= (n-m+fi)D + c - w \end{aligned}$$

Hence, the probability that *no* requests made in round m leads to a successful join in round n in a lossy channel with message loss probability h can be calculated from the previous equation as

$$\overline{q(m,n,h)} = \prod_{k=1}^{\lceil \frac{Df_i - w}{c} \rceil} \left(1 - q(m,n,k)(1-h)^2\right) \quad (6)$$

Thus, given that the mobile node spends a fraction f_i of time on a channel i, the probability of obtaining at least one lease in time t is the following:

$$p(f_i, t) = 1 - \prod_{m=1}^{\lfloor \frac{t}{D} \rfloor} \prod_{n=m}^{\lfloor \frac{t}{D} \rfloor} \overline{q(m,n,h)} \quad (7)$$

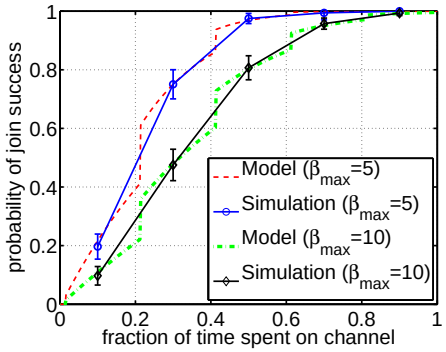

Figure 2: The probability of join success as a function of the fraction of time, f_i, spent on the AP based on the presented model (Eq. 7) and simulation. Mobile node with a scheduling period of $D = 500ms$ spends $t = 4s$ in the vicinity of the AP with $\beta_{min} = 500ms$, $\beta_{max} = 5s$ and $10s$. Switching overhead of the driver is $w = 7ms$. Join requests are sent every $c = 100ms$ and loss rate is $h = 10\%$.

Validation. We validate our analytical model by using a simulation with the same assumptions. Figure 2 shows the probability of successfully joining an access point as a function of the fraction of time spent on the channel for both the model (Eq. 7) and the simulation. Each

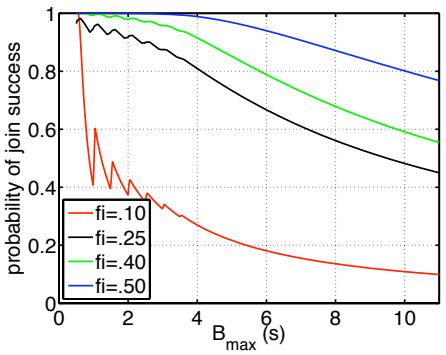

Figure 3: The probability of join success as a function of the maximum time it takes the AP to respond. When a fixed fraction of time is spent on the channel, shorter maximum join times lead to higher chances of join success. Mobile node with a scheduling period of $D = 500ms$ spends $t = 4s$ in the vicinity of the AP with $\beta_{min} = 500ms$. Switching overhead of the driver is $w = 7ms$. Join requests are sent every $c = 100ms$ and loss rate is $h = 10\%$.

point on the simulation line represents an average of hundred runs, and error bars represent one standard deviation. Each run uses a different seed and represents 100 trials where a join is attempted and the value of $\beta_{m,k}$ is sampled from the given distribution. We set other input parameters to typical values we observed in practice (see Section 2.2). The simulation results are statistically equivalent to the model and hence, internally validate it.

2.1.2 Discussion

Eq. 7 represents a non-linear relationship between the fraction of time spent on the channel, f_i, and the probability of a successful join $p(f_i, t)$. For instance, in Fig. 3, the probability of getting a lease during the first $t = 4$ seconds falls from 75% to 20% when the percentage of time devoted to the AP reduces from 30% to 10%. Further, the node should spend nearly 100% of its time on the channel for an *assured* successful join. These results motivate a *channel* switching approach to schedule concurrent Wi-Fi connections in mobile scenarios as opposed to the *AP* switching approach used in static cases [11]. Spending all its time on one channel, the mobile node can aggregate bandwidth from several APs on that channel without affecting the probability of successful joins.

Another implication of our analysis is the relationship between the maximum join time, β_{max}, and the probability of a successful join $p(f_i, t)$. Figure 3 graphs the value of $p(f_i, 4)$ with β_{max} as the independent variable for four different values of f_i, using Eq. 7 . Note that β_{max} represents the maximum amount of time to join to an AP in the absence of losses and in a non-virtualized scenario. When a fixed fraction of time is spent on the channel, shorter maximum join times lead to higher chances of a successful join. While this is a known fact for single-AP scenarios, our results demonstrate that techniques such as caching `dhcp` leases, maintaining a history of APs with short join times, and decreasing link layer timeouts that reduce β_{max} are essential for multi-AP systems. In other words, when these techniques are not available — when mobiles travel in areas they do not normally do and AP responses are slow — multi-AP systems will see significant drops in performance.

2.1.3 Throughput Maximization

A major objective of maintaining concurrent connections to multiple access points is bandwidth aggregation. To understand how channel schedules and node speed affect bandwidth aggregation, we formulate an optimization framework for throughput maximization.

The objective function and the associated constraints are shown in Equation 8. To construct it, we assume that the mobile node is in range of APs for T seconds. We let $E[X_i] = \sum_{t=0}^{T} p(f_i, t)$ be the expected amount of time to join an AP on channel i given f_i and T. We let B_w be the maximum bandwidth of each channel. We distinguish current and offered bandwidth: let B_j^i be the total end-to-end bandwidth from APs on channel i that the node has already joined to; let B_a^i be the end-to-end bandwidth available from APs that the node is attempting to join to and would have during the duration $(1 - E[X_i])T$. We let k be the number of available channels and we state the objective function as follows.

$$\max_{f_i} \left\{ T \sum_{i=1}^{i=k} (f_i B_w) \right\} \tag{8}$$

$$\text{s.t.} \qquad 0 \leq f_i \leq \frac{B_j^i + (1 - E[X_i])B_a^i}{B_w}, \ \forall i \tag{9}$$

$$\sum_{i=1}^{i=k} (f_i D + \lceil f_i \rceil w) \leq D \tag{10}$$

The formulation is similar to the FatVap optimization problem [11] except the constraint on f_i which considers the additional bandwidth gained from APs that the node is associating with. Note that for simplicity, we have assumed all APs are in range for a duration of T and thus have the same $E[X_i]$.

