High Performance Vehicular Connectivity with Opportunistic Erasure Coding

Ratul Mahajan Jitendra Padhye Sharad Agarwal Brian Zill

Microsoft Research

Abstract — Motivated by poor network connectivity from moving vehicles, we develop a new loss recovery method called opportunistic erasure coding (OEC). Unlike existing erasure coding methods, which are oblivious to the level of spare capacity along a path, OEC transmits coded packets only during instantaneous openings in a path's spare capacity. This transmission strategy ensures that coded packets provide as much protection as the level of spare capacity allows, without delaying or stealing capacity from data packets. OEC uses a novel encoding that greedily maximizes the amount of new data recovered by the receiver with each coded packet. We design and implement a system called PluriBus that uses OEC in the vehicular context. We deploy it on two buses for two months and show that PluriBus reduces the mean flow completion time by a factor of 4 for a realistic workload. We also show that OEC outperforms existing loss recovery methods in a range of lossy environments.

1. Introduction

Internet access on-board buses, trains, and ferries is increasingly common. Many public transit agencies provide this access today [48, 46, 14]. It is seen as an added amenity that boosts ridership, even in the age of the 3G smart phones [28, 35]. Corporations also provide such access on their commute vehicles [45, 47]. For instance, over one-quarter of Google's employees in the Bay Area use such connected buses [45]. By all accounts, riders greatly value this connectivity.

Our work is motivated by our experiences of poor performance of such connectivity and those of other users [43, 44]. Experiences with its commuter service led Microsoft to pre-emptively warn the riders that "there can be lapses in the backhaul coverage or system congestion" and suggest "cancel a failed download and re-try in approximately 5 minutes." Despite increasing popularity and a unique operating environment, the research community has paid little attention to how to best engineer these networks.

Figure 1 shows the typical way to enable Internet access on buses today. Riders use WiFi to connect to a device on the bus (e.g., [13]), which we call *VanProxy*. The device provides Internet access using one or more links based on wide-area wireless network (WWAN) technologies such as EVDO or HSDPA. The key to application performance in this setup is the quality of connectivity provided by the WWAN links.

To understand this connectivity, we conducted detailed measurements of multiple technologies. Consistent with earlier findings [34, 18], we find that WWAN paths offer poor service from moving vehicles. They have high delays and frequently drop packets. Occasionally, they suffer "blackout" periods with very high loss rates. Thus, poor application performance is only to be expected.

To improve user experience, we must mask losses from applications and offer them a more reliable communication channel. While numerous loss recovery schemes exist, we find that they fall short in this environment. Existing schemes can be categorized as either retransmission-based (ARQ) and erasure coding based. Retransmission-based schemes perform poorly because of the high delay in receiving feedback from the receiver.

Proactive erasure coding is more appropriate in high-delay scenarios but existing schemes (e.g., Maelstrom [2], CORE [23], LT-TCP [41]) have a basic limitation: they are oblivious to spare capacity along a path. For a given set of data packets, the number of erasure coded packets sent does not depend on the available capacity of the path. If this coding redundancy is low, existing schemes do not provide sufficient protection even though there may be spare capacity in the system. If it is high, valuable capacity is stolen from data packets.

In this paper, we explore a new point in the design space of erasure coding and evaluate it in the vehicular context. Our method, called opportunistic erasure coding (OEC), dynamically adjusts coding redundancy to match the spare capacity of the path at short time scales. Matching at short time scales is important because, as we show, the traffic is highly bursty. Matching coding overhead to average spare capacity is not sufficient, as it can lead to significant short-term mismatches.

To match coding redundancy to spare capacity at short-time scales, OEC sends coded packets opportunistically, based on an estimate of bottleneck queue length. Coded packets are transmitted as soon as and only when the queue is deemed empty. Thus, OEC does not delay data packets and yet provides as much protection as available capacity allows.

To make the best use of such opportunistic transmissions, we construct coded packets using a greedy encoding that maximizes the expected number of data packets recovered using each coded packet. Our encoding can be considered a generalization of Growth codes [19] that

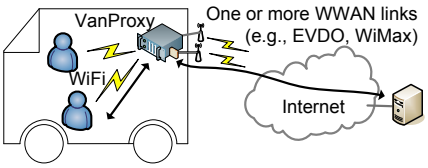


Figure 1: A common way of providing Internet access on board vehicles today.

