RAVEN: Improving Interactive Latency for the Connected Car

HyunJong Lee
University of Michigan
hyunjong@umich.edu

Jason Flinn
University of Michigan
jflinn@umich.edu

Basavaraj Tonshal
Ford Motor Co.
btonshal@ford.com

ABSTRACT

Increasingly, vehicles sold today are connected cars: they offer vehicle-to-infrastructure connectivity through built-in WiFi and cellular interfaces, and they act as mobile hotspots for devices in the vehicle. We study the connection quality available to connected cars today, focusing on user-facing, latency-sensitive applications. We find that network latency varies significantly and unpredictably at short time scales and that high tail latency substantially degrades user experience. We also find an increase in coverage options available due to commercial WiFi offerings and that variations in latency across network options are not well-correlated.

Based on these findings, we develop RAVEN, an in-kernel MPTCP scheduler that mitigates tail latency and network unpredictability by using redundant transmission when confidence about network latency predictions is low. RAVEN has several novel design features. It operates transparently, without application modification or hints, to improve interactive latency. It seamlessly supports three or more wireless networks. Its in-kernel implementation allows proactive cancellation of transmissions made unnecessary through redundancy. Finally, it explicitly considers how the age of measurements affects confidence in predictions, allowing better handling of interactive applications that transmit infrequently and networks that exhibit periods of temporary poor performance. Results from speech, music, and recommender applications in both emulated and live vehicle experiments show substantial improvement in application response time.

CCS CONCEPTS

• **Networks** → *Mobile networks*; • **Computer systems organization** → *Redundancy*;

MobiCom'18, October 29–November 2, 2018, New Delhi, India

ACM ISBN 978-1-4503-5903-0/18/10…$15.00
https://doi.org/10.1145/3241539.3241571

KEYWORDS

Vehicular networking; Redundancy; Network latency

ACM Reference Format:
HyunJong Lee, Jason Flinn, and Basavaraj Tonshal. 2018. RAVEN: Improving Interactive Latency for the Connected Car. In *MobiCom '18: 24th Annual International Conference on Mobile Computing and Networking, October 29–November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3241539.3241571

1 INTRODUCTION

Increasingly, vehicles sold today are "connected cars[1]." They have multiple built-in cellular [5, 7, 16, 48] and WiFi [11, 17, 48] interfaces that allow applications running on the vehicle human-machine interface (HMI) to connect to the Internet. Connected cars also act as mobile hotspots, so that passenger mobile devices can also connect via the vehicle's built-in network interfaces. Finally, tethering allows a mobile device to export its network interfaces for use by other devices within the vehicle. Thus, there is increasingly a plethora of wireless connectivity options available.

Many applications running on the HMI and passenger cellphones are user-facing and latency-sensitive; e.g. speech recognition and recommendation services such as Yelp. Unfortunately, wireless network performance from moving vehicles is notoriously unpredictable. Frequent disconnections and high tail latencies have been noted by prior studies [6, 8, 10, 13, 14, 28, 39, 40, 46]. For interactive applications, these problems manifest as unacceptable delays that degrade the user experience.

We begin our work with a detailed study of network quality for vehicle-to-infrastructure (V2I) communication. Our study focuses on supporting user-facing, latency-sensitive applications in a variety of scenarios (city, suburban, highway, and rural driving). We note several important findings. Although almost no open (free, without splash screen) WiFi access points are available, commercial WiFi offers valuable connectivity for suburban and urban driving. Network round-trip time (RTT) is very stable in aggregate, but varies substantially from second to second: a given network remains the lowest-latency option for a median duration of

[1]Note that while the definition of "connected car" may include both vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication, we focus only on the latter in this paper.

only two seconds. Thus, predicting future quality, or even the lowest RTT network, is quite challenging. Further, tail latency (e.g., at the 95th percentile of the RTT distribution) is very high. Yet, high RTTs are rarely correlated across networks, so one network could possibly mask high RTTs in another.

Based on our study, we created RAVEN,[2] an in-kernel MPTCP scheduler that uses redundant communication over multiple networks to reduce interactive delay for applications and mask high tail latency. RAVEN uses passive monitoring to measure RTT at the TCP subflow layer and predicts RTT for future communication. It explicitly calculates its confidence in each prediction and employs additional networks to send data when confidence intervals overlap. Thus, RAVEN sends data over a single network when it is confident that the network will offer the lowest RTT, but it employs multiple networks when it is unsure which network will be superior.

Compared to recent systems that also employ redundancy to mask uncertainty in wireless communication [20, 26], RAVEN has several important innovations. First, unlike prior systems, RAVEN requires no hints about transmission size and works with unmodified applications; it inspects kernel data structures such as congestion windows and queue sizes to automatically switch between redundant and non-redundant transmission. Second, RAVEN generalizes to three (or more) wireless networks, whereas these prior systems are fundamentally limited to using two networks. Third, RAVEN decreases its confidence in measurements as they become more stale, which provides a more efficient and elegant method than active probing for checking whether poorly-performing networks have improved. Fourth, confidence intervals provide an intuitive mechanism to tune the tradeoff between data usage and network performance. Finally, whereas prior systems have been implemented at user-level, RAVEN's in-kernel implementation creates several opportunities to proactively cancel work that becomes useless due to redundancy.

We evaluate RAVEN with three latency-sensitive applications: speech recognition, music streaming, and Yelp recommendation. Emulated and live vehicle results show that RAVEN reduces application latency for all applications across a diverse set of driving scenarios. In live vehicle experiments, RAVEN speeds up median application response time from 46% to more than a factor of 3, as compared to MPTCP. 95% tail response time is 2-11 times faster.

The contributions of this paper are:

- A study of vehicle-to-infrastructure connectivity focusing on latency-sensitive applications and examining the potential of using multiple wireless interfaces.
- Design of an in-kernel MPTCP scheduler that employs multiple networks to mitigate prediction uncertainty and high tail latencies through strategic redundancy.
- An exploration of using confidence in network predictions for tasks such as probing poorly-performing networks, tuning the tradeoff between performance and data usage, and determining when redundant transmission is appropriate.

[2]Redundancy-Aided VEhicular Networking

2 RELATED WORK

RAVEN builds on prior works that reduce interactive latency via redundant multipath communication, study wireless network performance for vehicle-to-infrastructure communication, and explore MPTCP scheduler design and implementation. We discuss each in turn.

DEMS [20] and Meatballs [25, 26] both reduce completion time in multipath communication by transmitting data redundantly. Although both use redundancy to mask prediction uncertainty, RAVEN has several important innovations. First, both prior systems require extensive application hints (DEMS requires applications to disclose transfer sizes, while Meatballs requires applications to distinguish small, latency-sensitive transfers). In contrast, RAVEN requires no hints or application modification. Second, DEMS and Meatballs are user-level implementations. RAVEN's kernel implementation allowed us to add several optimizations that cancel work rendered unnecessary by redundant operation. Third, our study shows substantial benefit available from using three networks. Whereas RAVEN easily scales to any number of networks, DEMS is fundamentally limited to two networks (because each network starts transmitting at one end of the data block and they meet in the middle), and Meatballs computation scales exponentially with the number of networks because it uses joint probability distributions. RAVEN also explores the power of explicit confidence intervals, using them to balance latency and data usage, decide when to transmit redundantly, and test poorly-performing networks for improvement.

ReMP [18] is an MPTCP scheduler that transmits data redundantly over *all* available networks; this will consume too much mobile data. In comparison, RAVEN uses redundancy only when it is most likely to improve user-perceived performance for latency-sensitive applications.

Researchers have noted that tail pocket drop can substantially degrade small, latency-sensitive TCP transfers, and they have proposed various methods for aggressively retransmitting tail data to mitigate this phenomenon [1, 15, 55]. These works can be viewed as potentially sending redundant data over the same network to reduce latency, whereas RAVEN proactively sends redundant data over multiple networks. Network coding [9, 30, 34, 47, 53] offers different tradeoffs between performance and data usage and may be

applicable when the number of possible networks is large. However, at small numbers of networks, redundancy can offer better latency since only one packet need arrive over any network (as opposed to m out of n networks with coding).

Sprout [49] minimizes latency for interactive TCP applications over cellular networks by using probabilistic inference to predict the near-future network link rate based on the packet arrival interval. Sprout targets high-throughput applications and limits bytes transmitted so that less packets are buffered. RAVEN targets smaller flows and employs redundancy over multiple networks to reduce network delays.

While many prior studies have examined wireless network performance [6, 10, 13, 28, 46], our specific focus is on examining network performance for vehicle-to-infrastructure communication and latency-sensitive applications. Thus, many prior studies measure bandwidth, while we measure round-trip time (RTT). Further, the findings in our study often differ significantly from those of prior vehicular studies. Balasubramanian et al. [6] study WiFi availability for offloading cellular traffic to WiFi. Deng et al. [13] and Chen et al. [10] study the goodput of MPTCP over WiFi and LTE. Sommers et al. [46] show high variability and tail latency for wireless networks.

Many prior studies investigate using intermittent open WiFi APs from fast-moving vehicles [8, 14, 39, 40]. In contrast to these prior studies, we find that *commercial* WiFi offerings now provide reasonable coverage in densely-populated areas and support moving vehicles with no modifications, yet *open* WiFi access points without splash screens or other authentication are very hard to find. Use of multiple networks for public bus WiFi has been studied [24, 37].