We numerically solve the above optimization to determine the optimal schedule as a function of the speed of the node. Specifically, we evaluate three scenarios for a two-channel case:

1. $B_j^1 = 0.75B_w$ and $B_a^2 = 0.25B_w$
2. $B_j^1 = 0.25B_w$ and $B_a^2 = 0.75B_w$
3. $B_j^1 = 0.50B_w$ and $B_a^1 = 0.50B_w$

Our goal is to determine the speeds at which it is optimal to switch channels, given a practical Wi-Fi range of 100

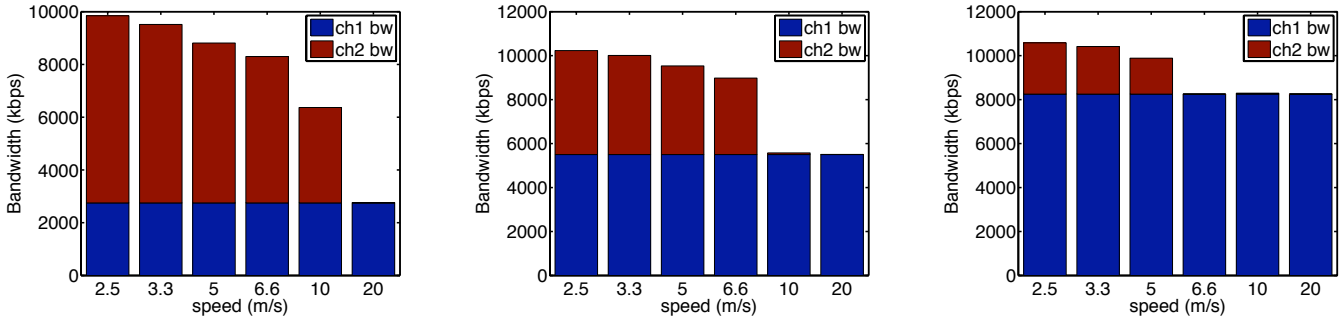

Figure 4: Maximum aggregated bandwidth for different speeds when only two channels are used. The offered bandwidth on the first and second channels are (25%,75%), (50%,50%), and (75%,25%) of the wireless bandwidth (B_w =11Mbps) respectively in each figure from left to right for $\beta_{max} = 10s$ and $\beta_{min} = 500ms$. Wi-Fi range is assumed to be 100m.

meters. Figure 4 shows the results for the three scenarios in terms of the optimal bandwidth that can be extracted from each channel. For every scenario, there is a dividing speed: when moving slower than the divide, the mobile node should switch channels to maximize bandwidth. Quantitatively, this speed is less than $10m/s$ for most scenarios. This result implies that for highly mobile networks (where the average node speed is greater than $10m/s$), the best policy to maximize bandwidth is to stay on a *single* channel. While we examined only three scenarios to produce Figure 4, we note that our model and optimization framework solves all combinations of inputs. We empirically validate and further explore this result in Section 4 using a mobile testbed.

2.2 Experimental Analysis

Our model shows that scheduling fractional time on several channels has an adverse effect on the probability of successfully joining APs (see Fig. 3). Additionally, for high vehicular speeds, maintaining concurrent connections to access points while staying on *one* channel leads to optimal bandwidth aggregation. However, our model makes two simplifying assumptions. First, it assumes that joining is a one-shot process, while in practice Wi-Fi joins involve a multi-phase bidirectional handshake involving both association and `dhcp`. Second, it assumes that the schedule is short enough so that TCP timeouts are avoided. The assumptions cause the model to be optimistic: multi-channel switching performs better in the model than can be expected in a real scenario.

To quantify the affect of switching on link-layer associations, `dhcp` and TCP, we performed two sets of experiments, one on an outdoor vehicular testbed and the other on an indoor wireless testbed. Our results quantify the relationship between channel schedule, join success, and TCP throughput. In sum, successful link-layer association and `dhcp` joins have some tolerance for switching, while TCP is more sensitive.

2.2.1 *Association and* `dhcp`

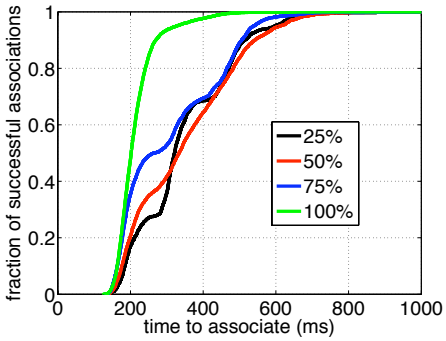

Figure 5: The rate of successful link-layer associations on a channel as a function of the amount of time the Wi-Fi driver spends on a single channel (out of 400ms for all channels).

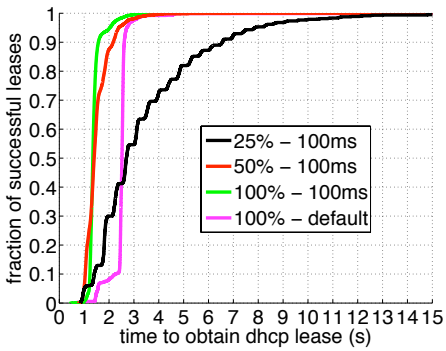

Figure 6: The rate of successful `dhcp` requests on a channel as a function of the amount of time the Wi-Fi driver spends on that channel as well as the `dhcp` timeout.

In our first set of experiments, we use a testbed of vehicular nodes and a Wi-Fi driver that can simultaneously associate with APs on different channels, just as we modeled above — in fact, the driver is the core of our full system, and we provide full details of its operation in Section 3. The goal of our outdoor experiments is to quantify the affect of varying the fraction of time

spent on each channel on the failure rate of associating and obtaining a `dhcp` lease. Moreover, these experiments allow us to quantify the surprisingly detrimental effect of decreased link-layer response timeouts on a multi-AP driver, an approach shown by Eriksson et al. [8] to improve single-AP/single-channel drivers.

To evaluate the effect of the driver's schedule on the join success rate, we performed several experiments, each lasting six hours on five vehicles moving around a small town, representing hundreds of trials. Each mobile node spends a fraction $f_6 = x$ of $D = 400ms$ on channel 6, and a fraction $f_1 = f_{11} = (1-x)/2$ on channels 1 and 11, applying terminology from our model and where $0 \leq x \leq 1$. Our results show the association delays on channel 6 and consider it as our *primary* channel. Since join delays are affected by link-layer timeouts, we reduced them from a standard of $1s$ to $100ms$ in these experiments[1].

Figs. 5 and 6 plot the empirical cumulative distribution functions for durations of association and `dhcp` respectively for these experiments. Separate conclusions can be drawn for association and `dhcp`.

Our analytical framework, which only evaluates a simple join-request scenario, predicts that join success is dependent on the channel schedule. However, these real experiments demonstrate that the performance of actual link-layer joins are more complicated given the interactions of scanning and the four-way handshake used. As shown in Fig. 5, when the driver spends all its time on one channel ($f_6 = 1$), the median association time is 200ms, and all associations complete within 400ms. However, when this fraction drops to $f_6 = 0.75$, the median association time increases to 300ms and only 75% of associations are successful within 400ms. Interestingly, this performance does not degrade significantly as f_6 decreases to 0.50 and 0.25 suggesting that link layer association is in some ways robust to switching.