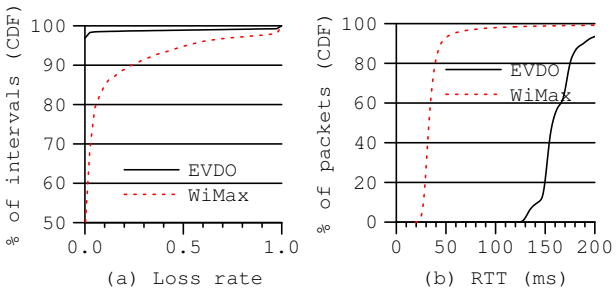


Figure 2: (a) Loss rate for paths to the buses. The y-axis begins at 50%. (b) Path round trip times (RTT).

explicitly accounts for the information available at the receiver while constructing the next coded packet. In contrast, conventional erasure coding methods such as Reed-Solomon [33] or LT [24] aim to minimize the number of packets needed at the receiver to recover all data. But when the required threshold of packets are not received, they recover very little data [36]. In a highly dynamic environment, it is difficult to guarantee that the required threshold number of packets will be sent, let alone received. Thus, these codes are not suitable for our use.

The combination of opportunistic transmissions and our encoding means that OEC greedily maximizes goodput with each packet transmission. OEC retains this property even when data is spread across multiple paths with disparate loss and delay. We accomplish this through delay-based path selection [9]: each data packet is sent over the path that is estimated to deliver it first.

We design and implement a system called PluriBus that applies OEC to the vehicular context. We deploy PluriBus on two buses for two months. Each bus is equipped with two WWAN links, one EVDO and one WiMax.

Our evaluation using this deployment as well as controlled experiments show that PluriBus is highly effective over a range of network conditions. In our deployment, it reduces the mean flow completion time for a realistic workload by a factor of 4 compared to the current practice of not using any loss recovery method (beyond end-to-end TCP). Compared to using retransmissions or capacity-oblivious erasure coding, OEC reduces the mean flow completion time by at least a factor of 1.4.

2. Target environment

We begin by characterizing the network and workload in our target environment. To understand the connectivity provided by WWAN links to moving vehicles, we use two buses that ply around the Microsoft campus from 7 AM to 7 PM on weekdays. Each bus has a computer equipped with an 1xEVDO (Rev. A) NIC on Sprint's network and a (draft standard) WiMax modem on Clearwire's network.

2.1 Network path characteristics

We characterize path quality by sending packets between the bus and a computer connected to the wired Internet. A packet is sent along each provider in each direction every 100 ms. Our analysis is based on two weeks of data. Figure 2(a) shows the CDF of loss rates, averaged over 5 second intervals, from the wired host to the buses. The reverse direction has similar behavior. We see that both paths are lossy. WiMax is worse—half of the intervals experience some packet loss, and 15% suffer over 10% loss. For EVDO, 97% of the intervals see no loss, but 2% suffer over 10% loss. Note that these losses are measured at the IP layer and represent cases where low-level reliability mechanisms (e.g., link-layer FEC) have failed. They will be experienced by TCP connections traversing these links.

Our observations are consistent with other WWAN studies [22, 18]. These studies also find that most losses are not due to congestion but occur due to problems inherent in wireless transmissions. Wireless collisions with other WWAN clients are not an issue; unlike WiFi, the WWAN MAC protocols prevent such collisions.

Figure 2 shows the CDF of RTT for each provider. Both providers have high delay. For EVDO, the median RTT is 150 ms. For WiMax, it is roughly 40 ms. Even this lower of the two delays is surprising given that the path end points are in the same city. We find using *traceroute* that nearly all of this delay is inside the wireless carriers' networks; in fact, a significant fraction is to the first IP-level hop from the wireless client. Details of this experiment are in our extended report [27]. This high delay has implications for how losses can be masked.

We also see that the two links have disparate loss and delay characteristics. This disparity creates significant complications if we want to use them simultaneously. For instance, the factor of three difference in the path RTTs implies that a scheme like round robin will perform poorly. It will significantly reorder packets and unnecessarily delay some packets even though a shorter path exists. Sending all data on the shorter path may not be possible due to capacity constraints, and as is the case in our setup, the lower delay path may have more loss. Using multiple links from the same provider can alleviate the disparity in path properties, but it also reduces reliability because of correlated losses [34].

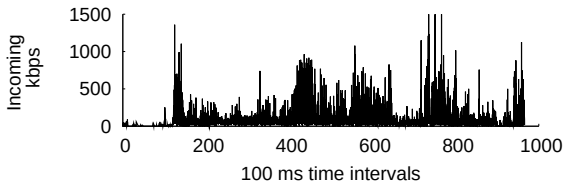


Figure 3: Traffic from Internet to clients

2.2 Workload characteristics

To understand the workload in our target environment, we collect traffic logs from two corporate commuter buses. These buses are different from the ones in our testbed. They have the setup shown in Figure 1, with one Sprint-based 1xEVDO NIC. We sniffed the intra-bus WiFi network for two weeks to capture packets that are sent and received by the riders.