MPTCP [23] extends TCP to provide multipath communication over multiple underlying networks [43]. Its default scheduler uses the network with the lowest RTT for small transfers and stripes larger transmissions across multiple networks. MPTCP is increasingly being adopted by mobile operating systems, and scheduler design is a very active area of innovation [2, 32]. RAVEN is a new MPTCP scheduler that implements strategic redundancy. Although many vendors are adding support for MPTCP [2, 32], deployment is still not universal. To support unmodified applications and operating systems, we use a TCP-to-MPTCP proxy, similar to the approach used by previous MPTCP research projects [20, 21, 38].

MPTCP performance issues when using heterogeneous mobile networks are well-documented. Several schedulers address aspects of this problem without redundancy. Wischik et al. [50] consider both RTT and congestion to select networks. Yang et al. [52] and Khalili et al. [31] compensate for RTT spikes with adaptive, rate-based scheduling. Paasch et al. [41] mitigate bufferbloat by detecting RTT spikes. While these solutions each ameliorate negative side-effects from unpredictable, high network latency, RAVEN instead attacks the underlying issue by transmitting over additional networks to mask latency spikes in a single network. The limitations of TCP's default RTT estimation algorithm [29] are also well-studied. Several studies [21, 36, 38] show it is vulnerable to RTT fluctuations arising from use of multiple heterogeneous networks. MPTCP has also been used in data centers to aggregate bandwidth of multiple networks [42, 54].

3 STUDY OF V2I CONNECTIVITY

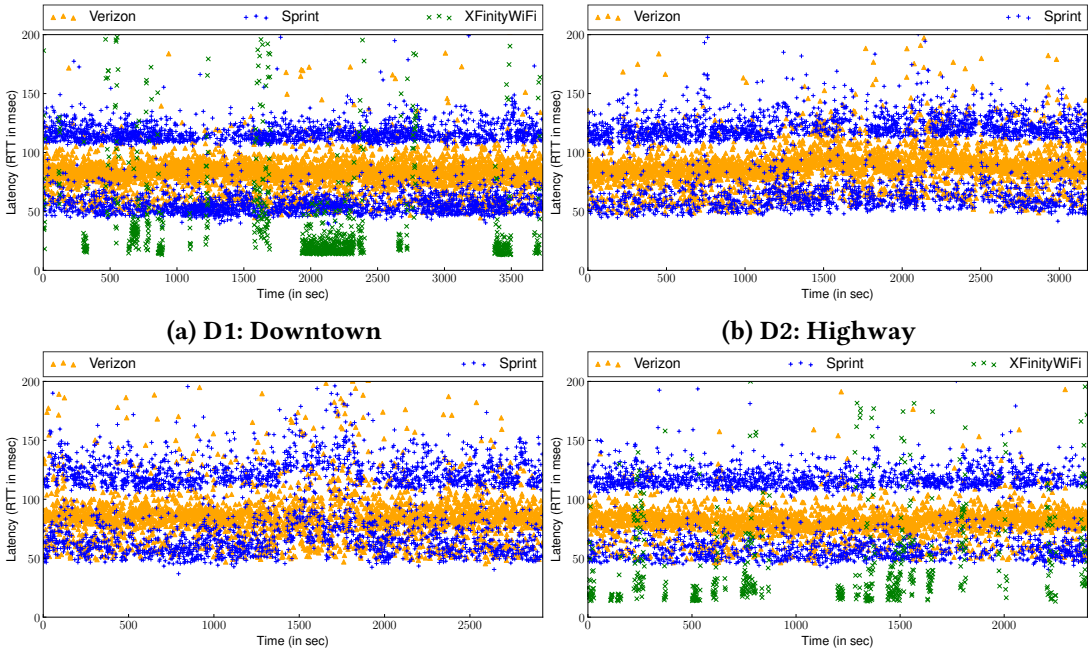
We begin by studying the current state of vehicle-to-infrastructure connectivity. We explore the possibilities enabled by multiple built-in WiFi [17, 48] and 4G LTE [16, 48] network interfaces, as well as tethered interfaces, by simultaneously measuring quality for multiple wireless networks from a moving vehicle in different scenarios. We focus specifically on how connectivity affects the performance of user-facing, interactive applications such as voice commands, GPS, and music streaming. Since these applications require good performance for small data transfers, we measure network RTT rather than bandwidth.

It has been observed that the number of open WiFi access points (APs) not requiring a splash screen or other authentication has been steadily decreasing. In fact, we found almost no availability of such WiFi APs. However, several large ISPs [4, 51] currently provide reasonable WiFi coverage by selling access to the wireless routers they rent to personal and business customers. This is done by turning an individual AP into two APs: one for the customer's use and another that acts as a hotspot. These commercial offerings support WiFi roaming by enabling seamless handoff between APs without breaking TCP connections. Thus, these offerings appear well-suited for vehicular WiFi access.

3.1 Methodology

We built a V2I trace collection tool, called VNperf, that simultaneously measures network quality over multiple wireless networks, as well as vehicle data such as speed and location via the vehicle's OBD2 port and an external GlobalSat BU-353-S4 USB GPS Receiver. Every second, VNperf samples network quality over two cellular networks (Verizon and Sprint) and commercial WiFi (Comcast's XFinityWiFi). The one second interval minimizes external variables that impact RTT measurements (e.g., network congestion and throttling) noted by a previous study [28].

Since we are interested in performance for latency-sensitive applications, we configured VNperf to measure network quality by performing a small RPC over TCP to a dedicated server at our home institution. We also tried hosting the server in various commercial clouds but found that our dedicated server offered the lowest RTTs and jitter.



(a) D1: Downtown

(b) D2: Highway

(c) D3: Rural

(d) D4: Suburban

Figure 1: Scatter plot of RTTs in four driving scenarios. RTTs above 200 ms are not shown.

Trace	Duration	Mean MPH	Verizon			Sprint			XFinityWiFi Hotspot			
			Median	95%	99%	Median	95%	99%	Median	95%	99%	Availability
D1: Downtown	62 mins	5.33	83	99	125	81	127	142	30	1166	4273	30.07%
D2: Highway	53 mins	66.55	87	120	182	115	370	12569	N/A	N/A	N/A	0.00%
D3: Rural	49 mins	35.71	85	174	2933	110	221	8533	N/A	N/A	N/A	0.00%
D4: Suburban	40 mins	7.52	82	98	127	111	135	9019	48	516	2333	25.75%

Table 1: Median and tail RTTs for traces collected in four driving scenarios. RTTs are given in milliseconds. The last column shows how often WiFi was available in each scenario.

Because XFinityWiFi supports seamless WiFi roaming, packets are queued while the interface is not connected to an AP and delivered once an association to a new AP is successful. The XFinityWiFi driver also handles WiFi authentication. Periods of network unavailability can sometimes appear to be intervals that exhibit extremely high latency. Thus, we declare that latencies of over 5 seconds represent periods where WiFi is disconnected. Note that this timeout does not change application behavior; it is merely a mechanism to better classify the data we have collected.

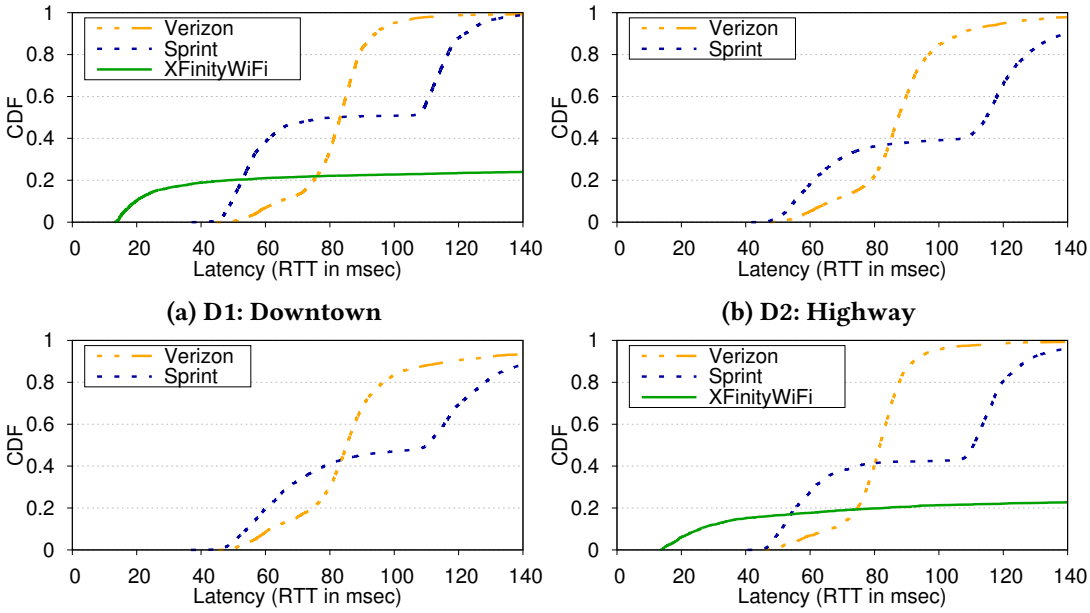
When multiple APs are available, XFinityWiFi selects the one with the highest signal strength. DHCP can incorrectly fail to trigger when the interface has not been associated with any AP for more than 5 minutes, so we corrected DHCP to trigger in this circumstance for routing to new gateways.

We ran VNperf on a Dell XPS 13 Developer laptop with 3.8 GHz CPU and 16GB RAM, running Ubuntu 16.04. The laptop has Verizon Wireless MiFi U620L and Sprint Franklin U772 interfaces. We used a TP-Link T4U USB WiFi interface card to connect to XFinityWiFi.