`dhcp` is a more complicated protocol. It relies on successful association and involves at least four more frames between client and AP. Moreover, in default implementations, the client attempts to acquire a lease for 3 seconds, and it is idle for 60 seconds if it fails. The performance of this default scheme dedicated to a single channel is shown in Fig. 6 (as "100% default") with a median join time of $2.5s$. The figure shows that reducing both timeouts above to 100ms [8] has a significant effect on performance. For the same schedule of $f_6 = 100\%$, the median join time reduces to $1.3s$ when a $100ms$ `dhcp` timeout is used. Once the schedule is set to $f_6 = 25\%$ and $D = 400ms$, equal to the timeout, repeated failures cause the accumulated time to degrade performance once again. Hence, while reconfigured `dhcp` timers are a boost to performance, we could not make `dhcp` robust to low fractions of scheduled time. This result suggests that the driver's time cannot be divided among more than two channels at 50% each in a mobile setting where the duration of time in range of an AP is limited.

[1]Note that the link-layer timeout reflects a timer for each message in a multi-step protocol and not a timeout for the entire request-response process.

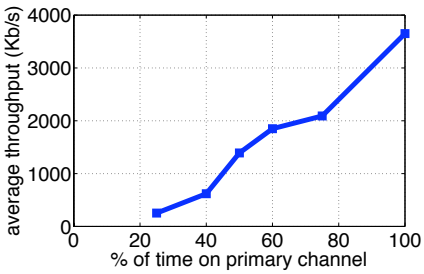

Figure 7: Average TCP throughput as a function of the *percentage of time* spent by the Wi-Fi driver on the primary channel. Since the cumulative time spent on all the channels is 400 ms (which is less than two RTTs) the throughput is proportional to the percentage of time spent on the primary channel.

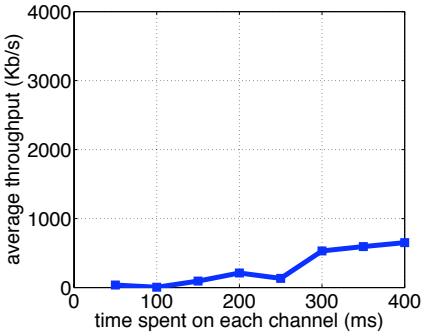

Figure 8: Average TCP throughput as a function of the *absolute time* spent on each channel. For time x spent on the channel, time $2x$ is spent away from it. The throughput is very sensitive to the amount of time spent by the driver on each channel due to TCP timeouts and TCP slow start.

2.2.2 *TCP performance*

Our throughput maximization framework assumes that the schedule does not lead to TCP timeouts. However, in a practical setting, if the channel schedule is skewed towards spending a large fraction of the time on a single channel, TCP connections on an orthogonal channel can timeout, potentially strangling performance. There is an inherent tension between the probability of successfully associating with APs on one channel and sustaining TCP connections on another channel.

To quantify this tradeoff, we performed a set of controlled experiments in an indoor setting. We configured an AP on one channel (the primary channel) and varied two parameters, the fraction of time spent on the pri-

mary channel for a fixed scheduling period of $D = 400ms$ and the *total scheduling time* equally distributed across channels 1, 6, and 11 ($f_1 = f_6 = f_{11} = 1/3$). When the fraction of time spent on the primary channel is varied for a 400 ms schedule (equal to two typical RTTs), the TCP throughput increases monotonically, as shown in Fig. 7. However, when the total scheduling time is varied instead, the throughput increase is non-monotonic, as shown in Fig. 8. This behavior is a result of increasing the total schedule which increases the amount of time *spent away* from the channel which can lead to TCP timeouts.

2.3 Main Result of Analysis

Our analytical model, throughput maximization framework, and experimental analysis point to the following conclusion: *At vehicular speeds, the best channel scheduling policy for throughput maximization is to spend all of the time on a single channel that provides maximal bandwidth.*

Several observations from our analytical and experimental framework substantiate this result. First, as shown in Figs. 2, 5, and 6, our model predicts that the probability of successfully joining to APs within a short time is high only when the node spends close to 100% of the time on the channel. Note that continuously associating with APs is mandatory in mobile scenarios to sustain connectivity, given short encounters at vehicular speeds (median of 8 s and average of 22 s in our town). Second, link-layer association, `dhcp` joins, and TCP throughput (shown in Figs. 5, 6, and 8) are all adversely affected if the card spends a large amount of time away from the channel. Therefore, when the mobile node spends a small amount of time in vicinity APs, it should aggregate bandwidth from one channel while simultaneously associating with APs on that channel.

3. SPIDER

Based on the results of our analytical framework and experimental analysis in Section 2, we have designed and implemented Spider, a system that leverages concurrent 802.11 connections to improve performance in highly mobile networks. Our implementation is a freely available, open source Linux kernel module[2].

3.1 Design

Our analytical framework suggests that leveraging multiple APs to maximize bandwidth in a highly mobile scenario requires a system that is different from multi-AP Wi-Fi drivers designed for static environments [5,11]. Hence, we have made several design choices in Spider that are different from stock virtual Wi-Fi drivers [18].

Design Choice 1: Channel-based switching. In contrast to previous work that slices time across *individual APs* [5, 11, 18], Spider schedules a physical Wi-Fi card among 802.11 *channels*. Spider maintains *one* packet queue per channel that is swapped in and out of the driver. As shown in the previous section, the time to join with access points can be long. Therefore, in an AP scheduling scheme such as FatVAP [11], the queue corresponding to an AP can reserve the driver for a long time. This implies that the card cannot communicate with other APs on the same channel, which as we discussed in Section 2 degrades performance. Per-channel queues mitigate the above problem since they allow the driver to communicate with *all* APs on the same channel simultaneously. Additionally, it incurs no switching overhead for interfaces on the same channel.

Design Choice 2: AP selection based on Join Success. A multi-AP solution in a highly mobile scenario requires a low-overhead AP selection algorithm. Unfortunately, selecting multiple APs while maximizing a given system utility function is NP-hard (see Appendix A for a proof). While an optimal dynamic programming approach can be formulated, its complexity increases exponentially in the size of the *power set* of access points, making a real-time solution infeasible in mobile scenarios where the node is within range of an access point for only a few seconds.

Spider uses a simple heuristic to select APs. The heuristic is driven by our observations in Section 2 that join times with APs is the critical factor for performance in highly mobile scenarios. Therefore, instead of choosing APs with maximum end-to-end bandwidth, we select APs that have the best history of successful joins.

