Towards WiFi Mobility without Fast Handover

Andrei Croitoru, Dragoș Niculescu, Costin Raiciu,
University Politehnica of Bucharest

Abstract

WiFi is emerging as the preferred connectivity solution for mobile clients because of its low power consumption and high capacity. Dense urban WiFi deployments cover entire central areas, but the one thing missing is a seamless mobile experience. Mobility in WiFi has traditionally pursued fast handover, but we argue that this is the wrong goal to begin with. Instead, we propose that mobile clients should connect to all the access points they see, and split traffic over them with the newly deployed MPTCP protocol. We let a mobile connect to multiple APs on the same channel, or on different channels, and show via detailed experiments and simulation that this solution greatly enhances capacity and reliability of TCP connections straight away for certain flavors of WiFi a/b/g. We also find there are situations where connecting to multiple APs severely decreases throughput, and propose a series of client-side changes that make this solution robust across a wide range of scenarios.

1 Introduction

Mobile traffic has been growing tremendously to the point where it places great stress on cellular network capacity, and offloading traffic to the ubiquitous WiFi deployments has long been touted as the solution. In dense urban areas, offloading between WiFi and cellular may not be needed at all: WiFi is always available because of uncoordinated deployments by many parties, and is preferable because it offers higher bandwidth and smaller energy consumption.

To improve the mobility experience, many solutions have been proposed that *coordinate* Access Points [1, 2]; however, real deployments are fragmented between multiple operators, which together cover entire central areas. This is the case, for instance, in pedestrian areas of the Bucharest city center with three different hotspot providers active. In this paper we address the problem of roaming through mixed, uncoordinated deployments of APs, without changes to the deployed infrastructure. We assume clients have access to multiple operators (perhaps due to roaming arrangements between operators), or to home-users' APs(also uncoordinated) as proposed by Fon[3]. Recent surveys [4, 5] show that in cities one can find tens of networks on most popular channels.

Traditional WiFi mobility techniques, as with all other L2 mobility mechanisms are based on the concept of fast handover: when a mobile client exits the coverage area of one Access Point (AP), it should very quickly find another AP to connect to, and quickly associate to it. There is a great wealth of research into optimizing fast handover including scanning in advance, re-using IP addresses to avoid DHCP, synchronizing APs via a backplane protocol, even the using additional cards[6] to reduce the association delay - see § 2 for more details. We think this is the wrong approach, for many reasons:

1. To start the handover mechanism, a client has to lose connectivity to the AP, or *break-before-make*
2. There is no standard way to decide which of the many APs to associate with for best performance
3. Once a decision is made, there is no way to dynamically adjust to changes in signal strength or load

We conjecture that the emerging standard of Multipath TCP (MPTCP) enables radical changes in how we use WiFi: use of multiple APs becomes natural, whether on the same channel or different ones, and the perennial handoff problem at layer 2 gets handled at layer 4 allowing for a clean, technology independent, end-to-end solution for mobility. In this paper we test the following hypothesis: *all WiFi clients should continuously connect to several access points in their vicinity for better throughput and connectivity.*

We carefully analyze the performance experienced by a mobile client locked into using a single WiFi channel and associating automatically to all the APs it sees, *without using any explicit layer 2 handover*. We run a mix of testbed experiments to test a few key usecases and simulations to analyze the generality of our testbed findings across a wide range of parameters and scenarios. We find that, surprisingly, the performance of this simple solution is very good out of the box for a wide range of scenarios and for many WiFi flavors (802.11a, b, g): a WiFi client connecting to all visible APs will get close to the maximum achievable throughput. We discuss in detail the reasons for this performance, namely the WiFi MAC behavior and its positive interaction with MPTCP. In particular, the *hidden terminal problem gets a constructive solution* with MPTCP, as subflows of a connection take turns on the medium instead of competing destructively.

We also find that performance is not always good for certain combination of standards (e.g. 802.11n and g), and for some rate control algorithms: in such cases the

client downloads too much data from APs far away, reducing total throughput. To address these issues, we design, implement and test: (a) a novel client side estimation technique that allows the client to accurately estimate the efficiency of the downlink from an AP, and b) a principled client-side algorithm that uses ECN marking to help the MPTCP congestion controller to find the most efficient APs. Finally, we implemented an adaptive channel switch procedure that allows harvesting capacity from APs on different channels.

We ran several mobility experiments in our building, comparing our proposal to using regular TCP connected to the individual APs. The results show a truly mobile experience: our client's throughput closely tracks that of a TCP client connected to the best AP at any given location. We also show that striping traffic across APs naturally and fairly harvests bandwidth in contention situations with hidden and exposed terminals.

2 Background and Related Work

Fast Handover. The 802.11 standards were originally designed for uncoordinated deployments, and are not particularly amenable for high speed handovers. The handover is performed in three phases: scanning, authentication, and association. The first phase has been empirically shown [7] to be 95% of the handover time, and has been the target for most optimizations in the literature.

One approach to reduce the scanning delay is to probe for nearby APs before the mobile loses its current link. SyncScan [8] goes off channel periodically to perform the scan, so that the mobile has permanent knowledge about all APs on all channels. DeuceScan [9] is a prescan approach that uses a spatiotemporal graph to find the next AP. Mishra et al. [10] uses neighbor graphs to temporarily capture the topology around the mobile and speed up the context transfer between APs using IAPP. When additional hardware is available, [6] delegates the task of continuous probing to a second card.

For enterprise networks there are several opportunities to optimize the handover process. One is the use of a wireless controller that has a global view of all the APs and the associated mobiles in the entire network. This architecture is supported by all major vendors, because it offers many other advantages in addition to controlled association and guided handover. Another avenue for better handover is to eliminate it altogether, with the adoption of a single channel architecture [11] where multiple coordinated APs "pose" as a single AP by sharing the MAC, an architecture currently in use by Meru, Extricom and Ubiquiti. In these architectures, the wireless controller routes the traffic to and from the appropriate AP, so that a mobile never enters the handover process.

802.11r [12] optimizes the last two components of the handover ensuring that the authentication processes and encryption keys are established before a roam takes place. Authentication occurs only once, when a client enters the mobility domain. Subsequent roams within a mobility domain use cryptographic material derived from the initial authentication, decreasing roam times and reducing load on back-end authentication servers. 802.11k [13] provides for roaming assistance from the AP: when it perceives the mobile moving away, it sends a notification and possibly a site report with other AP options to maintain connectivity. Finally, a survey of handover schemes [14] mentions several other efforts in this direction and classifies them based on the incurred overhead, necessity of changes (mobile, AP, or both), compatibility with standards, and complexity.

All these layer 2 solutions do improve handover performance, but their availability depends on widespread support in APs and clients. Performing a handover is a decision that locks the client into one AP for a certain period of time, which leads to poor performance when truly mobile: there is no good way of predicting the throughput the client will obtain from one AP in the very near future. By using a transport layer mobility solution (see also [15, 16]), we sidestep the need to upgrade all AP and client hardware for mobility and, more importantly, allow a client to utilize multiple APs at the same time.

Channel Switch [17, 18, 19, 20] has been used to allow round robin access to APs on different frequencies. The client maintains association to each AP, and uses an IP address in that subnet. All these schemes rely on a client's powersave mode to have an AP buffer packets while the client is exchanging data with other APs on other channels. We also used this method in our implementation (see section 7). Spider [21] also investigates multiple associations on the same and on different channels, and concludes that for high speed, sticking to one channel yields better results.

AP selection is the problem of choosing the right AP based on signal strength, available wireless bandwidth, end-to-end bandwidth, RTT[22], current load. [23] estimates the wireless bandwidth that the client would receive by using timing of beacon frames. [24] shows that WLAN multi-homing is desirable from several angles: pricing, efficiency, fairness. [25] also uses a game theoretical approach to explore AP association strategies depending on delay probing. [26] proposes collaborative sharing of a fraction of an APs bandwidth, which enables better centralized load balancing. Divert [1] is a heuristic that selects the best AP for downlink and uplink, exploiting physical path diversity. Similarly, MRD [27] also exploits path diversity, but only for uplinks.