We report results from four traces, collected in October 2017, each ranging from 40-62 minutes in length; these traces were specifically selected to illustrate behavior in different driving scenarios. Trace D1 was collected driving through the downtown areas of Ann Arbor, MI (population approximately 120,000 and metro area population approximately 350,000). Trace D2 was collected solely on interstate highway driving, primarily but not exclusively through rural areas. Trace D3 was collected on rural roads in sparsely-populated areas. Trace D4 was collected in suburban locations that included neighborhoods, subdivisions, and secondary roads. Additional details are shown in Table 1.

3.2 Results and discussion

Figure 1 shows a scatter plot of RTTs for all three networks, measured once per second. For clarity, only measurements



(a) D1: Downtown **(b) D2: Highway**

(c) D3: Rural **(d) D4: Suburban**

Figure 2: CDF of RTTs in four driving scenarios

for RTTs less than 200 ms are shown. It is immediately apparent that no network offers consistently superior performance. Further, measurements often vary substantially from second to second. This finding aligns with recent studies that have shown high variance in RTTs over cellular networks in high-mobility environments [10, 33]. In contrast, the *average* behavior of the networks is surprisingly stable. We see very few trends where RTT averaged over a longer time window increases or decreases.

Figure 2 plots the RTT CDFs for each network. The RTTs have very different distributions, with Verizon having an approximately normal distribution, Sprint exhibiting bimodal behavior, and XFinityWiFi mixing periods of low RTT with periods of high latency and frequent disconnection.

Table 1 shows the median, 95%, and 99% RTTs measured for each network. Both cellular networks have reasonable tail latency in the downtown trace. Sprint has extremely high 99% tail latency in all other traces, whereas Verizon shows very high 99% tail latency only in the rural trace. XFinityWiFi only was available in the downtown and suburban traces (where it offered coverage 30% and 26% of the time, respectively). Even counting only times when coverage was available, both 95% and 99% tail latencies are very high.

Our WiFi results are a considerable change from a previous 2010 study [6] that reported results from an area corresponding to our downtown trace. That study found average vehicle-to-WiFi access through open APs to be available only 11% of the time. We found almost zero access through open APs, but commercial WiFi access was available 3 times more often than APs in the prior study. WiFi performance is superior to that reported in prior studies. In 2014, Deng et al. found that only 20% of ping RTTs over WiFi were lower than RTTs over LTE [13]. We found that, when WiFi is available, it has lower RTTs than both cellular options 66% of the time in D1 and 58% of the time in D4.

We next examine the predictability of network latency. For each sample, we fist determine which network offers the lowest RTT. We then ask: how often does the network with the lowest RTT in each sample offer the lowest RTT in the next sample (one second later)? The answer is: surprisingly infrequently. Across the four traces, the lowest-latency network at a given time remains the lowest-latency network one second later only 56% of the time. If the vehicle is stopped, predictability increases: 61% of the time, the lowest-latency network for a stopped vehicle remains the best network one second later. For vehicles in motion, we found no correlation between the vehicle's speed and how often the lowest-latency network would remain best one second later.

When the lowest-latency network changes, the median difference between the latency of the new lowest-latency network and that of the previous lowest-latency network is 31 ms. Thus, even frequent active probing of network conditions has limited predictive power.

We also examined *run length*, i.e., the amount of time a given network remains the lowest-latency one before being supplanted by a different network. Across the four traces, the median run length is only two seconds. XFinityWiFi has

longer run lengths than other networks, but its performance also drops off the most sharply when RTTs increase.

We measured the impact of high tail latencies by examining instances in which the RTT for a network exceeded 250 ms. Verizon had high tail latency for 1% of all measurements, Sprint for 2.9% of all measurements, and XFinityWiFi for 3.6% of all measurements when available.

We then considered if high RTTs are correlated by measuring the number of instances in which all connected networks had RTTs over 250 ms. High RTTs for all connected networks occurred in only 0.17% of all samples, much less than for any individual network. This indicates that there exists considerable potential to mask RTT spikes in one network by sending data over another one. There is some correlation between high RTTs: if they were perfectly uncorrelated, we would expect only 0.02% of samples to exhibit high RTTs on all available networks. Over 95% of the correlated high RTT samples occurred in the rural trace.

These findings motivate the work in the next section:

- No network consistently offers the lowest RTTs.
- It appears quite challenging to predict which network will offer the lowest RTT over short time scales.
- At the tail of each CDF, RTTs are very high, which will substantially degrade interactive applications.
- High RTTs are weakly correlated. High RTTs on one network could be masked by using another network.

4 DESIGN AND IMPLEMENTATION

The findings from our study create a dilemma. On one hand, there is ample opportunity to improve interactive performance by using the network that currently offers the lowest RTT. On the other hand, predicting which network will be the best is extremely challenging. Our solution to this dilemma is to transmit latency-sensitive data over multiple networks; the receiver uses the data that arrives first and discards copies that arrive later.

The challenge is that naively transmitting all data redundantly can double (or triple, etc.) mobile data usage. Thus, we must balance interactive latency and extra bytes transmitted by employing *strategic redundancy*, i.e., we should send extra copies only when it does the most good.

We first discuss MPTCP background and then describe how RAVEN modifies MPTCP for strategic redundancy.

4.1 Background: MPTCP

MPTCP multiplexes a single socket connection over multiple low-level TCP *subflows* [23, 43] that traverse different routes. In the mobile setting, each subflow corresponds to a different wireless network interface. For intermittent networks such as WiFi, MPTCP detects that a network has become unavailable via a timeout. The default MPTCP scheduler (called the minRTT scheduler) sends each packet over one network at a time (i.e., it does not transmit redundantly). For each packet, the scheduler selects the subflow with the lowest predicted RTT among all networks that have not yet reached their congestion window. Once data sent over the minimum RTT network reaches the congestion window, the next-lowest RTT network is used, and so on. Thus, small transmissions are usually sent over the network with the smallest predicted RTT. Large transmissions are striped over all available networks. MPTCP links are bidirectional; both endpoints have independent schedulers. MPTCP is flexible; new schedulers can be implemented as a loadable kernel module. This has helped to make MPTCP scheduling an active area of innovation for industry [2, 32] and the research community [22, 35, 36] (e.g., the latest release of iOS has 3 separate MPTCP scheduling policies [2]).

MPTCP calculates a TCP RTT estimate for each subflow based on Jacobson's algorithm [29]; it collects samples from passive measurement of network traffic and calculates an exponentially-weighted moving average over those samples to estimate current RTT. RAVEN also uses per-subflow RTT measurements, but it instead employs the estimation algorithm described in the next two sections.

4.2 Adjusting for stale measurements

TCP weights RTT samples by the order in which they arrive; e.g., the *n*th sample receives the same weight if it was taken 100 ms or 10 seconds in the past. This is reasonable when the network is constantly used, since a highly-weighted observation is unlikely to be old. However, interactive applications transmit infrequently and may send small amounts of data, so predictions based on ordering may assign too much weight to stale samples.

Strategy: to support interactive applications, use elapsed time rather than order to weight RTT samples. RAVEN uses a weighted average of per-subflow RTT samples to predict RTT for each subflow:

$$RTT_{pred} = \frac{\sum_i^n w_i * RTT_i}{\sum_i^n w_i} \quad (1)$$

RAVEN weights the *i*th RTT sample by the time elapsed since the sample was taken:

$$w_i = e^{-(t_{now}-t_i)/\lambda} \quad (2)$$

where t_{now} is the current timestamp, t_i is the time the RTT sample was taken, and λ is a network-specific aging factor. Thus, two consecutive samples will have almost the same weight if they occur within a short time period and very different weights if they are separated by a long time.

RAVEN adaptively sets λ to minimize the root mean squared error (RMSE) of the prediction error for previous predictions made for each network. It temporarily logs each

prediction and the actual RTT measured by TCP. We observe that λ changes infrequently, so RAVEN recalculates values once per day based on the previous day's observations.

4.3 Confidence and multi-networking

RAVEN always sends data over the network with the lowest predicted RTT. When predicting RTT for each subflow, it also estimates certainty in the form of a confidence interval calculated using the distribution of past relative prediction errors. If the RTT confidence interval for any other network overlaps with the confidence interval of the lowest RTT network, RAVEN also transmits the data over that network. This yields strategic redundancy: RAVEN uses multiple networks when it has poor confidence that one will be faster than another.

Our study shows that different networks have different RTT and relative error distributions: we cannot assume a normal distribution or sufficient measurements for the central limit theorem to hold. RAVEN thus calculates confidence intervals from the order statistics of relative prediction errors. For each sample, it computes relative error by dividing the actual RTT measurement by the value it predicted. We tried using both relative error and absolute error and found that relative error yielded slightly better results. Specifically, RAVEN sorts the relative errors and takes, e.g., the 5% and 95% percentile values to calculate the 90% confidence interval over the relative error. Confidence intervals are recalculated once per day when RAVEN recalculates λ.

Strategy: Confidence intervals provide an effective knob for adaptively tuning the tradeoff between performance and data usage. An application or user can choose which confidence interval RAVEN should use and thereby adjust the tradeoff between minimizing RTT and minimizing data usage. Intervals of 100% cause data to be sent over all available networks; intervals of 0% yield no redundancy. Section 5.5 shows how changing the confidence interval affects RTT and data usage; based on this analysis, RAVEN uses 90% confidence intervals by default.

4.4 Replacing active probing

Protocols that send over the best predicted network often underperform when one network exhibits temporary poor performance that leads the protocol to become "stuck" in a mode where the network is no longer used. Poor performance leads to the network not being chosen, which in turn means that the protocol is unable to observe a subsequent performance improvement. A common, but ad-hoc, solution is to actively probe networks that have not been used due to poor predicted performance after an arbitrary timeout.