We find that this workload is dominated by short TCP flows [27] which are highly vulnerable to packet loss. It is also highly bursty, as illustrated by an example 100-second period in Figure 3. The average load over the entire trace is quite low, only 86 Kbps, which implies that there is ample leftover capacity along these paths on average. However, short-term load is often above 1 Mbps, which indicates short-term capacity limitations given the throughput that EVDO can achieve. This burstiness means that short-term spare capacity is bursty as well and can differ substantially from the long-term average.

2.3 Discussion

In summary, we characterize our network environment as follows: (i) paths are lossy; (ii) paths have high delays such that timely feedback on packet loss is not available; (iii) the workload is highly bursty such that while there is plenty of capacity available on average, the utilization can approach 100% at short time scales; and (iv) if multiple paths are used, different paths may have different loss and delay properties.

Improving application performance in this environment requires that we reduce packet loss experienced by applications. We could urge the wireless carriers to further improve the lower-layer reliability mechanisms and handoff protocols. This is a long-term proposition that requires significant investment and does not help today's users. We thus build a high performance system on top of existing unreliable connectivity. Improvements to lower-layer connectivity are complementary to our approach.

3. Limitations of existing options

There has been much work on improving application performance over lossy paths. The set of proposed schemes can be broadly classified as those that use retransmissions and those that use erasure coding.

3.1 Retransmission based methods

One way to combat packet loss is by having the sender retransmit lost packets based on feedback from the receiver. This method is used, for instance, in TCP and its variants.[1] However, retransmission based recovery is too slow in settings with high delays. Loss recovery takes at least 1.5 times the round trip time (RTT). We show later that this delay leads to poor performance.

Some methods reduce this delay by isolating the lossy segment of the path such that retransmissions can be performed more quickly. Such retransmissions can be done using support from the wireless base stations (e.g., Snoop TCP [1], Flow Aggregator [7], Ack Regulator [8]) or using additional proxies (e.g., Split TCP [21]). We cannot use these techniques because we do not have access to the wireless carrier's infrastructure. As long as we are sitting outside this infrastructure, the lossy segment of the path will have high delay as well. Thus, the performance of techniques like Split TCP is similar to using an end-to-end TCP connection. We have verified this behavior via experiments in our setting.

Tsao and Sivakumar propose to retransmit lost TCP segments on one interface via another [42]. Their proposal does not use coding is limited to mobile phones, requiring significant changes to TCP stacks on both ends.

3.2 Erasure coding methods

In environments with high delay, erasure coding is a better fit [2]. Erasure coded packets are sent proactively to guard against losses. However, existing erasure coding methods are capacity-oblivious. Systems such as Maelstrom [2] or CORE [23] transmit a fixed number of coded packets for a given set of data packets. If their coding overhead is too low, they do not provide sufficient protection even though there may be excess capacity in the system. If it is too high, they hurt goodput by stealing capacity from data packets.

Even adaptive systems such as MPLOT [38] or LT-TCP [41] adapt to path loss rate and not to spare capacity. Based on the expected loss rate, they add enough redundancy such that data is delivered with a high probability. But because losses as well as spare capacity are bursty, at any given time these systems too can provide insufficient protection even though spare capacity exists or hurt goodput when there is capacity pressure (§6.2).

We argue that the most effective way to protect against losses is to use *all* spare capacity [26]. However, the bursty nature of traffic and thus of spare capacity implies that it is not sufficient to match the level of redundancy to average spare capacity. The short-term mismatch can be significant, leading either to insufficient protection or to

[1] Some experimental TCP variants do not reduce sending rate for non-congestion losses, but their reliability mechanism is still based on retransmissions.

overload. Hence, we develop opportunistic erasure coding (OEC) that provides as much protection as the available capacity allows at short time scales without hurting data packets.

Rateless erasure codes such as LT [24] can generate an unlimited stream of coded packets, but they are not complete erasure coding systems. One must still decide when and how many coded packets to transmit. We also point out later why these codes are inappropriate if one wanted to opportunistically use leftover spare capacity.

4. Opportunistic erasure coding

OEC is meant for lossy environments in which timely feedback on which packets were lost is not available to the sender. Our current design assumes that all packets are equally important; extending OEC to unequal protection (e.g., for video codecs) is a subject of future work.

An ideal goal for an erasure coding scheme in a setting with short TCP flows is to minimize connection completion time, as that directly impact user experience. However, no practical method can achieve this goal when traffic, losses, and path capacity are highly dynamic. We thus modulate our goal to be greedy goodput maximization: each transmission should maximize the amount of new data at the receiver. We show later that this strategy leads to significant reduction in connection completion time.