MPTCP is a recently standardized extension of TCP [28] that has been implemented by Apple in the IOS 7 mobile operating system; more mobile phone manufac-

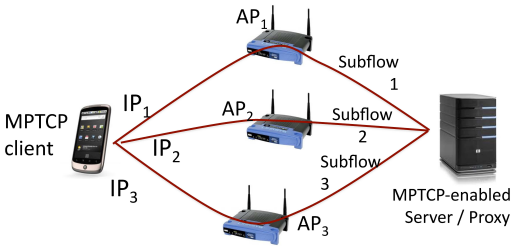


Figure 1: Instead of fast handovers, we propose that wireless clients associate to all the APs in range and use MPTCP to spread traffic across them.

turers are expected to follow suit.

The idea of associating to multiple APs for better robustness and throughput is not new [21, 29, 18]. What is missing is the ability of unmodified applications to benefit from multiple APs, as TCP uses a single interface by default. Multipath TCP enables unmodified apps to use these interfaces.

Our contribution in this paper is to study and optimize the interaction between the WiFi MAC and Multipath TCP. In contrast to previous works that focus mostly on channel switching, we examine carefully the case where the client is associated to multiple APs residing on the same channel. Our solution departs from the "one AP at a time" philosophy in existing works, allowing multiple APs to compete for the medium at packet level. Competition provides a number of features including elegantly solving hidden terminal issues, fast reaction to changes in channel conditions and throughput improvements even in certain static cases.

3 Towards an optimal solution for WiFi Mobility

Consider a wireless client that can associate to three distinct APs, as shown in Figure 1. Which one should the client pick and associate to? Prior work has shown that using signal strength is not a good indicator of future performance, so the client may actively probe or passively measure [23] all three APs briefly before deciding on picking one of them. However, this initially optimal choice may quickly become suboptimal because of multiple reasons outside the client's control: 1. the client may move; 2. other clients may use the medium, affecting this client's throughput and his choice; 3. the wireless channel to the chosen AP may have temporary short-term fluctuations, affecting its capacity (see evaluation in section 6 for an example).

The combination of these factors is impossible to predict in practice, and the best AP for any given client changes not only in mobility scenarios, but even when the client is stationary. All existing solutions that connect to a single AP are forced to be conservative, because fluctuations (flopping between APs) can affect performance; thus they all tend to stick to their choice for some time.

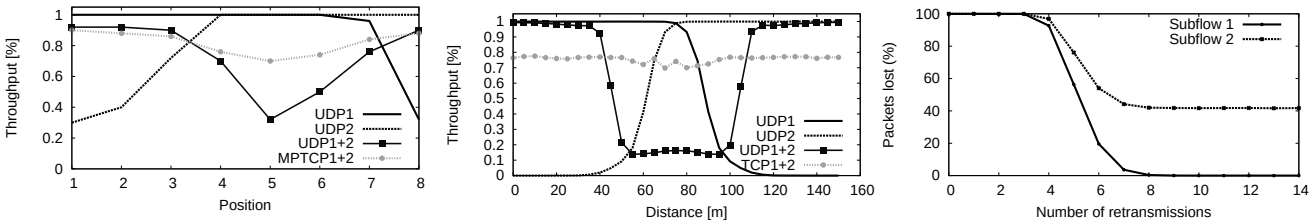
We observe that the emergence of MPTCP enables a radically different approach to WiFi mobility: instead of using only one AP at a time and doing handovers, **mobile clients should connect to all APs at any given time**. The solution is conceptually very simple, and is shown in Figure 1: we have the client associate to multiple APs, obtaining one IP address from each, and then rely on MPTCP to spread data across all the APs, with one subflow per AP. As the mobile moves, new subflows are added for new APs, while old ones expire as the mobile loses association to remote APs.

How should traffic be distributed over the different APs? As the client has a single wireless interface, it can only receive packets from one AP at a time, even if it is associated to multiple APs. Should the client spend an equal amount of time receiving data via each AP? This policy is optimal only when all APs offer equal throughput. In practice, one AP will offer the best performance, thus it is preferable for the client to transfer most data via this access point. However, all other feasible APs should be used to send probe traffic to ensure that the client can detect when conditions change and adapt quickly. While simple in principle, the key to this solution is understanding the interactions between MPTCP and the WiFi MAC. There are two high-level cases that need to be covered:

APs on the same wireless channel. There are three non-overlapping channels in 2.4GHz and more in 5GHz, but newer standards including 802.11n and 802.11ac allow bonding 2-8 of these 20Mhz channels to increase client capacity; the result is a smaller number of usable non-overlapping bonded channels (maximum three with 802.11ac, depending on the country).

If we disregard WiFi interference between APs, the theoretically optimal mobility solution is to always connect to every visible AP, and let MPTCP handle load balancing at the transport layer: if an AP has poor signal strength, its loss rate will be higher (because of lower bandwidth and similar RTTs) and the MPTCP congestion controller will simply migrate most of the traffic to the APs with better connectivity to the client. This way, handover delays are eliminated and the mobile enjoys continuous connectivity. But interference can be a major issue, and will be explored in depth in the next section.

APs on different wireless channels. In this case the mobile client must dynamically switch channels while associated to multiple APs, giving each AP the impression it is sleeping when in fact it is going on a different channel. Channel switching has already been proposed as a technique to aggregate AP backhaul capacity by a number of works including FatVAP [18] and Juggler[20]. We discuss the interactions between MPTCP and channel switching in section 7.



(a) Experimental results with 802.11a, 6Mbps: UDP flows systematically collide, while MPTCP subflows take turns on the medium.

(b) ns2 simulation of the same situation in 2a. In the middle region MPTCP exhibits higher variability because one subflow starves the other.

(c) The subflow sending packets infrequently experiences a much higher loss rate. This makes it hard for a (MP)TCP flow to escape its slowstart.

Figure 2: Hidden terminal (HT) experiments: using Multipath TCP results in very good throughput because one subflow monopolizes the air, while the other is starved.

4 Single-channel mobility

We implemented a prototype client that is locked on a single channel and continuously listens for beacons of known APs; when a new AP is found, the client creates a new virtual wireless interface[1] and associates to the AP, opening a new MPTCP subflow via the new AP. We ran this code on our 802.11a/b/g/n testbed without external interference, as well as in simulation to understand the interactions that can arise due to interference between different APs, and the extent to which this solution approximates the optimal one.

4.1 Hidden terminal experiments

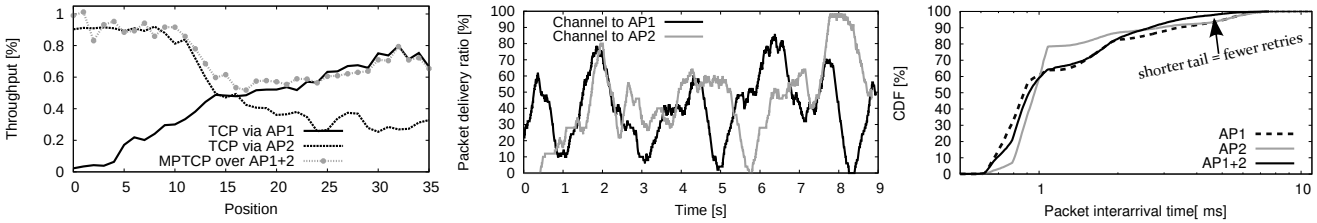
The first case we test is a pathological one: consider two APs that are outside CS range and the MPTCP client connects to both. Lack of CS means the CSMA mechanism does not function and the frames coming from the two APs will collide at the client. In fact, each AP is a hidden terminal for the other.

To run this experiment, we reduced the power of our two APs until they went out of CS, with the client still able to achieve full throughput to at least one AP at all test locations. Then, we place the client close to one AP and move it towards the other AP in discrete steps and measure the throughput for UDP and TCP via either AP (the status quo) as well as MPTCP. As shown in figure 2a, the graph exhibits three distinct areas. In the two areas close to either AP, neither UDP nor TCP throughput is affected: here the capture effect[30] of WiFi predominates, as packets from the closest AP are much stronger, and the effect of a collision is very small—the client will just decode the stronger packet as if no collision took place, and the subflow via the furthest AP will reduce its rate to almost zero because of repeated packet losses.

[1] Virtual interfaces are a standard Linux kernel feature: each interface has individual network settings (IP, netmask), MAC settings(association, retries), but share the PHY settings(channel, CS).