Strategy: Redundant transmission based on confidence in predictions is a more efficient and elegant method than periodic probing for determining if poorly-performing networks have improved. RAVEN uses its confidence in its predictions to determine when to transmit data redundantly over a network. Intuitively, if our last measurement of a poor-performing network is recent, we have high confidence in that prediction. As time passes without new measurements, our confidence in the prediction decreases. Eventually, the confidence interval for the poor-performing network will overlap with that of the best-predicted network, and RAVEN will transmit data redundantly over the poor-performing network, generating new RTT measurements. This provides a more strategic replacement for active probing.

RAVEN scales each confidence interval by a factor that is a function of the time since the most recent sample was taken. Note that the recency of samples does not affect the expected value of predictions since the exponential weightings of all prior samples decrease by the same relative amount as time passes. The scaling function is calculated from empirical observations once per day. RAVEN first calculates the RMSE for past samples. To calculate the scaling factor for the case where the most recent sample is n seconds old, it first calculates a prediction for each sample omitting any measurements that occurred less than n seconds prior to the prediction. It then calculates the RMSE for those predictions. The relative difference between RMSE is the scaling factor for time n. RAVEN stores the scaling factor for different values of n in a lookup table.

In summary, the bounds of the confidence interval for a RTT prediction are:

$$RTT_{pred} \pm (RTT_{pred}/RelError_{CI}) * Age(T_{now} - T_n) \quad (3)$$

where $RelError_{CI}$ is the confidence interval of the relative error calculated as described in the previous section, $Age()$ is the scaling function, T_{now} is the current time, and T_n is the most recent RTT measurement. Note that $RelError_{CI}$ and $Age()$ are calculated once per day, but the other variables are based on recent RTT measurements.

In contrast to periodic probing, RAVEN will not try poorly-performing networks when it is confident that a current network offers better performance. For example, with a stable low-latency WiFi connection, it makes little sense to see if a poorly-performing cellular network has improved; even the best case latency for the cellular network will not be superior to that of the currently-available WiFi.

4.5 Identifying latency-sensitive traffic

Prior work has required applications to disclose the size of each transfer [20] or which transfers require low latency [26, 27]. This has hindered adoption of redundant transmission.

Strategy: no application modification should be required to use redundant transmission. Rather than require application hints, RAVEN automatically detects traffic

for which low RTT is relatively unimportant and avoids redundancy in those instances. Small transfers are most sensitive to RTT. Large transfers (e.g., downloading a binary or video) are potentially latency-sensitive, but the impact of RTT on the relative completion time of the entire transfer is insignificant; the default MPTCP scheduling policy of striping across multiple networks is likely best [42, 43].

For each MPTCP connection, RAVEN automatically switches between redundant mode (in which it may transmit the same data over multiple networks) and non-redundant mode (which, in order to limit implementation complexity, is the same as the default MPTCP scheduler). Since RAVEN is implemented as a kernel module, it can observe several low-level data structures to attempt to differentiate small and large transfers. We examined several options and determined that the per-subflow TCP queue size, per-flow congestion windows, and main MPTCP socket queue occupancy were the most informative metrics to consider.

The RAVEN scheduler switches from redundant to non-redundant mode if at least $n-1$ out of n active subflows have their congestion window's worth of data in the per-subflow queue. It switches from non-redundant to redundant mode if all per-subflow queues and the main MPTCP socket queue have less than two packets to transmit.

With this algorithm, distinct transfers usually start in redundant mode. Small transfers typically complete before a mode switch happens, and large transfers switch after sending several packets. Our results in Section 5.6 show that this strategy results in very little data being sent redundantly for typical large consumers of data such as Web, video, and application download. Since large transfers naturally contribute the vast majority of bytes sent, most data sent by RAVEN will be transmitted non-redundantly. Yet, RAVEN still improves response time significantly for interactive applications with small transmission sizes.

We have also built an MPTCP proxy so that applications that are not MPTCP-aware can use RAVEN without modification. A client proxy, which we expect to run on the connected car's AP, accepts TCP connections and redirects traffic to either MPTCP-aware servers or a persistent MPTCP connection with a cloud proxy. The cloud proxy, in turn, redirects traffic to unmodified servers.

4.6 Canceling: Avoiding unneeded work

Prior user-level implementations [20, 26] have limited ability to cancel useless work (e.g., sending data over one network that has already been acknowledged by another) because they cannot easily revoke data that has already been given to the kernel. This leads to head-of-line blocking, and may require throttling to improve interactive latency [27].

Strategy: Kernel support can improve performance by proactively canceling work that becomes unnecessary. MPTCP requires each subflow to deliver data in order; it pushes data from the subflow queue to its meta-level queue only when the data is in order according to the subflow sequence number. In contrast, RAVEN also pushes data to the meta-level queue if data that is out of order at the subflow level would be in order at the meta level, and this avoids unnecessary retransmissions. Note that applications still receive data in order.

When sending acknowledgments, RAVEN inspects the MPTCP socket to see if missing packets have been delivered via other subflows; it includes such data when calculating the acknowledged sequence number, which in turn eliminates unneeded subflow-level retransmission. Finally, at the sender, when data is acknowledged by *any* subflow, RAVEN removes it from all per-subflow queues. If the data has not yet been sent by a subflow, this completely eliminates the redundant transmission. Otherwise, this avoids possible retransmissions.

Proactive cancellation is especially useful when RTT spikes are frequent or networks are intermittently available. For instance, when a network is temporarily unavailable, many packets can accumulate in the subflow queue. Eventually, this data is sent and acknowledged over another subflow. RAVEN proactively removes these packets when acknowledgments are received, so the subflow can transmit new data when connectivity returns. A user-level implementation cannot cancel this work, so considerable time and bandwidth are wasted transmitting useless data.

4.7 Implementation

RAVEN is a new scheduler for MPTCP v0.93, implemented as a Linux kernel module consisting of 3823 lines of code. Some aspects of proactive cancellation required a kernel patch (882 LoC); we hope these changes are adopted by the community. Otherwise, RAVEN requires no kernel changes.

The kernel module implements the MPTCP `next_segment()` and `get_subflow()` functions to enforce its scheduling decisions and direct packets to the appropriate subflow queue. It maintains a shadow data structure, the *redundant queue*, that records which subflows are responsible for transmitting which data, along with the per-subflow transmission status. Currently, RAVEN clones packets headers for data in the redundant queue; this avoids conflicting with packet counting for TCP Segment Offloading (TSO). RAVEN uses integer approximation of floating point calculations. For the exponential function, it uses Schraudolph's approximation [45]. Confidence intervals derived from order statistics and aging factors are stored in lookup tables to improve performance.

5 EVALUATION

Our evaluation answers the following questions:

- How much does RAVEN improve performance for interactive applications compared to default MPTCP?
- Are confidence intervals an effective method for balancing performance and data usage?
- How effectively does RAVEN conserve bandwidth by avoiding redundancy for larger transmissions?
- How much do individual elements of RAVEN's design contribute to its performance improvements?

5.1 Methodology

We use both trace-driven emulation and side-by-side comparison in live vehicle experiments. Section 5.4 reports live vehicular experiments; for repeatability, all other experiments use trace-driven emulation of the four network traces from the study in Section 3. We use a leave-one-out method in which we first train RAVEN to determine λ, confidence intervals, and scaling factors (discussed in Section 4) for each network using three traces, and then we evaluate the results of running RAVEN on the remaining network trace. Note that these traces were selected to be dissimilar, so this methodology biases against RAVEN to some degree.

Our emulated setup has a client with multiple interfaces and a server that responds to client requests. All machines run Ubuntu 14.04. The client has 8 3.5 GHz cores and 16 GB RAM; the server has 8 2.8 GHz cores and 8 GB RAM. We use TC to replay collected traces, changing the RTT for each network every second to match the RTTs measured in Section 3. Note that since we measured RTTs every second, our emulation results do not include delays from network power management that occur after multiple seconds of network idleness. We evaluate the impact of power management in live vehicle experiments in Section 5.4. Since we are evaluating latency-sensitive applications that send only a few bytes, bandwidth values do not particularly affect our results. We set bandwidth to 2 Mb/s.

For live vehicular experiments, we run the same application simultaneously on two identical laptops, configured as described in Section 3.1. Although these experiments are inherently unrepeatable, we try to keep characteristics similar to a specific trace by driving in the same geographic area in which we collected the trace. We also synchronize the two applications so that they initiate requests at the same time.

5.2 Applications

We use three applications to evaluate RAVEN: speech recognition, music streaming, and Yelp recommendation. We selected these applications because they are commonly used in a vehicle; they also have a diversity of network access patterns. Each application records the response time of its activities. Unless otherwise noted, the applications are not modified to use RAVEN or provide hints.

Our speech application is a Google speech API client. It streams an utterance to the server using the Google speech API and retrieves the corresponding text when the user finishes (as determined by a period of silence). To eliminate jitter during experiments, we modified the application to connect to a local server at our institution rather than a Google server. Our workload is the sample utterances from the Sphinx speech recognition engine [12]. The client sends raw audio input to the server every 100 ms; this allows recognition to proceed in parallel with speaking. Once the utterance finishes (e.g., after 100 ms of silence), the client asks the server to recognize the utterance. The response time is measured from the time when the client makes this request to the time when it receives the response.

Our music application is the Music Player Daemon (MPD), a popular music streaming server similar to commercial services such as Spotify. We use Sonata, a GUI MPD client, and we run a server at our institution. MPD and Sonata are unmodified TCP applications, so they connect via our MPTCP proxy. Our workload simulates a user scanning through a playlist to select a song. Every 5 seconds, Sonata randomly selects a song form a playlist and starts playing it. Our workload is the Billboard Top 100 songs in August 2017. The response time is measured from the time from when a switch is requested to the time when the new song starts playing. This involves two round trips to the server.