Stated formally, each AP m is assigned a utility U_m which is a function of the number of successful *join* attempts the client has made with the AP previously. A successful *join* comprises of three steps: *i*) link-layer *association*, *ii*) `dhcp` lease acquisition, and *iii*) end-to-end *connectivity* testing. Spider assigns fixed values v_a, v_b, or v_c (where $v_a < v_b < v_c$) to each specific join attempt depending on how far it succeeds in the joining process. Joins that fail during link layer associations are assigned a zero value. The intuition behind such weighing of values is that APs that have completed the join process are more *reliable* than those that have only been successful in link-layer associations or obtaining a `dhcp` lease. The utility U_m of each AP m is the weighted average of previous and recent join attempts—the recent joins are given larger weights. To bootstrap the process, every new open AP that has sufficient signal strength is assigned the maximum utility so that the AP is considered for association at least once. Signal strength is used to break ties when APs have the same utility. Additionally, to reduce the time to join to the APs, Spider uses `dhcp` caches and fine-tunes link-layer and `dhcp` timeout values.

Design Choice 3: One Linux interface per AP.

[2]URL removed for anonymity.

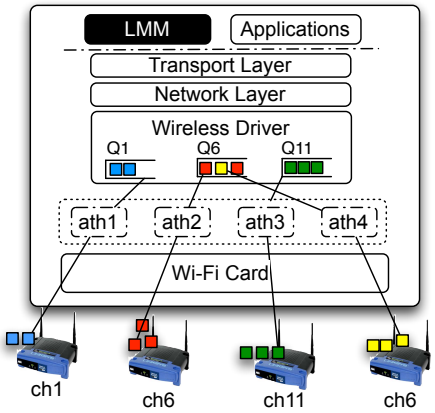

Figure 9: An illustration of Spider's components.

Spider exposes a separate Linux network device interface for each connection, allowing maximal flexibility in the way that applications may use concurrent Wi-Fi connections. Consequently, Spider does not require any custom modifications to the kernel socket data structure (unlike Juggler [18]). As a result, Spider remains a completely standalone loadable module that is compatible with off-the-shelf wireless utilities such as `iwconfig, iwlist, iwspy` and `iptables`, making link management and network debugging much easier.

3.2 Implementation

Spider consists of two components: a *custom Wi-Fi device driver* responsible for opportunistic scanning and channel switching, and a user-space *link management module* (LMM) that implements connection establishment and policies for channel switching and AP selection. The overall design of Spider is illustrated in Figure 9.

3.2.1 Wireless Driver

Spider is implemented as a modified MadWiFi driver that provides multiple "virtual" network interfaces. Spider's wireless driver schedules multiple APs on a single channel concurrently. It also supports opportunistic scanning, allowing new APs to be discovered in the background without sacrificing foreground data-transfers. The driver exposes configurable parameters such as the channel-switching schedule and link-layer timeouts to the link management module using a `proc` interface.

The driver follows a number of steps to switch between channels. First, it buffers outgoing packets on the virtual interfaces of the current channel. Next, it sends a management frame with the PSM bit set to each AP with which it is associated on the current channel, indicating that it is entering power-save mode. This causes the APs to buffer all frames destined for the client until it returns to the channel. After deactivating the virtual interfaces on the previous channel, the driver changes the state of the wireless card to the new channel. A hardware reset is required to apply the change. Finally, the set of interfaces that are associated or attempting to associate with APs on the new channel are activated.

Opportunistic Scanning. To maximize the time available for useful work, the driver scans opportunistically in the background without disrupting ongoing connections. While associated with an AP, a client often receives beacons and probe responses transmitted by other APs on the same channel. Spider accepts these frames and maintains a list of APs which it has heard from recently. Spider can also be configured to periodically broadcast probe requests.

3.2.2 Link Management Module

Spider's link management module is responsible for applying AP selection policies, managing concurrent connections, detecting lost connections and establishing new ones, as well as notifying applications of the availability of a link.

The link management module creates a configurable number of virtual interfaces on boot-up and sets an appropriate channel schedule akin to an *operation mode*. An operation mode is defined by the total amount of time to be scheduled among channels and the fraction of time spent on each channel. The link management module provides support for dynamically changing the schedule when needed. Since operation modes are configured from user-space, applications can easily change these settings without any modification to the driver or the linux kernel. We present results from experiments using different operation modes in Section 4.

After the initial setup phase, the link management module performs link discovery and management operations for that interface. Our synchronization mechanism ensures that no two interfaces are bound to the same AP. As soon as an interface joins a network and obtains an IP address, corresponding `iptables` rules are set to allow routing traffic to and from that specific interface. While in practice, IP address collisions are rare, if the same IP address is assigned to different virtual interfaces by different APs, we only use the most recently assigned interface. For each interface, the module selects the AP with the highest utility which is not already in use. In case of ties, signal strength is used as the final determinant.

Upon a successful link-layer association, per-BSSID `dhcp` caches are used to speed up the process of obtaining a lease. Additionally, after a successful join, Spider continuously uses end-to-end pings to determine whether the connection is alive. In case an AP does not allow `icmp` pings to propagate, Spider pings the gateway to test connectivity. If thirty consecutive pings fail (sent at a rate of 10 pings per second), Spider assumes that the connection is dropped, notifies applications using a shared flag resident on the system's RAM disk, and

tries associating with another AP on that interface.

4. SYSTEM EVALUATION

We intend Spider to complement cellular data services. Wi-Fi solutions like Spider are a natural supplement to cellular networks due to their higher *capacity*. Their drawback, however, is that they do not provide continuous connectivity by themselves.

Spider aims at improving throughput and connectivity for mobile clients using open Wi-Fi access by synergistically using multi-AP selection, channel switching, opportunistic scanning, and parallel per-channel association. Here, we evaluate the performance of Spider by focusing on the following key questions:

- What improvement in throughput and connectivity does Spider provide over stock Wi-Fi access?
- What is the effect of AP density on Spider's performance?
- Does Spider meet connectivity needs of common wireless users?

While answering these questions, we also present several micro-benchmarks and explore trade-offs available when setting different `dhcp` and link-layer timeouts.

4.1 Experimental Setup

We evaluated Spider on a vehicular platform in two different cities: our town and Boston, MA. We chose Boston because it was the site of Cabernet [8] experiments and is a major urban area.

During experiments in our town, almost all APs were on channels 1 (28%), 6 (33%), or 11(34%). Cabernet [8] reported comparable numbers for the Boston area with 83% of the APs on either of the three channels and 39% on channel 6. Since majority of APs dwell on these three channels, we configured Spider to schedule at most among these three channels.

We test four configurations of Spider. (1) *Single-channel, Single-AP:* Spider mimics off-the-shelf Wi-Fi on a single channel. (2) *Single-channel, Multiple-AP:* Spider stays on one channel (channel 1, 6, or 11) and joins to as many APs on the channel as possible. (3) *Multiple-channel, Multiple-AP:* Spider switches between the three orthogonal channels using static schedules. (4) *Multiple-channel, Single-AP:* Spider switches channels but is associated with one AP at a time. We also tested the unmodified MadWiFi driver as a point of comparison to configuration 1.