The area in the middle is more interesting. As we expected, the combined UDP throughput of two simultaneous iperf sessions is greatly diminished by the hidden terminal situation. However, by running two simultaneous MPTCP subflows, the combined throughput is surprisingly good. Repeated runs showed this result is robust, and we also confirmed this via ns2 simulation (Figure 2b). MPTCP connection statistics show that the high-level reason for the high throughput is that traffic is flowing entirely over one of the two subflows, while the other one is starved, experiencing repeated timeouts. This would suggest that the starved subflow is experiencing much higher loss rates, which would explain why it never gets off the ground properly.

To understand the reason of this behavior, we used simulation to measure the loss probability of the two subflows when contending for the medium. When subflow 1 is sending at full rate, subflow 2 sends a single packet which collides with a packet of subflow 1. The WiFi MACs will then backoff repeatedly until the max retransmission count has been reached, or until one or both packets are delivered. We run the simulation for a long time to experience many such episodes, and show the percentage of episodes each subflow experiences a loss in Fig.2c as a function of the retry count. When few retransmissions are allowed, both subflows lose a packet each when a collision happens, but the effect of the loss is dramatic for the second subflow pushing it to another timeout. As we allow more retransmissions, the loss probability is reduced dramatically: the second subflow loses around 40% of its packets if 6 or more retransmissions are allowed. The reason for the flattening of the line at 40% is the fact that the first sender always has packets to send, and when subflow 1 wins the contention for the first packet, its second packet will start fresh and again collide with the retransmissions from the second subflow, further reducing its delivery ratio. This also explains why subflow 1 experiences no losses after six retransmissions: either it wins the contention immediately, or it succeeds just after the second subflow delivers its



(a) Throughput of a client moving between AP_1 and AP_2: the MAC favors the sender with the better channel. Max throughput is 22Mbps.

(b) A fixed node has channels with a raw PDR≈50% to each AP. The two channels are weakly correlated.

(c) Packet inter-arrival rate exhibits a longer tail when a single AP is used. When both APs are sending, the tail is much shorter.

Figure 3: Carrier sense experiments: the client using MPTCP gets the throughput of the best TCP connection when close to either AP, and better throughput when in-between.

packet. In effect, we are witnessing a capture effect at the MPTCP level triggered by the interaction with the WiFi MAC. This behavior is ideal for the MPTCP client.

4.2 Carrier-sense experiments

The most common case is when a client connects to two APs on the same channel that are within carrier sense range of each other, so that the WiFi MAC will prevent both APs sending simultaneously. The mobile associates to both APs and again we move the client from one AP to the other in discrete steps. The performance of our MPTCP client in this case strongly depends on the rate control algorithm employed by the AP, and we explore a number of these to understand their effects.

First, we have our Linux APs use 802.11a and run the default Minstrel rate selection algorithm. The results are given in Fig. 3a, and they show that the throughput of the MPTCP client connected to both APs is as high as the maximum offered by any of the two APs. The reasons for this behavior are not obvious.

CASE I: In-between APs the client obtains slightly more throughput (10%) by using both APs than if we were using either AP in isolation. The fundamental reason lies at the physical layer: due to fading effects, individual channel quality varies quickly over time, despite both channels having a roughly equal longer-term capacity. The wide variability and burstiness of losses and successes in frame delivery is well documented in literature [31, 1]. To test this hypothesis, we simultaneously sent a low rate broadcast stream from each AP and measured their delivery ratios at the client. As broadcast packets are never retried, their delivery ratio captures the channel quality accurately; the low packet rate is used to ensure the two APs don't fight each other over the airtime, while still allowing us to probe both channels concurrently. The instantaneous packet delivery ratios computed with a moving window are shown in Fig. 3b, confirming that the two channels are largely independent.

The 802.11 MAC naturally exploits physical channel diversity: the sender that sees a better channel will decrease its contention window, and will be advantaged even more over the sender with a weaker channel. This behavior is experimentally verified by previous work [32] with several clients and bidirectional traffic to/from the APs. For our client downloading from two APs, when one has a slightly worse channel, it will lose a frame and double its contention window before retrying, leaving the other AP to better utilize the channel.

To validate our hypothesis, we analyzed inter-arrival times between packets for the client using either AP or both at the same time, and plotted the CDF in Figure 3c. The data shows that most packets arrive 1ms apart, and that AP1 prefers a higher PHY rate (24Mbps) while AP2 prefers a lower PHY rate (18Mbps) when used alone. Using both APs leads to inter-arrival times in between the two individual curves for most packets. The crucial difference is in the tail of the distribution, where using both APs results in fewer retries per packet. When one AP experiences repeated timeouts, the other AP will send a packet, thus reducing the tail.

The optimal AP changes at timescales of seconds, and a realistic way of harvesting this capacity is by connecting to multiple APs. Further experiments with 802.11n and simulations have shown this behavior is robust: even when the APs offer similar long-term throughput, a client connected to multiple APs will manage to harvest 10-20% more throughput, consistent with results in [1].

CASE II: One AP dominates. Consider now the cases when the client is closer to one AP; in such cases the most efficient use of airtime is to use *only* the AP that's closest to the client. In this case, the throughput of a client connected to all APs strongly depends on the rate selection algorithms used.

In the experiment in Fig.3a *minstrel* favors higher rates even at low signal strengths (with lower frame delivery rate), leading to more retries per packet for the far away AP. Each retry imposes a longer backoff time on the transmitter, allowing the AP with better signal strength to win the contention more often and

thus to send more packets; this explains the near-optimal throughput enjoyed by the MPTCP client near either AP. This behavior is strictly enforced by the L2 conditions, and we verified that the choice of TCP congestion control has no effect on the distribution of packets over the two paths; the same results were obtained with UDP.

We also verified in the simulator that when two senders use the same rate, the MAC preference for the better sender holds regardless of the maximum number of retransmissions allowed (0 - 10). What happens when the AP farthest from the client sends using lower rates, thus reducing its frame loss rate? Simulations showed that the effect on total throughput can be drastic: the client connecting to both APs can receive less then half of the throughput it would get via the best AP. This is because lower rates give the farthest AP and the closest one similar loss rates and thus chances of winning the contention for the medium. However, packets sent at lower bitrate occupy more airtime, thus decreasing the throughput experienced by the client [33].

Is this case likely to appear in practice? We ran the same experiment on APs and clients running 802.11n in the 5GHz frequency band. When the client is close to one of the APs, the results differed from 802.11a/g: the throughput obtained with MPTCP was only approximately half the throughput obtainable via the closest AP.

Monitoring the PHY bitrates used by the transmitters shows that *minstrel_ht* (the rate control algorithm Linux uses for 802.11n) differs from *minstrel* significantly: instead of using aggressive bitrates and many retransmissions, *minstrel_ht* chooses the rates to ensure maximum delivery probability of packets. The block ACK feature of 802.11n is likely the main factor in this decision, as the loss notification now arrives at the sender after a whole block of frames has been transmitted (as much as 20): the sender can't afford to aggressively use high bitrates because feedback is scarce.

This issue is not limited to 802.11n: any rate control algorithm that offers packet-level fairness between multiple senders in CS range greatly harms the combined throughput achievable by MPTCP with multiple APs.

In summary, a client that associates to multiple APs and spreads traffic over them with MPTCP will receive close-to-optimal performance in a hidden-terminal situation, but in CS performance is strongly dependent on the rate adaptation algorithms employed by the APs, and these are outside the client's control.

5 Making MPTCP and the WiFi MAC play nicely together

There are two reasons for the suboptimal interaction between the 802.11 MAC and MPTCP: for one, the loss rate perceived by MPTCP on each subflow does not reflect the efficiency of the AP routing that subflow. In cases where packet-level fairness exists between APs, MPTCP sees the same loss rate on all subflows, and is unable to properly balance the load. Secondly, when subflows have shared bottlenecks, MPTCP assumes that sending traffic via the subflows will not affect the bottleneck capacity. This is not the case in single-channel WiFi setups, where sending traffic via a faraway AP will decrease the total throughput.

To fix these problems, it is simplest to stop using APs that decrease total throughput, but this comes at the expense of poorer performance when mobile. A more sensible option is **on-off**: receive data from a single AP at any point in time while all the others are shut-off, and periodically cycle through all available APs. **on-off** has already been proposed in the context of channel switching [18, 20, 34] and we use it in Section 7 to cope with APs on different channels. In our context, **on-off** can be implemented either by using the MP_BACKUP mechanism in MPTCP [28] which allows prioritizing subflows, or by relying on WiFi power save mode. It seems natural to extend the **on-off** solution for single channel APs as in [34], since there is no real cost of "switching" between APs on the same channel, beyond a couple of control messages: there is no wasted airtime. However, there are also a few fundamental drawbacks:

- Switching between APs increases packet delay and jitter, which affects interactive traffic. For instance, with a 200ms duty cycle, many packets experience RTTs that are 200ms longer that the path RTT.
- Gains from channel independence are lost.
- When multiple clients make independent decisions to switch between APs, they may end-up switching at the same time to the same AP, wasting capacity available elsewhere. Simulation results in §7 that show around 35% of the available capacity can be wasted in such cases.