Our recommender application is a Yelp client. Our client uses Yelp's public REST API to query for nearby features and recommendations (e.g., gas stations and grocery stores) using the vehicle's recorded geolocation data. We insert delays between requests drawn from a normal distribution with mean of 20 seconds and standard deviation of 3.7 seconds. We recorded actual requests and responses for this workload. A pseudo-server at our institution provides repeatability by returning the recorded response for each request. The client and server are connected through our MPTCP proxy. The response time is measured from the time when the client makes the request to the time when it receives the response.

5.3 Application response time

We begin by evaluating how much RAVEN improves application response time compared to the default MPTCP scheduler. Figure 3 shows results for speech, music, and Yelp, from left to right. Results for the four traces from our study in Section 3 are shown from top to bottom.

RAVEN uses its default confidence interval of 90%; in Section 5.5, we explore the effect of changing this parameter. Since our applications do not send much data, MPTCP does

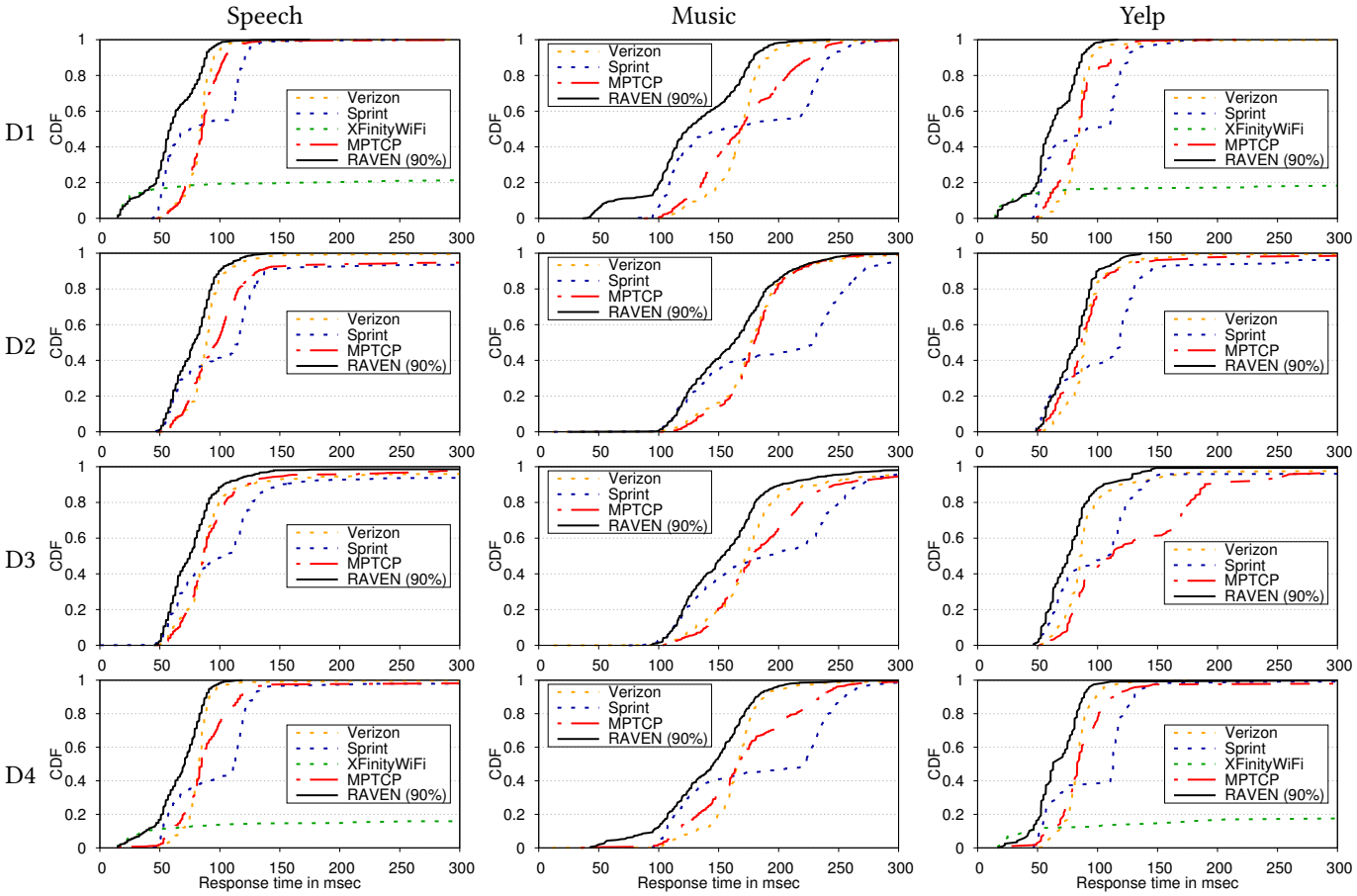


Figure 3: CDFs of application response time for speech, music and Yelp. We compare RAVEN (using its default 90% confidence interval) with the MPTCP default scheduler, as well as with TCP over cellular and WiFi.

App	Trace	Verizon			Sprint			MPTCP			RAVEN		
		Median	95%	99%	Median	95%	99%	Median	95%	99%	Median	95%	99%
Speech	D1	84	101	119	75	125	141	85	112	132	59	93	109
	D2	89	119	173	115	2240	T/O	98	414	T/O	79	111	129
	D3	87	215	T/O	105	820	T/O	87	156	781	75	127	T/O
	D4	83	97	155	113	135	T/O	85	129	T/O	71	93	103
Music	D1	169	201	242	154	255	283	168	236	275	124	188	213
	D2	177	240	341	228	300	1056	180	232	281	163	230	260
	D3	173	289	825	190	296	657	179	315	867	153	244	317
	D4	167	205	312	220	265	436	167	246	339	146	195	245
Yelp	D1	85	101	185	91	131	169	85	125	137	61	93	105
	D2	89	129	175	119	265	10677	87	145	579	83	117	135
	D3	86	161	815	109	149	5565	112	257	713	75	129	149
	D4	83	105	135	113	141	251	83	131	558	63	93	111

Table 2: Median, 95%, and 99% application response times for speech, music, and Yelp. All values are in msecs.

not stripe data and instead sends all data over the subflow with lowest predicted RTT. Thus, its behavior should be the same as other schedulers such as Apple's modified minRTT scheduler [3]. We also show performance using TCP over each network.

RAVEN substantially outperforms default MPTCP for all applications in all scenarios, as shown by the CDF line in each graph being further to the left. For example, in D1:Speech in the top left corner of Figure 3, RAVEN is able to effectively use WiFi when it offers low latency, but MPTCP often

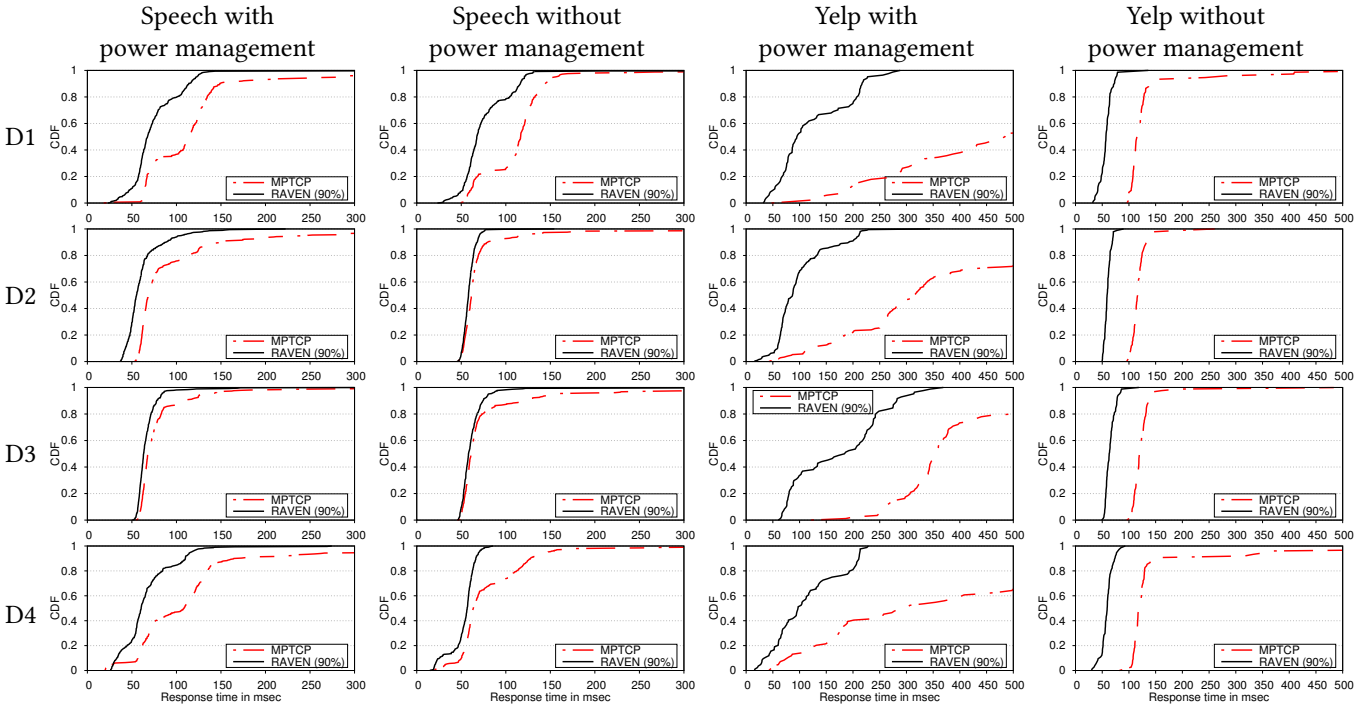


Figure 4: CDFs of application response time for speech and Yelp in live experiments.

does not use the network due to stale, high RTT measurements. At higher latencies, RAVEN has lower response times than either cellular network because it exploits whichever offers better connectivity at the moment. Across all scenarios, RAVEN provides an average speedup in median response time of 26%, 19% and 30% for speech, music, and Yelp, respectively, as compared to MPTCP.