OEC requires an estimate of the usable capacity of the path. This capacity is not necessarily the physical capacity of the path but is what the OEC traffic can use along the path without hurting others. It may be either be configured or estimated. In PluriBus, we estimate it using a technique based on recent bandwidth measurements tools (§5.2). It can also be estimated using other techniques, e.g., those similar to TCP Vegas [4].

We first describe how OEC functions in the case of one path between the sender and receiver and then describe the generalization to multiple paths.

4.1 Single path case

Consider the following idealized protocol, of which OEC is a practical instantiation. This protocol views network path as a communication channel whose bottleneck capacity matches the given usable capacity. It transmits new data packets as soon as they are generated by the application. If new data is being generated at a rate faster than the channel capacity, it will be queued at the bottleneck. The protocol transmits an erasure coded packet as soon as and only if the packet will find an empty queue. In this way, it uses for coded packets any and all leftover capacity at short time scales. Finally, it encodes each coded packet in a way that maximizes the amount of new data recovered at the receiver.

We argue that this protocol greedily maximizes goodput. By using all capacity, it achieves the highest possible throughput (i.e., rate of unique + non-unique data). Whether it maximizes goodput depends thus on the order and contents of the packets sent. In terms of order, strictly prioritizing data packets, as we do above, is optimal. The reception of a data packet provides one new data packet to the receiver and of a coded packet provides less than one on average [24]. Some coded packets may yield more than one but the average yield will be less than one. Finally, each coded packet is constructed to maximize the amount of new data recovered by the receiver. Thus, in combination, no other protocol can achieve higher goodput at each step, without future knowledge.

To implement this protocol, we need two capabilities. First, we need a method to estimate when the bottleneck queue, which is not necessarily local, will be empty. The knowledge of path capacity and past data and coded transmissions lets us estimate the number of OEC packets at the bottleneck at any given time. We can then transmit coded packets such that they reach the bottleneck when there are no other packets. This way, coded packets always defer to data packets and delay them by at most one packet, while providing as much protection as the amount of spare capacity allows.

In the extreme case where data packets are generated at a rate faster than capacity for an extended period, OEC sends no coded packets. This behavior is optimal with respect to our goal of greedy goodput maximization. However, a certain fraction of coded packets can be easily added to the stream if some minimum protection against loss is desirable at all times. Note, however, that if the end hosts are using end-to-end congestion control, the data rate is unlikely to stay higher than capacity for an extended period if the loss rate experienced is high.

The second capability is an encoding technique that maximizes the amount of data with each coded packet. Conventional erasure codes, whether rateless (e.g., LT [24]) or not (e.g., Reed-Solomon [33]), cannot be used for this purpose. These codes are designed for efficient recovery. They seek to minimize the number of packets needed at the receiver to recover all data. But they recover very little if fewer than the needed threshold number of packets are received [36]. In our setting, with bursty data arrivals, we cannot even predict how many coded packets can be transmitted, let alone how many can be received.

We develop an encoding that greedily maximizes the expected amount of new data recovered by each coded packet. It does that by explicitly accounting for what information might already be present at the receiver. Conventional codes do not consider receiver state at intermediate points in time. We describe our encoding next.

4.2 Greedy encoding

Consider a point in time when the sender has sent a window W of data packets and some coded packets con-

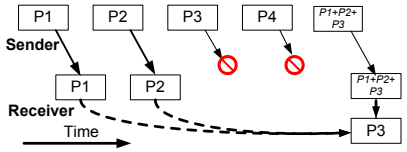


Figure 4: Illustration of our coding system. Data packets P3 and P4 are dropped in transit. The receiver is able to recover P3 after it receives coded packet P1+P2+P3.

structed per our scheme. The sender has not received any feedback from the receiver about the packets in W, and so it is unaware of the exact fate of each data packet. (In §5.1, we describe how W is updated as the sender sends data and coded packets and receives feedback from the receiver.) The receiver has a given data packet either if it received the original transmission of the data packet or if it recovered the packet using a subsequent coded transmission after the original transmission was lost. An example is shown in Figure 4.

Now, the sender has an opportunity to send one more coded packet. Our aim is to construct a coded packet that is "most useful" to the receiver. To keep encoding and decoding operations simple, like several other erasure codes (e.g., LT [24], Growth [19], Maelstrom [2]), we construct coded packets by XOR-ing data packets. Further, to keep analysis simple, we assume that the receiver discards coded packets that cannot be immediately decoded using the data packets that it has received or recovered in the past. The implementation can buffer such packets and decode them later, but we found that this optimization brings little additional gain in our environment.

Thus, to construct a coded packet, the sender must decide which data packets to XOR such that the resulting coded packet is likely to yield a previously missing data packet when decoded using data packets already at the receiver. From a sender's viewpoint, the optimal solution to this problem depends on the probability of each data packet being available at the receiver. This probability is in general different for different packets. It depends on the path loss process, and the precise sequence of coded packets transmitted thus far. It is computationally hard for the sender i) to track these probabilities, as the number of possible combinations grows exponentially; and ii) optimally encode based on individual probabilities.