Clients can monitor local PHY/MAC conditions accurately, but have a limited view of end-to-end downlink capacity available via an AP, because end-to-end loss rate and RTT are only available at the TCP sender. The sender, on the other hand, uses inaccurate metrics that are influenced by the WiFi behavior. For these reasons, our solution allows different APs to simultaneously send to the same client, while allowing the MPTCP congestion controller to direct traffic to the most efficient APs. In particular, our MPTCP client uses local WiFi information to find out which APs are preferable, and relays this information to the sender as additional loss notification. One non work-conserving way to relay this information is to have the client drop a percentage of packets. Instead, we use explicit congestion notification(ECN) to tell the server to avoid, if possible, the bad APs.

Our solution has two distinct parts discussed next: a novel client-side passive measurement technique that allows the client to accurately estimate the efficiency of each AP, and an algorithm that decides the amount of ECN marks that a client can set on a subflow to enable efficient airtime usage.

5.1 Measuring AP Efficiency

When deciding which AP it should prefer, the client needs to estimate the time T_i it takes on average for AP i to successfully send a packet, assuming the AP is alone in accessing the wireless medium. This metric purposely ignores the effect other WiFi clients may have on the transmission time by contending for the medium, or other clients that are serviced by the same AP. By comparing the resulting packet times, the client can decide which AP is preferable to use, and can signal the sender to steer traffic away from the other APs via ECN.

In contrast to previous work, we only care about the **hypothetical wireless bandwidth** from each AP to the client, as some of the interference from other APs is created by the client itself, so actual wireless bandwidth is not a good estimator.

We model the packet delivery time T (if the client were alone), when using the bitrate MCS at the AP and a PHY loss rate p with R retransmissions per packet, ignoring packets undelivered after R retries:

$$T = \sum_{i=0}^{R} [(\frac{MSS}{MCS} + K) \cdot (i+1) + C \cdot \sum_{j=0}^{i} \cdot 2^j] \cdot p^i \cdot (1-p) \quad (1)$$

In the model above, the first term measures the packet transmission time including the airtime used and K accounts for different WiFi constants such as SIFS, DIFS and the time needed to send the ACK at the lowest MCS (1Mbps). The term $C \cdot ...$ measures the time spent due to the contention interval, and models its increase on successive frame losses. Finally, the whole term is moderated by the probability that i retransmissions are needed to successfully send the packet.

The client knows the MCS used by the AP, however estimating the PHY loss rate is more difficult because it can only observe successful transmissions; for each successful transmission there may be an unknown number of retransmissions, which conceal the physical loss rate. Thus the obvious formula $delivery\_prob = \frac{N_{received}}{N_{total}}$ cannot be used at the client, as N_{total} is unknown.

We avoid this problem by leveraging the "retry" bit present in the MAC header of every frame, signaling whether the frame is a retransmission. The client counts N_0, the number of frames received with the retry bit set to 0. All the other frames reaching the client will have the retry bit set to 1, and are counted as N_1. We recast the previous formula to measure the delivery probability only for frames that are delivered on the first attempt:

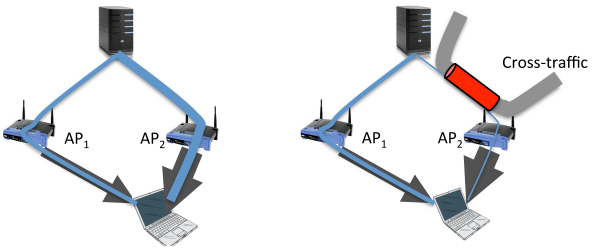


Figure 4: Scenario 1 (left): a client using AP_1 and AP_2 prefers AP_2 because of its more efficient use of airtime. Scenario 2 (right): moving all traffic to AP_2 with its better radio conditions is not the optimal strategy end-to-end

$$delivery\_prob = \frac{N_0}{N_0 + N_1 + N_{lost}} \quad (2)$$

The term N_{lost} captures packets that were not delivered by the AP despite successive attempts as shown by the sequence number present in the MAC header.

Implementation. To accurately estimate the delivery probability for all APs on a channel, the client maintains a FIFO queue of packets seen in a chosen interval, recording for each packet the time of arrival, its sequence number and its retry bit (10B in total). When new packets are received, or when timers expire, the packets outside the interval are purged, and N_0, N_1 and N_{lost} of the corresponding AP are modified accordingly. Our implementation uses an interval of 500ms, which results in an overhead per channel of around 10KB for 802.11a/g, and 100-200KB for 802.11n with four spatial streams.

5.2 Helping senders make the right choice

Consider the two scenarios depicted in Figure 4, where AP_2's packet time is shorter than AP_1's, and the two subflows going via AP_1 and AP_2 do not interfere at the last hop. MPTCP congestion control [35] requires that it does no worse than TCP on the best available path, and it efficiently uses network resources. MPTCP achieves the first goal by continuously estimating the throughput TCP would get on its paths using a simple model of TCP throughput, $B = \sqrt{\frac{2}{p}} \cdot \frac{MSS}{RTT}$. With this estimate, MPTCP adjusts its overall aggressiveness (total congestion window increase per RTT over all its subflows) so that it achieves at least the throughput of the best TCP. To achieve the second goal, MPTCP gives more of the increase to subflows with smaller loss rates.

In scenario 1, the throughput via AP_2 is higher than AP_1, resulting in a lower loss rate on the corresponding subflow and making the MPTCP sender send most of its traffic via AP_2. In scenario 2, other bottlenecks reduce

the throughput available via AP_2, and the load balancing of traffic over paths will depend on the amount of loss experienced on AP_2. Either way, MPTCP will use its estimate of throughput available over AP_1 to ensure it achieves at least as much in aggregate.

Our target is to help MPTCP achieve the same goals when the two subflows via AP_1 and AP_2 interfere. For this to happen, we use ECN to signal the sender that AP_1 is worse and should be avoided when possible. Just making traffic move away from AP_1 is simple: the client will simply mark a large fraction of packets (e.g. 10%) received via AP_1 with the Congestion Experienced codepoint, which will duly push the sender to balance most of its traffic via AP_2. However, this approach will backfire in scenario 2, where MPTCP will stick to AP_2 and receive worse throughput.

To achieve the proper behavior in all these scenarios, the rate of ECN marks sent over AP_1 must be chosen carefully such that it does not affect MPTCP's estimation of TCP throughput via AP_1. Our goal is to ensure that the **MPTCP connection gets at least as much throughput as it would get via AP_1 if the latter is completely idle**. In particular, say the rate of ECN marks the client adds is δ. As the TCP congestion window depends on loss rate, the congestion window will decrease when we increase the loss rate. For the bandwidth to remain constant, we would like RTT_δ, the RTT after marking, to also decrease. In other words we would like for the following equation to hold:

$$B = \sqrt{\frac{2}{p}} \cdot \frac{MSS}{RTT} = \sqrt{\frac{2}{p+\delta}} \cdot \frac{MSS}{RTT_\delta} \qquad (3)$$

We assume the subflow via AP_1 is the unique subflow at that AP; congestion control at the sender will keep AP_1's buffer half-full on average. Thus, the average RTT observed by the client can be decomposed as $RTT = RTT_0 + \frac{BUF}{2} \cdot T_1$, where RTT_0 is the path RTT, and the second term accounts for buffering. Note that we use T_1, the average packet delivery time for AP_1 estimated by our metric. If our client reduced its RTT to RTT_0 by decreasing its congestion window, it would still be able to fill the pipe, and more importantly it would allow the sender to correctly estimate the bandwidth via AP_1. Using these observations and simplifying the terms, we rewrite the equation above as:

$$B = \sqrt{\frac{1}{p}} \cdot \frac{1}{RTT} = \sqrt{\frac{1}{p+\delta}} \cdot \frac{1}{RTT - T_1 \cdot \frac{BUF}{2}} \qquad (4)$$

Finally, knowing T_1 gives us an estimate of the maximum bandwidth $B = \frac{1}{T_1}$. We now have two equations with three unknowns: p, δ and BUF. Fortunately, we don't need to know the exact value of BUF; using a smaller value will only lead to a smaller value for δ, reducing our ability to steer traffic away from AP_1. To get an estimate of BUF, we note that nearly all wireless APs are buffered to offer 802.11a/g capacity (25Mbps) to single clients downloading from servers spread across the globe (i.e. an RTT of at least 100ms). This implies the smallest buffers should be around 2.5Mbit, which is about 200 packets of 1500 bytes. We use 200 as our estimate for BUF, and can now solve the two equations for δ. The closed form we arrive at is:

$$\delta = \frac{1}{2} \cdot \left(\frac{50 \cdot T_1^2}{RTT \cdot (RTT - 50 \cdot T_1)}\right)^2 \qquad (5)$$

δ depends on the interface (T_1) and the RTT of the subflow that will be marked, both of which are available at the client. Note that δ provides a maximum safe marking rate, and the actual marking rate used may be lower. For instance, marking rates in excess of 5% brings the TCP congestion window down to around 6 packets and makes TCP prone to timeouts.