The equipment used in the experiments was comprised of a 1GHz Intel Celeron M system running Ubuntu Linux 2.6.18 and Atheros 802.11abg MiniPCI wireless card. The duration of each experiment was 30–60 minutes with the mobile node following the same route multiple times.

4.2 Driver Micro-benchmarks

	Num. of interfaces				
	0	**1**	**2**	**3**	**4**
Mean	4.942	4.952	5.266	5.546	5.945
Std Dev	0.009	0.009	1.236	0.823	1.121

Table 1: Channel switching latency (ms) of the Spider driver.

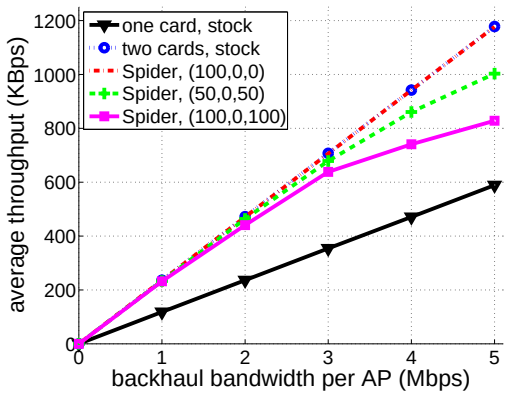

Figure 10: Throughput micro-benchmark. Spiderś throughput when dedicated to two APs on a single channel is equivalent to two cards running stock drivers.

We ran two micro-benchmarks designed to measure (1) the latency overhead incurred when switching channels, and (2) the ability of our driver to aggregate bandwidth across connections through multiple APs.

Table 1 shows the mean latency and the standard deviation of a channel switch operation. The channel switching latency is the time required to send a PSM frame to each associated AP on the old channel, perform a hardware reset to apply the channel change, and then send a PSM poll frame to each associated AP on the new channel. The latency is typically in the range of 5-6ms, increasing proportional to the number of APs, because a separate PSM frame must be sent to each AP. The largest contributor to the latency is the hardware reset step, which can vary depending on the model of the card. The latency is within a few *ms* of that achieved by other multi-AP drivers [11, 18].

Fig. 10 shows the ability of the driver to utilize the bandwidth offered by multiple APs. We measured mean aggregate throughput achieved while downloading large files over HTTP for a number of configurations: a host with a single card running stock MadWiFi, for comparison; a host with two physical cards running stock drivers; Spider connected to two APs on the same channel; and Spider associated with one AP on channel 1 and one on channel 11, with a schedule of 50ms on each channel; and the previous configuration while spending 100ms on each channel. The APs and servers were connected via LANs in our lab, and a traffic shaper was used to adjust

(Config) Parameters	Throughput	Connectivity
(1) Channel 1, Multi-AP	121.5 KB/s	35.5%
(2) Channel 1, Single-AP	28.0 KB/s	22.3%
(3) Multi-channel, Multi-AP	28.8 KB/s	44.6%
(4) Multi-channel, Single-AP	77.9 KB/s	40.2%
(2) Channel 6, single-AP*	90.7 KB/s	36.4%
MadWiFi driver *	35.9 KB/s	18.0%

Table 2: Avg. throughput and connectivity for Spider configurations. Staying on a single channel and leveraging multiple AP connections provides best avg. throughput. The multi-channel, multi-AP approach provides the best connectivity. (Multi-channel scenarios use a static schedule of 200 ms on ch. 1, 6, and 11. * denotes experiments performed in Cambridge, where channel 6 was the best.)

the backhaul bandwidth available through each AP.

The host with two physical interfaces and the host with Spider running on a single channel (connected to two APs) both achieved an aggregate throughput equal to twice that achieved by the host with a single card and stock driver—this is expected, as Spider incurs no channel-switching overheads in this configuration and does not run the risk of causing TCP timeouts when using one channel. The results for the multi-channel Spider configurations show the trade-off between exploiting new connectivity opportunities and extracting throughput from connected APs. When high-bandwidth links are available, a schedule which switches more rapidly between channels is able to achieve greater throughput by reducing the risk of TCP timeouts.

4.3 Connectivity and Throughput

We analyze throughput and connectivity of Spider using four key metrics. (1) *Average throughput*: the amount of data transferred to a sink per unit time during an experiment. (2) *Average connectivity*: the percentage of time that a non-zero amount of data was transferred to a sink. The average throughput and connectivity are bounds on open Wi-Fi performance using multi-AP solutions. (3) *Disruption length*: the contiguous period of time when there is no connectivity. This distribution indicates whether interactive applications such as VoIP or web search can be supported. (4) *Instantaneous bandwidth*: the amount of data per second transferred by a Spider node when there is connectivity. This metric indicates whether multi-AP solutions can support applications that require bursts of high throughput connectivity.

We present the average throughput and average connectivity for a Spider node in its four configurations in Table 2. These experiments were performed using a passenger car in our downtown area. A static schedule of $D = 600ms$ and $f_1 = f_6 = f_{11} = 1/3$ was used in multi-channel scenarios.

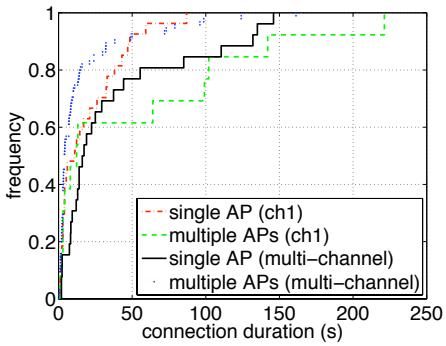

Figure 11: CDF of the Internet connectivity duration for Spider configurations.

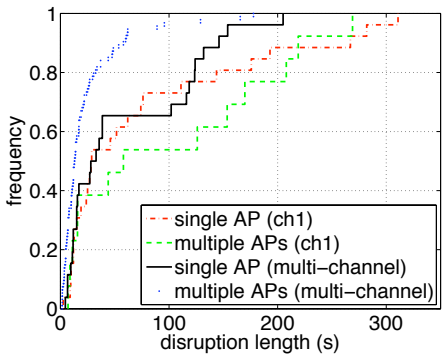

Figure 12: CDF of disruptions for Spider configurations.

Two conclusions can be drawn from the results. First, the single-channel multi-AP configuration performs best in terms of throughput. It has an average throughput of more than 4 times that of the single-AP counterpart. The use of multiple channels incurs an additional overhead of switching and associating on orthogonal channels, strangling throughput. This is in agreement with the observation made from another analytical framework where we demonstrated that at vehicular speeds, aggregating bandwidth from one channel leads to optimal performance. Second, the multi-channel multi-AP solution has the best performance in terms of connectivity. Although the average throughput is lower, multiple channels host a larger pool of APs for Spider to choose from.