For tractability, the sender makes a simplifying assumption that each data packet has the same probability, r, of being present at the receiver. We explored heuristics that account for different per-packet probabilities, but found that the performance hit of this assumption is negligible for loss rates and encoding windows sizes that occur in practice. In §5.1, we describe how the sender can estimate r based on path loss rate and past transmissions.

With this assumption, the problem of determining the composition of an ideal coded packet boils downs to how many packets should be XOR'd [10]. Suppose the sender XORs c data packets. The probability that this coded packet will yield a previously missing data packet at the receiver equals the probability that exactly one out of the c packets is missing. Thus, the expected yield of this coded packet is:

$$Y(c) = c \cdot (1-r) \cdot r^{c-1} \quad (1)$$

To maximize the expected yield, we have:

$$c = -1/ln(r)$$

This result can be intuitively explained. Observe that c is inversely proportional to r. If the fraction of data packets at the receiver is low, we construct a coded packet by XOR-ing few data packets. For instance, if most packets are missing, the best strategy is to encode only one packet (i.e., send a duplicate); coding even two is sub-optimal as the chance of both being absent and nothing being recovered is high. Conversely, if a higher fraction of packets are present at the receiver, encoding more packets recovers missing data faster.

Thus, the sender randomly selects $\max(1, \lfloor \frac{-1}{ln(r)} \rfloor)$ data packets to XOR. We round down because including fewer data packets is safer than including more.

4.3 Generalizing to multiple paths

OEC can be generalized to the case where transmissions are spread over multiple paths with disparate loss and delay characteristics, while maintaining the greedy goodput maximization property.

In the presence of multiple paths, we send each data packet along the path that currently offers the least delay [9], which is judged using estimates of queue length and propagation delay. We continue to send traffic along the fastest path until queuing increases its delay to the level of the next fastest path, and so on. This method naturally generalizes striping mechanisms such as round robin to the case of paths with different delays and capacities. It minimizes average packet delay, and makes reordering less likely. Variations and mis-estimations of path delay can still lead to some reordering, which we handle using a small sequencing buffer. Coded packets are sent as before, when spare transmission opportunities open up along any path.

We argue that the use of delay-based striping in OEC greedily maximizes goodput. Let there be k paths between the two proxies and let the capacity, delay, and loss rate of path i be c_i, d_i and p_i respectively. The least delay selection policy then creates a virtual path whose capacity is equal to the sum of the individual capacities, delay is less than the maximum individual delay, and the loss rate is the weighted average of individual loss rates [9]. That is, $C = \sum_{i=1}^{k} c_i$, $D \leq \max_{i=1}^{k} d_i$, and $P \leq \sum_{i=1}^{k} \frac{p_i * c_i}{\sum_{i=1}^{k} c_i}$. This combination is optimal with

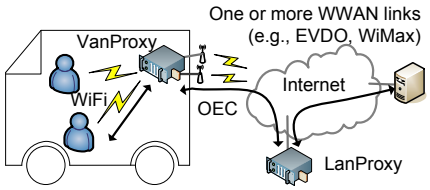


Figure 5: The architecture of PluriBus. It uses OEC between the two proxies and can combine multiple WWAN links for additional capacity.

respect to goodput [9]. OEC on top of this virtual path greedily maximizes goodput, as it does for a single path.

4.4 Applying OEC

Applying OEC in an environment requires three tasks: i) specify W and r for greedy encoding; ii) estimate the bottleneck queue length to guide when coded packets are transmitted; and iii) if multiple paths exist, estimate current delay along each, so that the least delay path is used for each data packet. We describe below how we conduct these tasks in the vehicular environment, which is particularly challenging because it is highly dynamic.

OEC is designed to use all spare capacity to improve user performance. As such, it is more appropriate for settings where i) the underlying transmission channel isolates users from one another, as is the case for WWAN MACs; ii) the incremental cost of sending data is small, as is the case with fixed-price, unlimited-usage data plans. We revisit this issue in § 5.5.

5. PluriBus: OEC in moving vehicles

Figure 5 shows the architecture of PluriBus. The VanProxy is equipped with one or more WWAN links. All packets are relayed through LanProxy, which is located on the wired Internet.[2] Such relaying allows us to mask packet losses on the wireless links without modifying the remote computers to run PluriBus. OEC is used between the two proxies for data flowing in both directions.

We describe below how we accomplish the three tasks for applying OEC. Our methods for the latter two tasks borrow heavily from prior work.