In our implementation, the client computes the estimation of δ for every AP it is connected to. The client monitors the values of T_i for all of its interfaces, and sets ECN marks for subflows routed via interfaces with a packet time at least 20% larger than the best packet time across all interfaces. The 20% threshold is chosen to avoid penalizing good APs for short fluctuations in performance ECN marking happens before the packets get delivered to IP processing code at the client.

6 Evaluation

We have implemented our solution in the Linux 3.5.0 kernel, patched with MPTCP v0.89. Most of the changes are in the 802.11 part of the Linux kernel, and are independent of the actual NIC used. The patch has 1.3KLOC, and it includes code to compute the packet time for each wireless interface, the ECN marking algorithm, and channel switching support.

In this section we analyze each of our contributions in detail both experimentally and, where appropriate, in simulation. We first measure our client-side link estimation technique in a controlled environment. Next, we analyze the marking algorithm using 802.11n in the lab, and extensively in simulation to find it provides close to optimal throughput (90%) over a wide range of parameters. Next, we analyze fairness to existing clients and performance for short downloads. Finally, we run preliminary mobility tests "in-the-wild" using APs we do not control, finding that our solution does provide near-optimal throughput in real deployments.

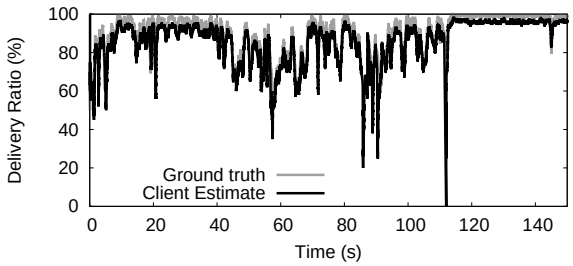


Figure 5: Client-side estimation of PHY delivery probability in 802.11a, fixed rate (54Mbps)

6.1 Client-side link quality estimation

To test the accuracy of our client-side estimation of PHY delivery probability, we setup one of our APs in 802.11a, place a sniffer within 1m of the AP, and place the client around 10m away from the AP. The AP's rate control algorithm is disabled, and we set the MCS to 54Mbps.

Both the client and the sniffer measure the average delivery ratio over a half-second window. The size of the window is a parameter of our algorithm: larger values take longer to adapt to changing channel conditions, while smaller values may result in very noisy estimations. Half a second provides a good tradeoff between noise and speed of adaptation. MPTCP congestion control (we use the OLIA algorithm [36]) reacts fairly quickly to moderate differences in loss rates (20% higher loss rate on one path). Experiments show that it takes between 1s and 2s for traffic to shift to the better path once an abrupt change has occurred, when the RTT is 50ms.

The client downloads a large file and we plot its estimation of the delivery probability (relation (2)) against the ground truth, as observed at the packet sniffer near the sender. Two minutes into the experiment we increase the transmit power of the AP to the max, thus improving the delivery probability. The results are given in Figure 5 and show that the client's estimation closely tracks the actual delivery ratio, and the mean error across the whole test is around 3%. We ran many experiments with 802.11g/n and observed similar behavior: client side estimation closely tracks the ground truth, and the mean error rate was under 5% in all our experiments.

Our metric is based on the assumption that the delivery ratio is independent of the state of the packet (the number of retries). This assumption is reasonable when packet losses occur due to channel conditions, but breaks down in hidden terminal situations, where a collision on the first packet will most likely trigger collisions on subsequent packets. In such cases, our metric's focus only on the initial transmissions will lead to errors, as follows:

- When competing traffic is sparse, our metric will overestimate the PHY delivery probability by around 10% in our tests.

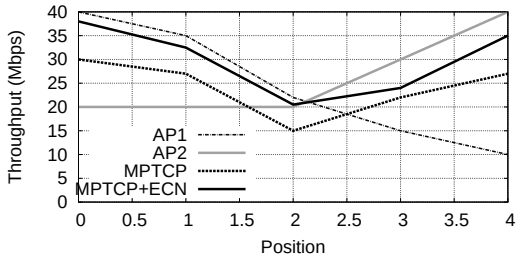


Figure 7: Throughput of a nomadic client at different position between AP1 and AP2 in 802.11n. MPTCP with ECN marking provides 85% of the optimal throughput.

- In heavy contention, one AP may be starved completely, and our client's estimate will be unreliable.

This drawback does not affect our overall solution: **we need client-side estimation only when the two APs are in carrier sense**. When in hidden terminal, our experiments show that the interaction between the MAC and Multipath TCP leads to a proper throughput allocation, and no further intervention via ECN is needed.

When a rate control algorithm picks a different rate to resend the same packet, that packet will not have its "retry" bit set despite being retransmitted. To understand whether this affects our results, we ran experiments as above but with rate control enabled; however the were no discernible differences in the efficacy of our algorithm.

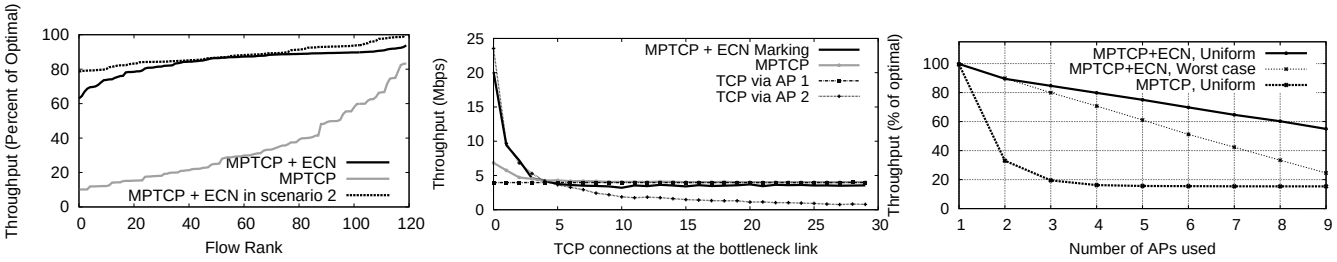
6.2 ECN Marking

We reran all the 802.11a/g experiments presented so far with our client-side load balancing algorithm on. We found that the marking did not influence the actual results: in particular, we verified that marking was not triggered in the channel diversity setup we discussed before.

For a static 802.11n client, we applied the ECN marking as indicated by relation (5). The results shown in Fig. 7 reveal that our metric and ECN algorithms work well together, pushing traffic away from the inefficient AP. Using the same setup, we then moved the client at walking speed between the APs, as the whole distance was covered in around 10s. The results (not shown) are much noisier, but show that the ECN mechanism still works fairly well overall; a similar result with a mix of 11n and 11g is later discussed in Figure 8. All further experiments presented in this paper are run with the ECN algorithm enabled, unless otherwise specified.

6.2.1 Simulation analysis

To understand how our proposed marking algorithm works in a wide range of scenarios, we also implemented it *htsim*, a scalable simulator that supports MPTCP and that has been used in previous works [37, 38]. *htsim*



(a) Flow throughput across a wide range of parameters. Marking achieves on average 85% of the optimal throughput.

(b) ECN Marking delivers near optimal throughput for scenario 2.

(c) Connecting to more APs reduces the total available throughput.