Figs. 11 and 12 are the CDFs of the disruption and connection lengths for different configurations of Spider respectively. The results demonstrate several trade-offs. The longest periods of Internet connectivity are obtained by staying on one channel and maintaining concurrent connections to several APs. However, that strategy also experiences the longest disruptions due to areas where there is no Wi-Fi coverage on the chosen channel. In contrast, the multiple-channel multi-AP solution experiences the shortest connections due to disruptions caused joins to APs on separate channels. However, such an approach has the shortest disruptions due to larger set of

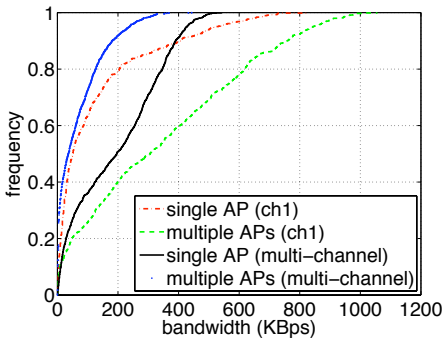

Figure 13: CDF of KB/s transferred (during connectivity) for Spider configurations. Single-channel configs. provide the best instantaneous throughput. Multi-AP, multi-channel reduce throughput due to the overhead of joining.

parameters	**Failed** `dhcp`
chan 1, linklayer: 100ms, dhcp: **600ms**, 7 interfaces	23.0% ±6.4%
channel 1, linklayer: 100ms, dhcp: **400ms**, 7 interfaces	27.1% ±5.4%
chan 1, linklayer: 100ms, dhcp: **200ms**, 7 interfaces	28.2% ±4.0%
3 Chans, static 1/3 schedule, linklayer: 100ms, dhcp: 200ms, 7 interfaces	23.6% ±10.7%
Chan 1, **default timer**, 7 interfaces	13.5% ±6.3%
3 Chans, static 1/3 schedule, **default timer**, 7 interfaces	21.8 % ±6.9%

Table 3: `dhcp` failure probabilities for different timeout configurations for Spider. Primary differences are bolded.

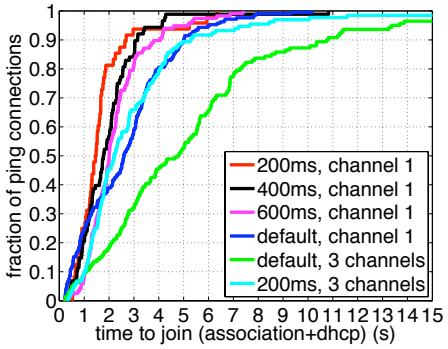

Figure 14: The rate of successful joins as a function of `dhcp` timeout. The cost of switching among channels overshadows the benefit of quickly establishing connections when timeouts are reduced.

possible APs that the node can transfer data with. The single-AP configurations provide trade-offs between the two extremes. We compare these results with usability needs of wireless users in subsection 4.7.

Fig. 13 shows the instantaneous bandwidth that Spider provides when actively transferring data. The single-channel, multi-AP configuration performs best in terms of per-connection throughput. The 60^{th} percentile is around 300 KBps and the 90^{th} percentile is around 1000 KBps — comparable to the throughput provided by FatVAP in a static environment (see Figure 13 in [11]). Spider's multi-channel, multi-AP solution performs poorly in terms of instantaneous bandwidth due to the overhead of association and `dhcp` on separate channels, clearly illustrating (as in our analytical framework) the importance of staying on a single channel if high throughput is the design goal.

4.4 Effect of AP Density

We evaluated the effect of AP density on the performance of Spider using the same set of experiments listed in Table 2. Here, we compare the case in which Spider is allowed to associate with one AP with the case where it maintains connection with multiple APs. During our experiments, Spider associated with a maximum of three APs 5% of the time, 2 APs 10% of the time, and is associated with one AP around 85% of the time. It is notable that, even with such a meager open Wi-Fi density in our town, multi-AP Spider has an average throughput that is four times that of a single AP case.

For external validation, we ran a similar set of experiments in the city of Cambridge, an environment with a different mobility pattern and AP density from ours. The last two entries in Table 2 are results for those experiments. Interestingly, on Channel 6, Spider has an average throughput that is 800% more than the throughput reported by Cabernet for the same city (a throughput of 10.75 KBps [8])[3]. Additionally, when comparing the results with the stock MadWiFi driver, we find that Spider provides $2.5x$ improvement in throughput and $2x$ improvement in connectivity.

4.5 Effect of Join Timeouts

As discussed in section 2, one of the primary challenges in designing multi-AP solutions for mobile Wi-Fi access is the overhead associated with `dhcp` and association. A way to minimize this overhead is to reduce the timeouts associated with `dhcp` and link layer retries. Table 3 and Fig. 14 show the effect of reducing these timeouts on Spider. Table 3 presents the increase in failure rate of `dhcp` requests with reduced timeouts while maintaining concurrent connections on multiple channels. Compared to the default timers, reducing timeouts can lead to a two-

[3]It is of course impossible to set up the exact conditions in which Cabernet was tested: 802.11G is now widely available and it is not possible to determine if more or less open APs are available.

Parameters	Throughput	Connectivity
3-channel (equal schedule)	28.8 KB/s	44.7%
2-channel (equal schedule)	25.1 KB/s	35.8%
Single-channel	121.5 KB/s	35.5%

Table 4: Average throughput and average connectivity seen when applying different static schedules for multi-channel configuration for Spider.

fold increase in `dhcp` failure rates. Similarly, switching among multiple channels while trying to associate with multiple APs leads to high probability of failure (as high as 30–35%).

Although the number of failed `dhcp` attempts increases with timeout reduction, in Fig. 14 we find that the median time to successfully obtain a lease improves—similar to the observation made in Cabernet [8]. However, the absolute median time to join is still 2–3 seconds, which increases by 2x when using multiple channels. Such long join times imply that is best to stay on one channel to maximally aggregate throughput.

4.6 Effect of Number of Channels

To understand the effect of number of channels on throughput, connectivity, and join overhead, we present Fig. 15 and Table 4. We tested three scenarios (1) three channels with a equal schedule (200 ms) and (2) two channels with an equal schedule (200 ms each), and (3) a single channel. Fig. 15 shows that the single channel mode with reduced timeouts performs best in terms of join time–however, the reduced timeouts lead to a large number of `dhcp` failures. Moreover, the three channel schedule performs worse than the dual channel scheduling. Hence, switching between channels during association is a primary source of overhead in multi-AP solutions. Table 4 presents throughput and connectivity results for the different configurations—as expected throughput is maximized when Spider uses a single channel and connectivity is maximized when it uses an equal schedule on three channels.