5.1 Specifying W and r for greedy encoding

The sender initializes W=ϕ (i.e., empty set) and r=0 and updates these values when a data or coded packet is sent or feedback is received from the receiver.

[2]Relaying via LanProxy may increase end-to-end latency. However, because of the high delay inside wireless carrier networks, any increase is small if the LanProxy is deployed in the same city. Internet paths within a city tend to be short [40]. Interestingly, relaying through our deployed LanProxy actually reduced latency to most destinations due to Detour effects [37].

i) When a new data packet is sent, it is inserted in W and then r is updated to reflect the probability that the new packet is received. More precisely:

$$r \leftarrow ((|W| - 1) \cdot r + (1 - p))/|W|$$

where p is a rough estimate of the loss rate of the path along which the packet is sent. Receivers estimate p using an exponential average of past behavior and periodically inform the sender of the current estimate. Burstiness of losses can complicate the task of estimating loss rates. Our experiments show that PluriBus is robust to the inaccuracies that we find in practice [27].

ii) When a coded packet, formed by XOR-ing c data packets, is sent, W does not change, and r is updated to reflect the probability that the coded packet is received, and yielded a new packet. That is:

$$r \leftarrow (|W| \cdot r + (1 - p) \cdot Y(c))/(|W|)$$

where $Y(c)$ is the expected yield of the packet (Eq. 1).

iii) When the receiver returns the highest sequence number that it has received, which is embedded in packets flowing in the other direction (§5.4), packets with lower or equal sequence numbers are purged from W. We reset r to p.

The purge from W ensures that the sender encodes only over data packets generated roughly in the last round trip time. Because higher-layer protocols such as TCP detect losses and initiate recovery at this time-scale, it avoids duplicate recovery of packets. Thus, even though OEC logically uses all spare capacity, in practice it may not. No coded packets are sent when W is empty, that is, no new data packets have arrived in the last RTT.

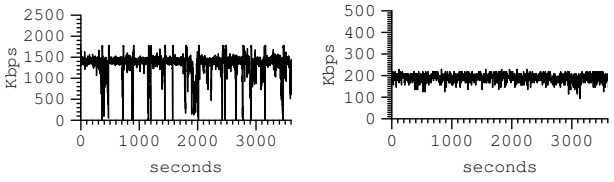
5.2 Estimating queue length

We maintain an estimate of queue length along a path in terms of the *time* required for the bottleneck queue to fully drain our packets. It is zero initially and is updated after packet transmissions:

$$Q \leftarrow \frac{(PacketSize * PathCapacity)}{max(0, Q - TimeSinceLastUpdate)}$$

$PathCapacity$ refers to the capacity of the path. The capacity of a path is the rate at which packets drain from queue at the bottleneck link. It is different from throughput, which refers to the rate at which packets reach the receiver. The two are equal only in the absence of losses. We conservatively estimate path capacity using a simple method described below.

The WWAN MAC protocols control media usage by individual transmitters, making it easier to estimate capacity than CSMA-based links (e.g., WiFi). As an example, Figure 6 shows the throughput of WiMax paths in the two directions for a one-hour window in which we generate traffic at 2 Mbps in each direction. We see roughly stable peak throughputs of 1500 and 200 Kbps, which



(a) Downlink (b) Uplink

Figure 6: WiMax downlink and uplink throughputs. The y-axis ranges of the two graphs are different.

correspond to their capacity. Incoming sequence numbers confirm that throughput dips are due to packet losses and not slowdowns in queue drain rate. For a detailed analysis of 3G link capacity, see [22].

Our capacity estimation is conservative, so that PluriBus is more likely send fewer coded packets than sending too many. The estimator, like other bandwidth estimation tools [16, 17], is based on a simple observation: if the sender sends a train of packets faster than the path capacity, the receive rate corresponds to the path capacity. However, unlike prior tools [16, 17], we do not use separate probe traffic. Instead, we rely on the burstiness of data traffic and the capacity-filling nature of OEC to create packet trains with a rate higher than path capacity.

We bootstrap the proxies with expected path capacities. The receiver measures the rate of incoming packets and computes the sending rate using the transmission timestamp in each packet. The two rates are computed over a fixed time interval (500 ms). The capacity estimate is updated based on intervals in which the sending rate is higher than the current estimate. If the receive rate is higher than the current estimate for three consecutive intervals, the estimate is increased to the median rate in those three intervals. Similarly, if the receive rate is lower for three consecutive intervals, the estimate is decreased to the median rate. Because our sending rate equals at least our estimated capacity, when actual capacity is lower, the estimate is downgraded quickly. Changes in capacity estimate are communicated to the sender.