Figure 6: ECN simulation in *htsim*.

does not model 802.11; instead, we implemented a new type of queue that models the high level behavior of shared access in 802.11 DCF, namely: different senders' (AP) packets take different time for transmission on the medium, and when multiple senders have packets ready, packets are chosen according to a weighted fair-queueing algorithm, with predefined weights for the different APs.

Using this simple model, we can explore specified outcomes in the MAC contention algorithm, for example when AP1 wins contention four times more often than AP2, that are difficult to obtain in 802.11 setups simply by choosing different rate selection algorithm. Our simulated topology is shown in Fig. 4a, where the client is using both AP_1 and AP_2. In all our tests, AP_2 offers a perfect 802.11a/g connection (max 22Mbps), meaning that T_2, the packet time for AP_2 is set to 0.5ms.

We ran simulations testing all the combinations of parameters important in practice:: RTT (10, 25, 50 and 100ms), T_2 (from 1ms to 6ms), and the weights for different APs (1, 2, 4, 8 or 16). We ran a total of 120 simulations, and we present the throughput obtained as percentage of the optimal, sorting all values ascendingly. Figure 6a shows that the ECN marking algorithm is very robust: its average performance is 85% of the optimal (median is 87%), and its worst case is 65%. In contrast, MPTCP alone fares poorly: 34% throughput on average (28% in the median). Finally, the throughput of MPTCP in Scenario 2 is also robustly close to the optimal: average at 84% and median at 88%.

There are parameter combinations where the ECN algorithm is not as effective: when RTTs are small, δ is fairly high so ECN does manage to reduce the congestion window over AP_1. However, even a very small window of packets sent via AP_1 is enough to eat up a lot of airtime that would be better used by AP_2, and this effect is more pronounced when AP_1 wins the contention more often, because it has fewer retries.

In all experiments, we cap the marking rate to a maximum of 5% to avoid hurting throughput in Scenario 2. This is a direct tradeoff: the higher the allowed rate, the better MPTCP behaves in scenario 1, but the worse it behaves in scenario 2. The reason for this behavior is that the traditional formula used by MPTCP to estimate throughput over its subflows is overly optimistic for higher loss rates, where retransmit timeouts begin to dominate throughput.

To analyze scenario 2, we use a setup where $T_1 = 3ms$ (approx. 4Mbps), $RTT = 25ms$ and vary the number of TCP flows contending for the uplink of AP_2, whose speed is set to $25Mbps$. Figure 6b shows that MPTCP alone fails to deliver when the AP_2 uplink is idle, but obtains the maximum possible throughput when AP_2's uplink is busy (same as TCP over AP_2). MPTCP with ECN marking gets the best of both worlds: it closely tracks the performance of a single TCP flow via AP_2 when there is little contention for AP_2's uplink, and it stabilizes to just under 4Mbps when AP_2 uplink is congested.

Increasing the number of APs. We've looked at connecting to two APs until now. What happens when there are more APs the client can connect to? We ran an experiment where the best AP offers maximum rate, and we are adding a varying number of other APs. In our first experiment, we consider a worst case where all the added APs are poor: their packet times is set to $6ms$ (2Mbps); in our second experiment we distribute the packet times of the APs uniformly between $0.5ms$ and $6ms$, mimicking a more realistic situation, and plot both results in Figure 6c. The results show a linear drop in the throughput obtained as the number of APs increases when ECN is used, however the slope is steeper when all APs have poor performance. This graph shows that connecting to more than 3-4 APs, is a bad idea: the client should choose the few best APs and connect to those alone.

6.3 A mobility experiment

We now discuss a mobility experiment run in a building with mixed WiFi coverage: the user starts from a lab on the second floor of the building, goes down a flight of stairs and then walks over to the cafeteria. En route, the mobile is locked on channel 6 and can associate to five different APs, each covering different parts of the route. We repeat the experiment several times, each taking of around one minute, during which the client is ei-

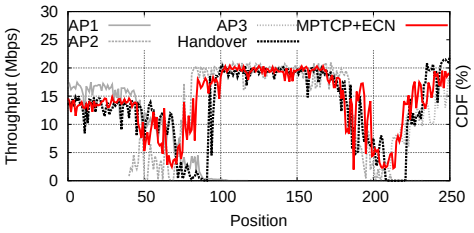


Figure 8: Mobility experiment: indoor client walking speed.

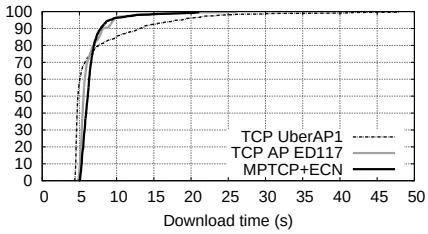


Figure 9: Static clients experience p formance variations outside their contr MPTCP offers predictable performance

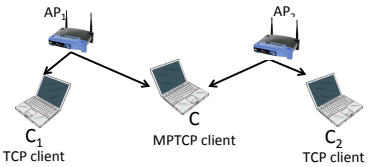


Figure 10: Experimental setup to test fairness

ther: a) using one AP at a time, with standard TCP; b) using MPTCP and associating to all APs it sees all the time; or c) performing handover between the different APs by using MPTCP. Our client monitors the beacons received over a 2s period, and switches to a new AP when it receives Δ more beacons than the current AP. It is well known that TCP performance suffers in cases of frequent mobility [39]. The same effect happens during MPTCP handovers, when a new subflow is created and has to do slowstart after switching to a new AP. In-between APs the client may flip-flop between APs based on its observed beacon count, reducing overall performance. To avoid this effect, we experimentally chose $\Delta = 10$.

In another experiment, the client slowly moves through the building at 1km/h, and the results are shown in Fig.8. At the beginning of the walk, the client has access to two Linux APs running minstrel, and these are also accessible briefly on the stairs, in the middle of the trace. The departments' deployment of uncoordinated APs (Fortinet FAP 220B) are available both in the lab at very low strength, on the stairs, and closer to the cafeteria. Our mobile client connects to at most three APs simultaneously. Throughout the walk, the throughput of our MPTCP mobile client closely tracks that of TCP running on the best AP in any given point; the handover solution suffers because it uses a *break−before−make* strategy and throughput drops to nearly zero for 5-10s. We also noticed that in the beginning of the trace our ECN-based load balancing algorithm penalizes the subflow over the department AP—if we disable it, the throughput of MPTCP in that area drops dramatically.

6.4 Static clients

Our experiments so far show that connecting to multiple APs pays off when mobile. Is it worth doing the same when the client is static? We had our client connect to two APs (channel 11) visible in our lab and that we do not control, and the performance from both APs is similar. Our client repeatedly downloads the same 10MB file from one of our servers using either TCP over AP1, TCP over AP2 or MPTCP+ECN over both APs. We ran this experiment during 5 office hours, and present a CDF of the throughputs obtained in Figure 9. The figure shows there is a long tail in the throughput obtain via either AP because of external factors we do not control: other WiFi users, varying channel conditions, etc. The median download time for AP1 is 5s, 5.6s via AP2 and 6s with MPTCP (20% worse). However, MPTCP reduces the tail, cutting the 99% download time in half.

Power consumption While connected to different APs, the solution adds the following per AP costs: association and authentication handshakes, DHCP, and the null frames required whenever the mobile goes in and out of power save. These are negligible, as the bulk of the cost is due to the radio processing and the TCP/IP stack [40]. The energy cost of our solution is therefore dependent on the actual throughput achieved, which is near-optimal.

6.5 Fairness

We have focused our analysis so far on the performance experienced by a client connecting to multiple APs, and showed that there are significant benefits to be had from this approach. We haven't analyzed the impact this solution has on other clients: is it fair to them? Does our solution decrease the aggregate throughput of the system? In answering these questions, our goals are neither absolute fairness (WiFi is a completely distributed protocol and also inherently unfair), nor maximizing aggregate throughput (which may mean some distant clients are starved). Rather, we want our solution's impact on other clients to be *no worse than that of a TCP connection using the best available AP*.

The theory of MPTCP congestion control reassures us that two or more subflows sharing the same wireless medium will not get more throughput than a single TCP connection would. Also, if an AP has more customers, Multipath TCP will tend to steer traffic away from that AP because it sees a higher packet loss rate.

We used the setup shown in Figure 10 to test the behavior of our proposal. There are two APs each with a TCP client in their close vicinity, using 802.11a, and an MPTCP client *C* using both APs.