4.7 Matching Usability Needs

An open question that is widely asked is *whether open Wi-Fi access can cater to connectivity needs of mobile users?* To answer this question, we performed a study using data from a permanent Wi-Fi mesh we deployed in our downtown. The mesh consists of 25 nodes and covers an area of about 0.50 km^2. We collected performance data on all TCP flows from 161 wireless users for an entire day. Although, all users might not be mobile, the data provides us with a plausible baseline. Overall, there were 128,587 completed TCP connections in the collected data and a total number of 13,645,161 packets (1.7 GB) were sent by the users. Of these, 86,838 connections were made to the `http` port (68% of the connections).

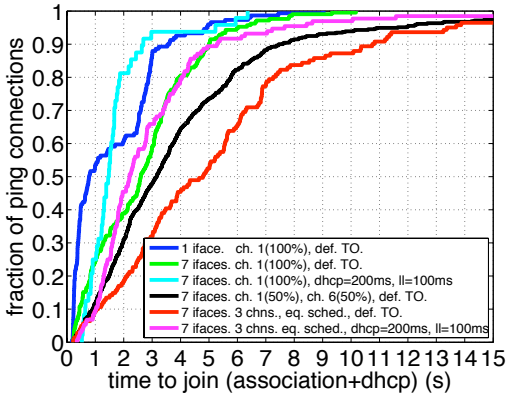

Figure 15: Delay in obtaining a dhcp lease and link layer association for different scheduling policies in Spider. The figure also considers reduced timeouts.

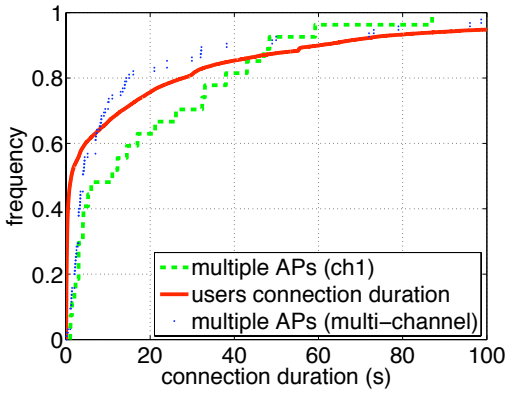

Figure 16: Comparison of connection lengths for wireless users and Spider.

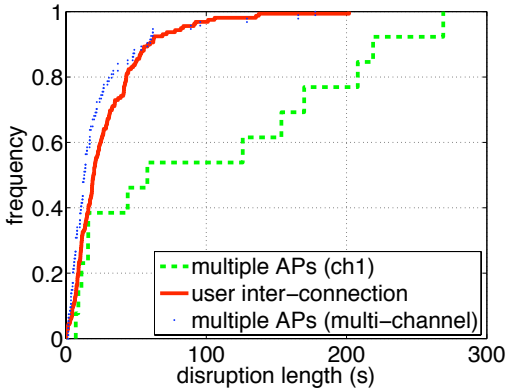

Figure 17: Comparison of disruption lengths for wireless users and Spider.

We compare the traffic needs of wireless users with those provided by Spider based on two key metrics: (1) distribution of the duration of TCP connections, and (2) distribution of inter-connection time.

Fig. 16 compares the TCP flow lengths gathered from

actual users using our mesh network and Spider in its multi-channel and single-channel modes. The figure shows that Spider can support all the TCP flows that users need. Additionally, in Fig. 17 we compare the time between two connections for the mesh users and disruption time for Spider. When Spider uses multiple channels and multiple APs, it experiences disruptions comparable to what real users can sustain.

These results present Spider as a plausible complement to cellular data services. However, more data on mobile user's connectivity needs and network usage pattern is required in order to find out the degree to which Spider can *align* itself with the needs of each individual user. Conducting this study forms part of our future work with Spider.

4.8 Limitations and Future Work

The primary limitation of our work is largely with the design of Spider. First, the system is not adaptive. Specifically, it does not change its scheduling policy dynamically based on speed and AP density. An augmented design would encompass both *mobile* and *nomadic* scenarios by alternating between staying on one channel at high speeds and managing multiple channels when moving slowly. Although we have evaluated the effect of such schedules in separate experiments, as future work, we are extending Spider to dynamically adapt schedules based on these parameters. To facilitate such integration, Spider's AP selection has to incorporate a suite of other criteria such as end-to-end bandwidth estimates in addition to the past successful joins. Current design only considers throughput maximization as an objective. We are planning to extend the design to minimize connectivity outages by dynamically exploiting connections on multiple channels.

5. RELATED WORK

Spider builds on previous work on Wi-Fi access from mobile nodes, fast Wi-Fi and cellular hand-offs, and using multiple APs for throughput aggregation. Here, we compare and contrast Spider with the most relevant literature.

Wi-Fi Access from Moving Vehicles: Several challenging problems such as lossy wireless mediums [4, 8, 9, 13], tuning TCP performance for mobility [1], and AP selection [17] are well studied. The feasibility of using cached history to reduce association and dhcp overheads [6] has been demonstrated. Directional antennas have also been used to improve throughput [15]. However, this body of work concentrates on using a stock Wi-Fi model—association and data transfer with one AP at a time. However, as we have demonstrated, using a aggregation of APs can provide high aggregate throughput and better connectivity in a mobile environment.

Performance through diversity: Using technological and spatial diversity to improve Wi-Fi connectivity has also been studied in the past. This class of related work can be broadly classified by either infrastructure-end or client-end modifications. Infrastructure-end modifications includes coordination or selection amongst multiple open APs [3, 12–14, 14, 22]. In contrast to these approaches, Spider is a purely client-side solution that aims at improving a mobile user's performance in an *organic* Wi-Fi setting. *Client-side* diversity-based solutions rely on aggregating bandwidth across multiple APs [5, 11, 18]. However, these solution are tuned to work efficiently only in a *static*, stationary wireless environment.

An orthogonal approach to aggregating bandwidth in mobile nodes is using additional hardware—the assumption is that each client has more than one Wi-Fi card and data can be striped across concurrent connections to APs. PERM [21] is a multi-homed solution that aggregates throughput across multiple residential ISPs, profiles on-going connections, and assigns flows to interfaces to minimize delay. MAR [20] exploits heterogeneity of existing wide-area wireless networks by using a router architecture that aggregates independent cellular links into one fat reliable virtual data transfer pipe. Horde [19] is a middleware solution on mobile nodes that performs network striping over diverse cellular links tuned to application needs. Most of these data striping approaches can be built into Spider to enhance mobile user performance.