Errors in capacity estimate can lead to errors in the queue length estimate. In theory, this error can grow unboundedly. In practice, we are aided by periods where little or no data is transmitted, which are common with current workloads. Such periods reset the estimate to its correct value of zero. While we cannot directly measure the accuracy of our queue length estimate, we show in §6.3.2 that our path delay estimate, which is based on it, is fairly accurate.

5.3 Identifying minimum delay path

When spreading data across multiple paths, PluriBus needs to estimate the current delay along each path. A simple method is to use the running average of one-way delays observed by recent packets, based on feedback from the receiver. However, we find that this method is quite inaccurate (§6.3.2) because of feedback delay and because it cannot capture with precision short time scale processes such as queue build-up along the path. Capturing such processes is important to consistently select the path with the minimum delay.

Our estimate of path delay is based on i) transmission time, which primarily depends on path capacity; ii) queue length; and iii) propagation delay. We described above how we estimate the first two. Measuring propagation delay requires finely synchronized clocks at the two ends, which may not be always available. We skirt this difficulty by observing that we can identify the faster path even if we only compute the propagation delay plus a constant that is unknown but same across all paths. This constant happens to be the current clock skew between the two proxies.

Let the propagation delay of a path be d and the (unknown) skew between the two proxy clocks be δ. We estimate $d + \delta$ based on Paxson's method [30]. A packet that is sent by the sender at local time s will be received by the receiver at local time r, where $r = s + d + \delta + Q + \frac{PacketSize}{PathCapacity}$. If there is no queuing, $d + \delta = r - s - \frac{PacketSize}{PathCapacity}$. We can thus compute $d+\delta$ using local timestamps of packets that see an empty queue.

To enable the estimate above, the receivers keep a running exponential average of $r - s - \frac{PacketSize}{PathCapacity}$ (i.e., $d + \delta$) for each path. Only packets that have likely sampled an empty queue are used for computing the average. Packets that get queued at bottleneck link arrive roughly $\frac{PacketSize}{PathCapacity}$ time units after the previous packet. We use in our estimates packets that arrive at least twice that much time after the previous packet. The running average is periodically reported by the receiver to the sender.

It is now straightforward for the sender to compute the path with least delay. This path is the one with the minimum value of $\frac{PacketSize}{PathCapacity} + Q + (d + \delta)$, which is in fact an estimate of the reception time at the receiver.

5.4 Implementation

We now briefly describe our implementation of PluriBus. We encountered and overcame many interesting engineering challenges while deploying PluriBus on our testbed. For instance, we need to correctly handle frequent IP address changes for the VanProxy in a way that is transparent to users and maintains their connections across changes. Due to lack of space, we omit most of these details from this paper and document them separately [27].

The VanProxy and the LanProxy create a bridge between them, and tunnel packets over the WWAN paths between them. The IP packets sent by users and remote computers are encapsulated within UDP packets that are sent over these paths. We do not use lower-overhead IP-in-IP tunneling as our wireless carriers block them. The

UDP packets include a special header that contains timestamps and additional information to allow the other end to correctly decode and order received packets. Each proxy caches incoming and decoded data packets for a brief period (50ms). This cache allows it to decode coded packets and temporarily store out of order packets. In-order data packets are relayed immediately. Those received out of order are relayed as soon as the previous packet is relayed or upon expiration from the cache.

The PluriBus header and the encapsulation lowers the effective link MTU by 41 bytes, which may lead to fragmentation issues (similar to those with VPNs). To minimize fragmentation, we inform the clients of the lower MTU via DHCP. Some clients inform their wide area peers of their MTU during TCP connection establishment, via the MSS option. For other clients, we are experimenting with modifying the MSS option of TCP SYNs as they traverse the VanProxy. With these changes, only large UDP packets destined for the clients, which constitute a small fraction in our traces, will be fragmented.

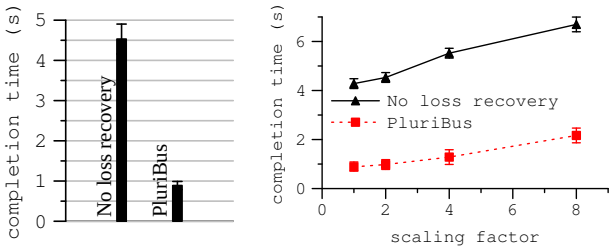
5.5 Discussion

PluriBus aggressively uses spare capacity. If transit operators subscribe to fixed-price, unlimited-usage plans, this "selfish" design maximizes user performance. However, if they have a usage-based plan, their costs will increase. In theory, this increase can be significant because OEC logically fills the pipe. But in practice PluriBus is not constantly transmitting because it encodes only over data in the last round trip time. We show later that PluriBus increases usage by only a factor of 2 for realistic workloads, with the increment being lower when the baseline demand is higher. We expect that transit operators would be willing to pay extra for better performance, as the cost of wireless access is likely a small fraction of their operational cost and amortizes over many users.