The first test we run has both APs using maximum power: when alone, all clients will achieve the maxi-

C conn. to	AP_1-C_1	AP_2-C_2	C
AP_1 (TCP)	5	10	5
AP_2 (TCP)	10	5	5
AP_1&AP_2	7	7	7

Table 1: APs&clients in close range: MPTCP provides perfect fairness (802.11a, throughput in Mbps).

C conn. to	AP_1-C_1	AP_2-C_2	C
AP_1(TCP)	3.5	13	3.5
AP_2(TCP)	10	5	5
AP_1&AP_2	10	5	5

Table 2: Client close to AP_2: MPTCP client behaves as TCP connected to AP_2

C conn. to	AP_1-C_1	AP_2-C_2	C
AP_1(TCP)	3.5	13.5	3
AP_2(TCP)	14	3	3
AP_1&AP_2	8.5	6.5	5

Table 3: Client in-between APs: MPTCP client improves overall fairness

mum rate in 802.11a, around 22Mbps. The results of the experiment are shown in table 1: when the client connects to both APs, the system achieves perfect fairness. In comparison, connecting to either AP alone will lead to an unfair bandwidth allocation. In this setup, MPTCP congestion control matters. If we use regular TCP congestion control on each subflow, the resulting setup is unfair: the MPTCP client receives 10Mbps while the two TCP clients each get 5Mbps.

We next study another instance of the setup in Fig. 10 where the APs, still in CS, are farther away, and while the TCP clients remain close to their respective APs, they get a lesser channel to the opposite AP. First, we place C close to AP_2. When C connects to AP_1, which is farther, it harms the throughput of C_1, and the overall fairness is greatly reduced. When C connects to both APs, its traffic flows via the better path through AP_2, offering optimal throughput without harming fairness (Table 2). When the client is between the two APs, traffic is split on both paths and the overall fairness is improved, while also increasing the throughput obtained by our client (Table 3).

In summary, by connecting to both APs at the same time and splitting the downlink traffic between them, MPTCP achieves better fairness in most cases, and never hurts traffic more than a TCP connection would when using the best AP.

7 Channel-switching

To connect to APs on different WiFi channels, clients can use channel switching, a technique supported by all NICs for probing. This technique, was proposed and shown to work in previous work [18, 19, 20, 21]; We implement a similar procedure, but with adaptation based on the actual bandwidth obtained on each channel.

Say the client spends a slot c_i on channel i, such that the sum of all slots equals the global duty cycle $C = \sum_i c_i$. While on channel i, the client measured the bandwidth it receives on that channel, b_i, by counting the number of bytes received in a slot and dividing it by c_i. We consider the following family of algorithms for channel switching:

$$c_i = \frac{b_i^\alpha}{\sum_j b_j^\alpha} \cdot C \tag{6}$$

The equation prescribes how to compute c_i for the next interval based on the throughput observed in the current interval, where the interval is a multiple of C. α dictates how aggressively we prefer the good channels over the bad ones: higher values lead to more time spent on the best channels. Choosing α strikes a tradeoff between throughput obtained and accurate probing that enables quick adaptation in channel conditions.

The discussion so far has assumed MPTCP is able to allow all APs on different channels to send at flat rate during their slot; in other words Multipath TCP manages to keep all the paths busy. Also note that there are no direct interactions between the MACs of the different APs during this time: enabling MPTCP to work over channel switching is a much easier task. All we need is to make sure the MPTCP subflows do not suffer frequent timeouts, which can occur due to:

- Wildly varying round-trip times leading to inaccurate values of the smoothed RTT estimator.
- Bursts of losses suffered when congestion windows are small and fast retransmit is not triggered.

The first problem is quite likely to appear during channel switching, as the senders will see bimodal RTT values: path RTT for packets sent during the channel's slot, and C for packets sent while outside the slot. To avoid this problem, we impose that C is smaller than the smallest possible RTO at clients, which must be higher than the delayed ACK threshold (200ms). Hence, our first restriction is that $C \leq 200ms$.

To avoid the second problem, we lower bound the time spent on any channel to a minimum value that allows the AP to send at least one packet per slot; this implies that the smallest slot has to be at least 10ms.

We have implemented channel switching support in the Linux kernel, together with the family of algorithms discussed above. With this implementation, we ran a series of experiments to understand the basic performance of channel switching in our context. We have the client associate to two APs in (channels 40 and 44, 802.11a) and modify the transmit power of the APs while we observe the adaptation algorithms at work. The results are shown in the table below. The experiments with both APs set at max power show that the channel switching overheads (of around 5ms in our measurements) reduce the total available throughput by around 10% when switching between two channels with a duty cycle of $200ms$. If we decrease the power of AP2, $\alpha = 2$ does

a good job of increasing the slot of AP1, and obtaining 87% of the optimal throughput. In contrast, the algorithm $\alpha = 0$ assigns equal slot to both APs and throughput is the average of both APs' throughput:

Power for AP1&AP2	TCP AP1	TCP AP2	MPTCP + switch $\alpha = 2$	MPTCP + switch $\alpha = 0$
Max & Max	20	20	18	18
Max & Low	20	14	17.5	16
Max & Low	20	5	17.5	12

The experiments show that MPTCP and channel switching play nicely together. We note that the experiments work similarly regardless the WiFi standard used. Our driver independent channel switching procedure, through its adaptive slot, makes it possible for an MPTCP based mobile to access capacity on independent channels in a fluid manner.

7.1 Channel switching with many users

The one key difference between the single channel and multi channel scenarios is the behavior when multiple users are connected to the same APs. When on the same channel, users tend to stick to the AP closest to them as our experiments showed in Section 4. When switching, the clients are not coordinated and will affect each other's throughput, depending on how their slots overlap. Intuitively, when multiple clients make independent switching decisions we expect the overall channel utilization to be suboptimal. We resort to simulation to understand these effects. We model a number of mobile clients connected to three APs on three distinct channels, and all clients can download from every AP at 22Mbps (i.e. the 802.11g perfect channels). The optimum is for the clients' speeds to sum up to 66Mbps. With channel switching, however, clients' slots will overlap and some channels will be idle while others may have two clients using them simultaneously. Our simulator uses a simplified model that assumes no channel switch overheads, and that bandwidth is shared equally amongst all clients using a channel. When computing slot times, we also add a number of ms chosen uniformly at random between 1-10ms, to model for random delays experienced by channel switching code due to interactions with the AP [18]. In the table below we plot the average throughput obtained as a percentage of the optimum.

Users	1	2	3	4	5	7	10
$\alpha = 2$	100	80	64	70	76	81	87

The results show worst results when three users contend for three channels and a third of the capacity is lost; if fewer or a lot more users share the channels, the effects are less pronounced. Note that these results also hold for the single channel setting, when the AP backhaul is the bottleneck (i.e. DSL scenarios).

8 Conclusion & Future work

We are witnessing a paradigm shift for wireless mobility: instead of using a single AP at any one time and racing to quickly change to the next when signal fades, the emerging MPTCP standard allows clients to use multiple APs in range and achieve a *truly mobile experience, independent of L2 handover*. Our main contribution in this paper is understanding the interaction between the WiFi MAC and MPTCP, and optimizing it for mobility.

Our experiments have shown that, in many cases, the load balancing job is done by the WiFi MAC (our carrier-sense experiments with 802.11a/g) or by interactions of the MAC and MPTCP congestion control (hidden terminal). However, there are situations when connecting to multiple APs can hurt throughput. We have offered a solution to these cases that utilizes a novel client-side measurement method together with an algorithm that uses ECN marking to enable the sender congestion controller to balance of traffic to the most efficient AP.

We have implemented and tested our solution both in simulation and in the Linux kernel. Nomadic and mobile experiments show our solution gets near-optimal throughput and is robust to changes in AP quality be they from client mobility or other factors.

In future work we plan to incorporate heuristics that allow WiFi clients to quickly associate to APs in vehicular mobility scenarios, as proposed by [41, 21]; a wider range of mobility experiments is also needed to ensure we have covered all the relevant situations. Additionally, we need to deploy our solution on mobile devices and quantify the energy consumption of our proposal, particularly in the channel switching scenario.

Our end-goal is to build a usable mobility solution which will combine channel switching as well as associating to multiple APs on the same channel. Understanding how our single channel solutions interact with switching is area worth of exploration.