Soft hand-off and AP selection: Spider also builds on related work on fast cellular hand-offs and AP selection in mobile Wi-Fi networks. Fast hand-off is used to mitigate the adverse effects of disruptions in cellular networks [7]. While fast soft hand-off is plausible in a cellular network where the cell towers are under the control of one central authority, it is not feasible in Wi-Fi networks laid down by third-party users. The only practical soft hand-off solution using client side modifications is Spider that virtualizes the Wi-Fi card and maintains concurrent AP connections. Access point selection has also been an active area of research in Wi-Fi mobile networks. Several techniques including RSSI [10] and history-based techniques [16] have been proposed. However, multi-AP selection, solved by Spider is a harder problem (as we demonstrate in the Appendix) since it involves selection of a *set* of APs.

6. CONCLUSION

We presented an in-depth analysis of the performance of attempting concurrent AP connections from highly mobile clients. Through an analytical model, optimization framework, and numerous indoor and outdoor experiments, we isolated several factors that affect the poor performance of multi-channel networking in contrast to

single-channel, multi-AP networking and conventional approaches. We found that connection duration, AP response times, channel scheduling, and attained and offered bandwidth all affect performance. Moreover, link association, `dhcp` joins, and TCP are all negatively affected by fractional channel scheduling.

Building off these results, we designed and implemented a novel multi-AP driver. Spider uses utility-based multi-AP selection, channel-based scheduling, and opportunistic scanning to maximize throughput while mitigating the overhead of association and `dhcp`. While Spider manages multiple channels if desired, we show empirically that using multiple APs on a single channel achieves higher throughput than scheduling on multiple channels, as predicted. Our evaluation of Spider on a vehicular testbed shows that it provides a significant improvement in throughput, making it an effective supplement to cellular data services for highly mobile nodes.

7. REFERENCES

[1] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz. A comparison of mechanisms for improving TCP performance over wireless links. In *ACM SIGCOMM*, pages 256–269, 1996.

[2] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting Mobile 3G Using WiFi: Measurement, Design, and Implementation. In *Proc. ACM Mobisys*, 2010.

[3] A. Balasubramanian, R. Mahajan, A. Venkataramani, B. Levine, and J. Zahorjan. Interactive wifi connectivity for moving vehicles. *Proc. ACM SIGCOMM*, 2008.

[4] V. Bychkovsky, B. Hull, A. K. Miu, H. Balakrishnan, and S. Madden. A Measurement Study of Vehicular Internet Access Using in situ 802.11 Networks. In *Proc. ACM MOBICOM*, pages 50–61, Sept 2006.

[5] R. Chandra, P. Bahl, and P. Bahl. MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card. In *Proc. IEEE INFOCOM*, March 2004.

[6] P. Deshpande, A. Kashyap, C. Sung, and S. Das. Predictive Methods for Improved Vehicular WiFi Access. In *Proc. ACM Mobisys*, 2009.

[7] N. Ekiz, T. Salih, S. Kucukoner, and K. Fidanboylu. An overview of handoff techniques in cellular networks. In *Intl. Journal of Information Technology*, 2006.

[8] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: Vehicular Content Delivery Using WiFi. In *Proc. ACM MobiCom*, Sept. 2008.

[9] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *ACM SenSys*, pages 125–138, October 2006.

[10] G. Judd and P. Steenkiste. Fixing 802.11 access point selection. In *ACM CCR*, 2002.

[11] S. Kandula, K. C.-J. Lin, T. Badirkhanli, and D. Katabi. FatVAP: aggregating AP backhaul capacity to maximize throughput. In *Proc NSDI*, pages 89–104, 2008.

[12] V. Leung and A. Au. A wireless local area network employing distributed radio bridges. *Wirel. Netw.*, 2(2):97–107, 1996.

[13] A. Miu, A. Miu, H. Balakrishnan, C. Emre, K. Mit, and C. Science. Improving Loss Resilience with Multi-Radio Diversity in Wireless Networks. *Proc. ACM Mobicom*, pages 16–30, 2005.

[14] A. Miu, G. Tan, H. Balakrishnan, and J. Apostolopoulos. Divert: fine-grained path selection for wireless LANs. In *Proc. ACM MobiSys*, pages 203–216, 2004.

[15] V. Navda, P. Subramanian, K. Dhanasekaran, A. Timm-giel, and S. Das. Mobisteer: Using steerable beam directional antenna for vehicular network access. In *Proc. ACM Mobisys*, 2007.

[16] A. Nicholson, Y. Chawathe, M. Chen, B. Noble, and D. Wetherall. Improved access point selection. In *Proc. MobiSys*, pages 233–245, 2006.

[17] A. Nicholson and B. Noble. BreadCrumbs: forecasting mobile connectivity. In *Proc. ACM Mobicom*, 2008.

[18] A. Nicholson, S. Wolchok, and B. Noble. Juggler: Virtual Networks for Fun and Profit. In *IEEE Trans. Mobile Computing*, 2009.

[19] A. Qureshi and J. Guttag. Horde: separating network striping policy from mechanism. In *Proc. ACM MobiSys*, pages 121–134, 2005.

[20] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. Mar: a commuter router infrastructure for the mobile internet. In *Proc. ACM MobiSys*, pages 217–230, 2004.

[21] N. Thompson, G. He, and H. Luo. Flow Scheduling for End-host Multihoming. In *Proc. IEEE INFOCOM*, 2006.

[22] A. Viterbi, A. Viterbi, K. Gilhousen, and E. Zehavi. Soft Handoff Extends CDMA Cell Coverage and Increase Reverse Link Capacity. In *Proc. Intl. Zurich Seminar on Digital Communications*, pages 541–551, 1994.

APPENDIX

A. PROOF OF NP-HARDNESS

We assume that the mobile node spends T seconds on a road segment that has n open WiFi access points. We also assume that the driver has n virtual interfaces, thus, it is possible to establish concurrent connections to the n APs. Let S_i be subset i of the power-set of n APs. We define a *value* V_i for each subset S_i. V_i is a function of the APs in the subset and quantifies connectivity or throughput. For example, if cumulative throughput is our desired metric, and the wireless bandwidth that S_i can provide is W_i then $V_i = T_i \times W_i$, where T_i is the time spent by a Spider node within the range of the APs in S_i. We also define a cost C_i associated with S_i. C_i is the sum of the time that Spider spends within range of the APs in S_i, the association time, and the switching overhead among channels and processing per channel queues. If D_i is this overhead, $C_i = T_i + \lceil T_i/T \rceil \times D_i$.

With these parameters as input, the goal of the multi-AP optimization problem is to select a set of subsets S_i, such that the sum of their values is maximized subject to the following constraints: (1) the total cost should not exceed the total time T and, (2) each T_i must be positive and less than T. Formally stated:

$$\max \sum_i T_i W_i \tag{11}$$

$$\text{such that} \sum_i (T_i + \lceil T_i/T \rceil D_i) \leq T \tag{12}$$

$$\forall i, 0 \leq T_i \leq T \tag{13}$$

The above problem is equivalent to the 0-1 knapsack problem, which is known to be NP-hard.