We note that these results seem to track how frequently each application transmits: the music application's network usage is most continuous, whereas Yelp's is the most infrequent. Our belief is that RAVEN's explicit incorporation of when each measurement was taken into its predictions yields greater improvement for applications with less frequent network usage. We explore this further in Section 5.8.

Table 2 provides more detail about tail behavior for the individual cellular networks, MPTCP, and RAVEN (we omit WiFi since it is not frequently available for any entire trace). These results show that both RAVEN and MPTCP substantially improve tail response time compared to the individual networks by using a different network when performance on a given network is poor. However, RAVEN also substantially improves on MPTCP response times. Across the four scenarios, RAVEN provides an average speedup in 95% tail response time of 66%, 20% and 30% for speech, music, and Yelp, respectively, as compared to MPTCP.

For 99% tail response time, the speech application times out for over 1% of the recognitions in two scenarios for MPTCP, but only one for RAVEN. RAVEN provides a speedup of 21% over MPTCP in the other speech scenarios. Across all scenarios, RAVEN provides an average speedup in 99% tail response time of 52% for music and 341% for Yelp as compared to MPTCP. Thus, RAVEN reduces the substantial delays at the tail of the distribution.

5.4 Live vehicular results

We confirmed our emulated results by repeating a subset of the experiments in the vehicle. With two laptops available, we chose to compare RAVEN running with default 90% confidence intervals and MPTCP using its default scheduler. Figure 4 shows results for speech and Yelp in live scenarios of about 1 hour duration similar to our four traces.

Interestingly, RAVEN's relative performance benefit in the live experiments is greater than in the emulated experiments. More variation occurs in a vehicle; e.g., disconnected networks and poorly performing networks are not always distinguished correctly, and the frequency of RTT variation is greater than the one-second variation in our traces. RAVEN is designed to deal with uncertainty, and so it adapts better to this variation. For speech, RAVEN's average speedup in median response time over MPTCP is 46%, ranging between 8% and 80% for the four scenarios. RAVEN's average 95% response time is over two times faster, and its average 99% response time is over three times faster.

MPTCP performs extremely poorly for Yelp in every scenario. RAVEN's average speedup in median response time over MPTCP is 3.4x. There is over an order of magnitude difference in both 95% and 99% response times.

From packet logs, we found the reason for this behavior: the default MPTCP scheduler interacts extremely poorly with network power management. Yelp transmits infrequently, so when a network is selected, it is often in power saving mode. When sending a request to the server, both RAVEN and MPTCP experience a power promotion delay, which is the time for the network to resume sending data when it was previously in a low-power mode. However, MPTCP often incurs a second power promotion delay because the server chooses to send the response over a different network. RAVEN often avoids this cost because it sends the original request over two or more networks; the second network has already exited from its low-power mode when the response is transmitted. For speech, MPTCP will often incur a power promotion delay when switching to a new network; RAVEN often hides this delay by transmitting redundantly during periods where the lowest-latency network is changing.

A second poor interaction is that MPTCP's retransmission mechanism is unaware of power management delays; as a result, it can be either too aggressive or too conservative. The default MPTCP scheduler retransmits data over additional subflows if the subflow-level retransmission timer expires. The timer is conservatively set by multiplying the RTT estimation by 4 (see [44] for details). Thus, when the default MPTCP scheduler selects wrong subflow, it can wait a long time before retransmitting over other subflows (i.e., it is too conservative). However, when a power promotion delay occurs, the default MPTCP scheduler will observe a long delay in receiving an acknowledgment and incorrectly switch to another subflow. In turn, that subflow will likely experience a power promotion delay. Because the default scheduler is not aware of power management delays, it is too aggressive in switching between subflows. Strategies that aggressively retransmit tail data [1, 15, 55] would help with the first problem, but also would likely make the second problem worse unless they explicitly consider power management behavior.

We confirmed these observations by re-running all experiments while injecting a small amount of additional traffic to keep all networks from entering power save mode. The second and last columns of Figure 4 show results for speech and Yelp, respectively. As expected, the speech results are roughly similar (especially given the lack of repeatability in live driving), but the Yelp results show considerable improvement for MPTCP, and, to a lesser extent, for RAVEN. Still, for Yelp, RAVEN provides median application response time more than two times faster than MPTCP, while 95% and 99% tail response times are approximately four times faster.

5.5 Effect of changing confidence intervals

Figure 5 shows the effect on RTT of modifying which confidence intervals RAVEN uses. Figure 6 shows the extra bytes sent over the network (normalized to the total bytes transmitted by the application) for each confidence interval. These and subsequent experiments are run in the emulated environment.

It is important to note that these experiments are designed to generate only latency-sensitive traffic with small data sizes. Thus, these values do not reflect how much redundant data would be sent for a typical workload. As we will see in the next section, RAVEN transmits almost no data redundantly for larger transmissions, and, due to their size, one would expect such transmissions to comprise the vast majority of bytes sent and received by a mobile device.

100% confidence intervals are always redundant: all bytes are sent over all networks. This offers the lowest RTTs, but it also effectively doubles the number of bytes sent when two networks are available and triples it with three networks (actually, the total number of bytes is slightly larger due to retransmission). Shrinking the confidence intervals results in successively fewer bytes sent, but also higher RTTs. While users may have different preferences, we chose 90% confidence intervals as the default setting for RAVEN because it lies at the knee of the curve. This setting has similar performance to always sending redundantly, but it sends many fewer bytes. Lower confidence intervals offer further reduction in bytes sent, but they also noticeably degrade performance.

5.6 Effect of mode switching

Unlike prior systems [20, 26], RAVEN does not require applications to declare or hint about data size in order to decide when to transmit redundantly. Instead, it infers small and large transmissions as described in Section 4.5,

Figure 7 shows how mode switching affects the number of bytes sent redundantly for different transfer sizes. We transmit data from the client to the server 10 times, with a pause of 30 seconds between transmissions, while we replay the D1 trace. We configured RAVEN to always send redundantly in this experiment to isolate the effect of mode switching.

For small transfers (1 KB and 10 KB), all data is sent redundantly. For 50 KB transfers, 41.67% of data is sent redundantly. Yet, only 6.28% of 100 KB transfers and 1.27% of 1 MB transfers are sent redundantly. We do not explicitly measure power usage because, in connected cars, the built-in network interfaces are powered by the vehicle engine, and mobile devices connect to these interfaces via the hotspot. With the MPTCP proxy at the AP, mobile devices transmit only a single copy of the data.