Acknowledgements: This work was partly funded by Trilogy 2, a research project funded by the European Commission in its Seventh Framework program (FP7 317756), and by Mobil4, a research project funded by the Romanian government (UEFISCDI 11/2012). We also thank the anonymous reviewers and our shepherd Venkat Padmanabhan for their insightful feedback.

References

[1] A. Miu, G. Tan, H. Balakrishnan, and J. Apostolopoulos, "Divert: fine-grained path selection for wireless lans," in *Proceedings of the 2nd international conference on Mobile systems, applications, and services*, pp. 203–216, ACM, 2004.

[2] R. Murty, J. Padhye, R. Chandra, A. Wolman, and B. Zill, "Designing high performance enterprise Wi-Fi networks.," in *NSDI*, vol. 8, pp. 73–88, 2008.

[3] B. T. F. Network, "http://www.btfon.com/."

[4] A. Farshad, M. K. Marina, and F. Garcia, "Urban wifi characterization via mobile crowdsensing," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pp. 1–9, IEEE, 2014.

[5] A. S. Engineering and Q. Associates, "Study on the use of wi-fi for metropolitan area applications." Final Report for Ofcom, April 2013.

[6] V. Brik, A. Mishra, and S. Banerjee, "Eliminating handoff latencies in 802.11 WLANs using multiple radios: Applications, experience, and evaluation," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, (Berkeley, CA, USA), pp. 27–27, USENIX Association, 2005.

[7] A. Mishra, M. Shin, and W. Arbaugh, "An empirical analysis of the IEEE 802.11 MAC layer handoff process," *SIGCOMM Comput. Commun. Rev.*, vol. 33, April 2003.

[8] I. Ramani and S. Savage, "SyncScan: practical fast handoff for 802.11 infrastructure networks," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 1, pp. 675–684, IEEE, 2005.

[9] Y.-S. Chen, M.-C. Chuang, and C.-K. Chen, "Deucescan: Deuce-based fast handoff scheme in IEEE 802.11 wireless networks," *Vehicular Technology, IEEE Transactions on*, vol. 57, pp. 1126–1141, March 2008.

[10] M. Shin, A. Mishra, and W. A. Arbaugh, "Improving the latency of 802.11 hand-offs using neighbor graphs," in *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, (New York, NY, USA), pp. 70–83, ACM, 2004.

[11] "Virtual cells: The only scalable multi-channel deployment." MERU white paper, 2008.

[12] "IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 2: Fast basic service set (BSS) transition," *IEEE Std 802.11r-2008 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008)*, pp. 1–126, July 2008.

[13] "IEEE standard for information technology– local and metropolitan area networks– specific requirements– part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications amendment 1: Radio resource measurement of wireless LANs," *IEEE Std 802.11k-2008 (Amendment to IEEE Std 802.11-2007)*, pp. 1–244, June 2008.

[14] S. Pack, J. Choi, T. Kwon, and Y. Choi, "Fast-handoff support in IEEE 802.11 wireless networks," *IEEE Communications Surveys and Tutorials*, vol. 9, no. 1-4, pp. 2–12, 2007.

[15] A. C. Snoeren and H. Balakrishnan, "An end-to-end approach to host mobility," in *Proc. Mobicom*, ACM, 2000.

[16] W. M. Eddy, "At what layer does mobility belong?," *Communications Magazine, IEEE*, vol. 42, no. 10, pp. 155–159, 2004.

[17] R. Chandra and P. Bahl, "Multinet: Connecting to multiple IEEE 802.11 networks using a single wireless card," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 2, pp. 882–893, IEEE, 2004.

[18] S. Kandula, K. C.-J. Lin, T. Badirkhanli, and D. Katabi, "FatVAP: aggregating AP backhaul capacity to maximize throughput," in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, (Berkeley, CA, USA), pp. 89–104, USENIX Association, 2008.

[19] D. Giustiniano, E. Goma, A. Lopez, and P. Rodriguez, "WiSwitcher: an efficient client for managing multiple APs," in *Proceedings of the 2nd ACM SIGCOMM workshop on Programmable routers for extensible services of tomorrow*, pp. 43–48, ACM, 2009.

[20] A. J. Nicholson, S. Wolchok, and B. D. Noble, "Juggler: Virtual networks for fun and profit," *IEEE Transactions on Mobile Computing*, vol. 9, pp. 31–43, Jan. 2010.

[21] H. Soroush, P. Gilbert, N. Banerjee, B. N. Levine, M. Corner, and L. Cox, "Concurrent wi-fi for mobile users: Analysis and measurements," in *Proceedings of the Seventh COnference on Emerging Networking EXperiments and Technologies*, CoNEXT '11, (New York, NY, USA), pp. 4:1–4:12, ACM, 2011.

[22] A. J. Nicholson, Y. Chawathe, M. Y. Chen, B. D. Noble, and D. Wetherall, "Improved access point selection," in *Proceedings of the 4th International Conference on Mobile Systems, Applications and Services*, MobiSys '06, (New York, NY, USA), pp. 233–245, ACM, 2006.

[23] S. Vasudevan, K. Papagiannaki, C. Diot, J. Kurose, and D. Towsley, "Facilitating access point selection in IEEE 802.11 wireless networks," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurements*, pp. 26–26, USENIX Association, 2005.

[24] S. Shakkottai, E. Altman, and A. Kumar, "Multihoming of users to access points in WLANs: A population game perspective," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 6, pp. 1207–1215, 2007.

[25] B. alexander Cassell, T. Alperovich, M. P. Wellman, and B. Noble, "Access point selection under emerging wireless technologies," 2011.

[26] O. B. Karimi, J. Liu, and J. Rexford, "Optimal collaborative access point association in wireless networks," in *Proc. IEEE INFOCOM 2014*, 2014.

[27] A. Miu, H. Balakrishnan, and C. E. Koksal, "Improving loss resilience with multi-radio diversity in wireless networks," in *Proceedings of the 11th annual international conference on Mobile computing and networking*, pp. 16–30, ACM, 2005.

[28] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," rfc6824, IETF, 2013.

[29] A. Balasubramanian, R. Mahajan, A. Venkataramani, B. N. Levine, and J. Zahorjan, "Interactive wifi connectivity for moving vehicles," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 427–438, 2008.

[30] A. Kochut, A. Vasan, A. U. Shankar, and A. Agrawala, "Sniffing out the correct physical layer capture model in 802.11 b," in *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, pp. 252–261, IEEE, 2004.

[31] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11 b mesh network," in *ACM SIGCOMM Computer Communication Review*, vol. 34, pp. 121–132, ACM, 2004.

[32] S. Choi, K. Park, and C.-k. Kim, "On the performance characteristics of WLANs: Revisited," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, pp. 97–108, June 2005.

[33] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, vol. 2, pp. 836–843, IEEE, 2003.

[34] D. Giustiniano, E. Goma, A. Lopez Toledo, I. Dangerfield, J. Morillo, and P. Rodriguez, "Fair wlan backhaul aggregation," in *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking*, MobiCom '10, (New York, NY, USA), pp. 269–280, ACM, 2010.

[35] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," in *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, NSDI'11, (Berkeley, CA, USA), pp. 8–8, USENIX Association, 2011.

[36] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, and J.-Y. Le Boudec, "Mptcp is not pareto-optimal: Performance issues and a possible solution," in *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, (New York, NY, USA), pp. 1–12, ACM, 2012.

[37] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. Usenix NSDI 2011*.

[38] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath tcp," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (New York, NY, USA), pp. 266–277, ACM, 2011.

[39] P. Manzoni, D. Ghosal, and G. Serazzi, "A simulation study of the impact of mobility on tcp/ip," in *Network Protocols, 1994. Proceedings., 1994 International Conference on*, pp. 196–203, Oct 1994.

[40] A. Garcia-Saavedra, P. Serrano, A. Banchs, and G. Bianchi, "Energy consumption anatomy of 802.11 devices and its implication on modeling and design," in *Proceedings of the 8th International Conference on Emerging Networking Experiments*

and Technologies, CoNEXT '12, (New York, NY, USA), pp. 169–180, ACM, 2012.

[41] J. Eriksson, H. Balakrishnan, and S. Madden, "Cabernet: vehicular content delivery using wifi," in *Proceedings of the 14th ACM international conference on Mobile computing and networking*, pp. 199–210, ACM, 2008.