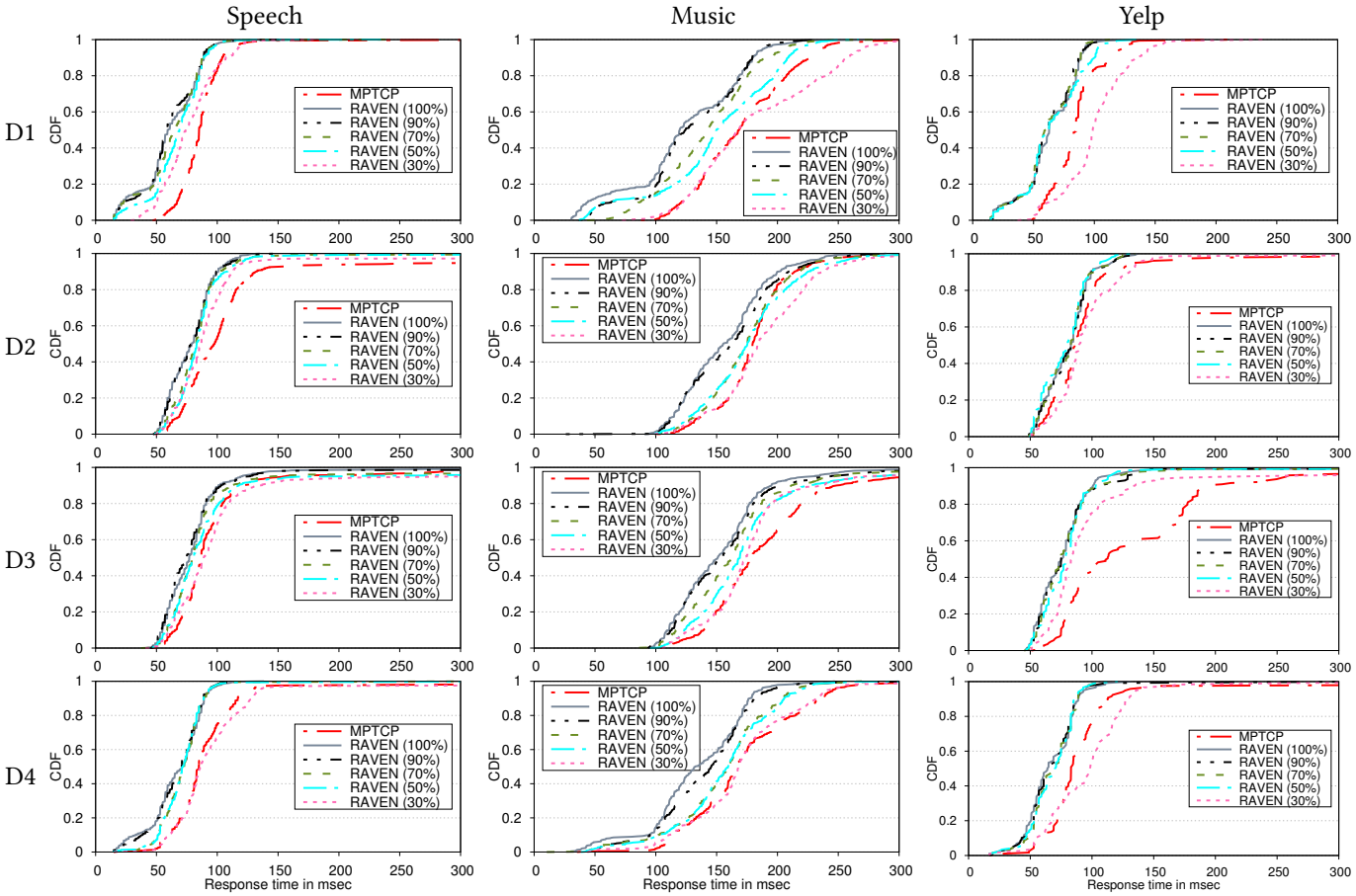


Figure 5: These CDFs show how the choice of confidence interval affects application response time for speech, music and Yelp. We show results with the default MPTCP scheduler for comparison.

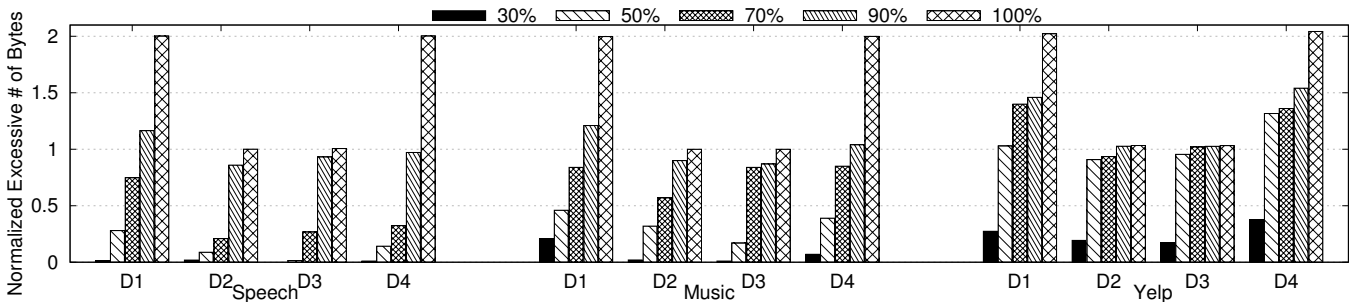


Figure 6: This graph shows how the choice of confidence interval affects extra bytes sent by the speech, music and Yelp applications. A value of 0 indicates no data was transmitted redundantly, 1 indicates twice as many bytes were sent, etc.

We confirmed the micro-benchmark results by examining redundant bytes transmitted for representative mobile workloads while replaying the downtown trace. All applications are unmodified and use our MPTCP proxy.

Web, video, and, and application download are three large consumers of mobile bandwidth. We loaded the home pages of the Alexa top 100 Web sites (as of March 2018) in the Chrome Web browser. Overall, 4.9% bytes were sent redundantly. 2.4% of bytes received and 55.3% of byte sent by the client are transmitted redundantly. The difference reflects the size disparity between typical HTTP requests and responses, and the results show that our heuristics do a good job of distinguishing small and large transmissions. When

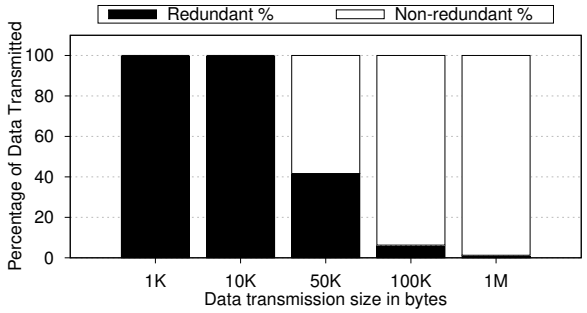


Figure 7: Percentage of data sent redundantly and non-redundantly for different data sizes.

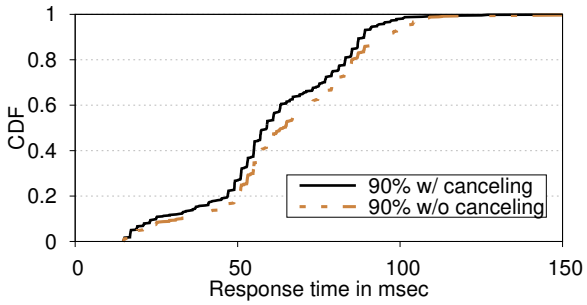


Figure 8: Speech application response time for the downtown scenario with and without proactive cancellation.

we used mplayer to stream a 50 MB mkv video via MPEG-DASH, only 0.03% of bytes were transmitted redundantly. We installed the libreoffice-help-en-us package via apt-get, which sent 2.4 MB of data to the client. Only 0.59% of the bytes were sent redundantly. Thus, for heavy consumers of mobile bandwidth (Web, video, and application download), RAVEN transmits very little data redundantly.

5.7 Effect of proactive cancellation

We next examine how much benefit RAVEN derives from its in-kernel implementation that allows it to cancel useless work (described in Section 4.6). Figure 8 compares RAVEN with and without cancellation enabled for the speech application and the downtown trace (other applications and scenarios exhibit similar behavior). We see a small but consistent improvement across the entire CDF. RAVEN cancellation speeds up both median and 95% tail response time by 9% because queue lengths shorten as work is removed.

5.8 Effect of using scaling for sample age

Finally, we examine the impact of RAVEN's policy of discounting samples by the amount of time that has passed since they were collected (discussed in Section 4.2). We modified RAVEN to not take sample age into account when calculating confidence intervals. Figure 9 compares RAVEN with and without aging for the Yelp application and trace

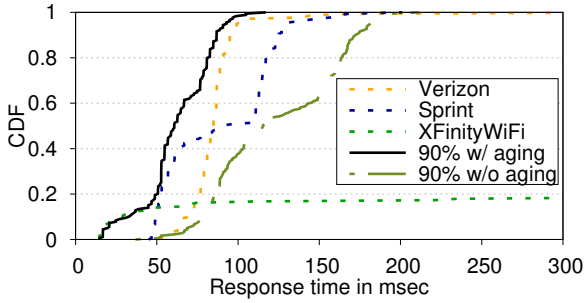


Figure 9: Application response time for Yelp in the downtown scenario with and without scaling for sample age.

D1 (we chose Yelp because it has the least-frequent network transmissions). The aging mechanism appears to be very important, as RAVEN actually underperforms both cellular networks with the mechanism disabled. Without scaling, RAVEN attaches too much importance to stale samples, creating an overconfidence in its predictions that leads to it not employing redundancy when it should.

5.9 Discussion

While our evaluation shows that RAVEN benefits substantially from using more than one cellular network, current practice is for connected devices to either use a single cellular network or switch between networks as with Google's Project Fi [19]. By pointing out the substantial benefit from using multiple cellular networks simultaneously, these results can provide impetus to change practice in the future, at least for vehicle-to-infrastructure communication.

Further optimizations are possible. For instance, if multiple networks offer predictably low RTT that is good enough for a particular application, employing redundancy is unnecessary. However, this optimization requires application-specific knowledge of delay tolerance, which could require application modification. Alternatively, considering external factors such as vehicle motion and geolocation could allow better estimation of confidence, but would add complexity.

6 CONCLUSION

RAVEN is an MPTCP scheduler that reduces interactive latency for vehicular applications through the strategic use of redundant transmission over multiple wireless networks. In emulation, RAVEN speeds up median interactive latency by 19-30% compared to the default MPTCP scheduler. Live vehicular experiments show even greater improvements.

ACKNOWLEDGMENTS

We thank our reviewers and shepherd for their thoughtful comments. This work has been partially supported by the National Science Foundation under grant CNS-1717064.

REFERENCES

[1] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and Hurtig, P. Early retransmit for TCP and stream control transmission protocol (SCTP). IETF RFC 5827, 2010.

[2] Apple opens MultiPath TCP in iOS11. http://www.tessares.net/highlights-from-advances-in-networking-part-1.

[3] A service whereby MultiPath TCP attempts to use the lowest-latency interface. https://developer.apple.com/documentation/foundation/urlsessionconfiguration.multipathservicetype/2875971-interactive.

[4] Understanding AT&T Wi-Fi Hot Spot. http://www.att.com/esupport/article.html.

[5] Audi and AT&T to wirelessly connect 2016 model year vehicles. https://www.audiusa.com/newsroom/news/press-releases/2015/03/audi-and-at-t-to-wirelessly-connect-2016-model-year-vehicles.

[6] Balasubramanian, A., Mahajan, R., and Venkataramani, A. Augmenting mobile 3G using WiFi. In *Proceedings of the 8th International Conference on Mobile Systems, Applications and Services* (San Francisco, CA, June 2010), pp. 209–221.

[7] Mercedes-Benz mbrace. https://www.mbusa.com/vcm/MB/DigitalAssets/pdfmb/mbraceservicebrochures/1527_{M}Bfactsheet$_{0}814_{K}H_{v}2$.pdf.

[8] Bychkovsky, V., Hull, B., Miu, A., Balakrishnan, H., and Madden, S. A measurement study of vehicular internet access using in situ Wi-Fi networks. In *Proceedings of the 12th International Conference on Mobile Computing and Networking* (2006).

[9] Chaporkar, P., and Proutiere, A. Adaptive network coding and scheduling for maximizing throughput in wireless networks. In *Proceedings of the 13th International Conference on Mobile Computing and Networking* (2007).

[10] Chen, Y.-C., Lim, Y.-s., Gibbens, R. J., Nahum, E. M., Khalili, R., and Towsley, D. A measurement-based study of MultiPath TCP performance over wireless networks. In *Proceedings of the 2013 Internet Measurement Conference* (2013).

[11] 2017 Chevrolet Cruze catalog. https://www.chevrolet.com/content/dam/chevrolet/na/us/english/index/shopping-tools/download-catalog/02-pdf/2017-chevrolet-cruze-catalog-r2.pdf.

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

[13] Deng, S., Netravali, R., Sivaraman, A., and Balakrishnan, H. WiFi, LTE, or both? Measuring multi-homed wireless internet performance. In *Proceedings of the 2014 Internet Measurement Conference* (2014).

[14] Deshpande, P., Hou, X., and Das, S. R. Performance comparison of 3G and metro-scale WiFi for vehicular network access. In *Proceedings of the 2010 ACM Conference on Computer Communications* (2010).

[15] Flach, T., Dukkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., and Govindan, R. Reducing web latency: the virtue of gentle aggression. In *Proceedings of the 2013 ACM Conference on Computer Communications* (2013).

[16] Ford SYNC Connect Overview. https://owner.ford.com/how-tos/sync-technology/sync-3/sync-connect/sync-connect-overview.html.

[17] How to install Ford SYNC 3 updates with Wi-Fi. https://owner.ford.com/how-tos/sync-technology/sync-3/software-updates/how-to-install-sync-3-updates-with-wi-fi.html?sync=sync-3.

[18] Frommgen, A., Erbshäusser, T., Buchmann, A., Zimmermann, T., and Wehrle, K. ReMP TCP: Low latency multipath TCP. In *IEEE International Conference on Communications (ICC)* (2016).

[19] Google's Project Fi service turns multiple phone networks into one. https://www.engadget.com/2015/04/22/google-project-fi/.

[20] Guo, Y. E., Nikravesh, A., Mao, Z. M., Qian, F., and Sen, S. Accelerating multipath transport through balanced subflow completion. In *Proceedings of the 23rd International Conference on Mobile Computing and Networking* (2017).

[21] Han, B., Qian, F., Hao, S., and Ji, L. An anatomy of mobile web performance over Multipath TCP. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies(CoNEXT)* (2015), ACM.

[22] Han, B., Qian, F., Ji, L., and Gopalakrishnan, V. MP-DASH: Adaptive video streaming over preference-aware multipath. In *Proceedings of the 12th ACM Conference on Emerging Networking Experiments and Technologies(CoNEXT)* (2016), ACM.

[23] Handley, M., Bonaventure, O., Raiciu, C., and Ford, A. TCP extensions for multipath operation with multiple addresses. IETF RFC 6824, 2013.

[24] Hare, J., Hartung, L., and Banerjee, S. Beyond deployments and testbeds: experiences with public usage on vehicular WiFi hotspots. In *Proceedings of the 10th International Conference on Mobile Systems, Applications and Services* (2012).

[25] Higgins, B. D. *Balancing Interactive Performance and Budgeted Resources in Mobile Computing*. PhD thesis, Computer Science and Engineering, University of Michigan, 2014.

[26] Higgins, B. D., Lee, K., Flinn, J., Giuli, T. J., Noble, B. D., and Peplin, C. The future is cloudy: Reflecting prediction error in mobile applications. In *Proceedings of the 6th International Conference on Mobile Computing, Applications, and Services (MobiCASE)* (November 2014).

[27] Higgins, B. D., Reda, A., Alperovich, T., Flinn, J., Giuli, T. J., Noble, B., and Watson, D. Intentional networking: Opportunistic exploitation of mobile network diversity. In *Proceedings of the 16th International Conference on Mobile Computing and Networking* (Chicago, IL, September 2010), pp. 73–84.

[28] Huang, J., Qian, F., Guo, Y., Zhou, Y., Xu, Q., Mao, Z. M., Sen, S., and Spatscheck, O. An in-depth study of LTE: Effect of network protocol and application behavior on performance. In *Proceedings of the 2013 ACM Conference on Computer Communications* (2013).

[29] Jacobson, V. Congestion avoidance and control. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM)* (Stanford, CA, August 1988), pp. 314–329.

[30] Katti, S., Rahul, H., Hu, W., Katabi, D., Médard, M., and Crowcroft, J. XORs in the air: Practical wireless network coding. In *Proceedings of the 2006 ACM Conference on Computer Communications* (2006).

[31] Khalili, R., Gast, N., Popovic, M., Upadhyay, U., and Le Boudec, J.-Y. 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)* (2012).

[32] KT's GiGA LTE. https://www.ietf.org/proceedings/93/slides/slides-93-mptcp-3.pdf.

[33] Li, L., Xu, K., Wang, D., Peng, C., Xiao, Q., and Mijumbi, R. A measurement study on TCP behaviors in HSPA+ networks on high-speed rails. In *Proceedings of the 34th Annual IEEE International Conference on Computer Communications* (2015).

[34] Li, S.-Y. R., Yeung, R. W., and Cai, N. Linear network coding. *IEEE Transactions on Information Theory 49*, 2 (2003), 371–381.

[35] Lim, Y.-s., Chen, Y.-C., Nahum, E. M., Towsley, D., Gibbens, R. J., and Cecchet, E. Design, implementation, and evaluation of energy-aware multi-path TCP. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies(CoNEXT)* (2015), ACM.

[36] Lim, Y.-s., Nahum, E. M., Towsley, D., and Gibbens, R. J. ECF: An MPTCP path scheduler to manage heterogeneous paths. In *Proceedings of the 13th ACM International on Conference on emerging Networking Experiments and Technologies (CoNEXT)* (2017).

[37] Mahajan, R., Padhye, J., Agarwal, S., and Zill, B. High performance vehicular connectivity with opportunistic erasure coding. In *Proceedings of the 2012 USENIX Conference on USENIX Annual Technical*

Conference (2012).

[38] Nikravesh, A., Guo, Y., Qian, F., Mao, Z. M., and Sen, S. An in-depth understanding of Multipath TCP on mobile devices: Measurement and system design. In *Proceedings of the 22nd International Conference on Mobile Computing and Networking* (2016).

[39] Ott, J., and Kutscher, D. Drive-thru Internet: IEEE 802.11b for automobile users. In *Proceedings of the 23rd Annual IEEE International Conference on Computer Communications* (2004).

[40] Ott, J., and Kutscher, D. A disconnection-tolerant transport for Drive-thru Internet environments. In *Proceedings of the 24th Annual IEEE International Conference on Computer Communications* (2005).

[41] Paasch, C., Ferlin, S., Alay, O., and Bonaventure, O. Experimental evaluation of Multipath TCP schedulers. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop* (2014).

[42] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., and Handley, M. Improving datacenter performance and robustness with Multipath TCP. In *Proceedings of the 2011 ACM Conference on Computer Communications* (2011).

[43] Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., and Handley, M. How hard can it be? Designing and implementing a deployable Multipath TCP. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation* (2012).

[44] Sargent, M., Paxson, V., Allman, M., and Chu, J. Computing TCP's retransmission timer. IETF RFC 6298, 2011.

[45] Schraudolph, N. N. A fast, compact approximation of the exponential function. *Neural Computation 11*, 4 (1999), 853–862.

[46] Sommers, J., and Barford, P. Cell vs. WiFi: On the performance of metro area mobile connections. In *Proceedings of the 2012 Internet Measurement Conference* (2012).

[47] Sundararajan, J. K., Shah, D., Médard, M., Mitzenmacher, M., and Barros, J. Network coding meets TCP. In *Proceedings of the 28th Annual IEEE International Conference on Computer Communications* (2009).

[48] Tesla Model S software release notes v5.8. https://www.tesla.com/sites/default/files/blog$_{a}$ttachments/software$_{u}$pdate$_{5}$.8.pdf.

[49] Winstein, K., Sivaraman, A., Balakrishnan, H., et al. Stochastic forecasts achieve high throughput and low delay over cellular networks. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation* (2013).

[50] Wischik, D., Raiciu, C., Greenhalgh, A., and Handley, M. Design, implementation and evaluation of congestion control for Multipath TCP. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation* (2011).

[51] XFinityWiFi Hotspots overview. http://www.xfinity.com/support/internet/about-xfinity-wifi-internet.

[52] Yang, F., Amer, P., and Ekiz, N. A scheduler for Multipath TCP. In *22nd International Conference on Computer Communications and Networks (ICCCN)* (2013).

[53] Zhang, X., and Li, B. Optimized multipath network coding in lossy wireless networks. In *Proceedings of the 28th International Conference on Distributed Computing Systems(ICDCS)* (2008).

[54] Zhou, J., Tewari, M., Zhu, M., Kabbani, A., Poutievski, L., Singh, A., and Vahdat, A. WCMP: Weighted cost multipathing for improved fairness in data centers. In *Proceedings of the 9th ACM European Conference on Computer Systems* (2014).

[55] Zhou, J., Wu, Q., Li, Z., Uhlig, S., Steenkiste, P., Chen, J., and Xie, G. Demystifying and mitigating tcp stalls at the server side. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)* (2015).