6. Evaluation

We now evaluate PluriBus. We show that it significantly improves application performance (§6.1) and that OEC outperforms loss recovery based on retransmissions or capacity-oblivious erasure coding (§6.2). We also provide microbenchmarks for some aspects of PluriBus (§6.3).

Experimental platforms: We deployed PluriBus on two buses that operate regularly on a corporate campus (§2). Each bus has one WiMax link and one EVDO link. The observed average loss rate is 5% for WiMax and under 1% for EVDO, though it can be bursty. The round trip delays are 40 and 150 ms respectively. The variations in path loss, delay and capacity are all natural; we do not control them in any way. A computer placed on each bus generates the workload described below. Because of support and manageability issues, we were not allowed to carry real user traffic on our experimental system. These two buses are our primary platform for studying the performance of PluriBus in a real environment. For more extensive experimentation and to consider different environments, we complement it with controlled experiments using a network emulator. To avoid confusion, we label our results with "Deployment" or "Emulator," depending on the platform used for the experiment.



(a) Measured workload (b) Scaled workload

Figure 7: Benefit of loss recovery in PluriBus. [Deployment]

Workload: For the experiments in this paper, we generate realistic, synthetic workloads from the traces described in §2.2. We first process the traces to obtain distributions of connection sizes and inter-arrival times, where a connection is the standard 5-tuple. The synthetic workload is based on these distributions of connection sizes and inter-arrival times [12]. The average demand of this workload is 86 Kbps but it is highly bursty.

To verify if our conclusions apply broadly, we also experimented with other workloads. These include workloads with a fixed number of TCP connections and those generated by a synthetic Web traffic generator [3]. The results are qualitatively similar to those below.

Performance measure: We use connection completion time as the primary measure of performance. It is of direct interest to interactive traffic such as short Web transfers that dominate the vehicular environment.

This paper uses the mean to aggregate performance across trials and connections. To show that the differences in means are statistically significant, we plot confidence intervals as well. Results that plot median and interquartile ranges can be found in our extended report [27].

6.1 Benefit of PluriBus

We start by studying the benefit of PluriBus compared to the current practice of not using any loss recovery (beyond end-to-end TCP). We study other loss recovery mechanisms in the next section. The results in this section are based on our deployment.

Figure 7(a) shows connection completion times for PluriBus and without any loss recovery. The latter uses delay-based path selection [9].[3] These results are based

[3]Today, more capacity, if needed, is added by installing addi-

on over four weeks of data. Each method operated for at least four days and completed tens of thousands of connections.

The graph shows the mean and 95% confidence intervals (CI) around the mean computed using Student's t distribution. We see that PluriBus significantly reduces completion time. Its mean completion time is under 1 second compared to over 4 seconds without loss recovery. This reduction represents a relative improvement factor of over 4 and an absolute improvement of over 3 seconds.

The reduction in completion time due to PluriBus can significantly improve user experience. Web transactions tend to have multiple connections (some sequential, some parallel) and even tens of milliseconds of additional delay can impact users' interaction with some Web sites [20].

Though not shown here, we find that PluriBus reduces the loss rate seen by end hosts to almost zero (0.3%). Without loss recovery, this loss rate is over 3%.

Benefit under higher load: Since the gains of PluriBus stem from using spare path capacity, an interesting question is whether these gains disappear as soon as the workload increases. To study the performance of PluriBus as a function of load, we scale the workload by scaling the inter-arrival times. To scale by a factor of two, we draw inter-arrival times from a distribution in which all inter-arrival times are half of the original values, while retaining the same connection size distribution. Our workload synthesis method does not capture many details, but it captures the primary characteristics that are relevant for our evaluation. We find that the performance for a synthetic workload scaled by a factor of 1 is similar to an exact replay of connection size and arrival times.

Figure 7(b) plots the mean and 95% confidence intervals of flow completion time as a function of the scaling factor used for the workload. Across both buses, these results are based on over four weeks of data. Each data point is based on at least two days. We see PluriBus performs well even when the workload is scaled by a factor of eight. In fact, its performance at that extreme level is better than what the absence of loss recovery offers without scaling the workload at all. Even at such high load levels, there is ample instantaneous spare capacity for PluriBus to mask losses by sending coded packets and improve performance (see §6.3.1). The loss rate seen by end hosts is roughly 0.5% with PluriBus, while it is 3% without any loss recovery.

tional VanProxies, each with its own WWAN link. Each user connects to exactly one VanProxy (i.e. an AP) and all her traffic is exchanged through that VanProxy. This policy balances load poorly because it operates at the granularity of users and often there are only a handful of active users. Our experiments (not shown here) confirm that its performance is poorer than that of delay-based path selection.

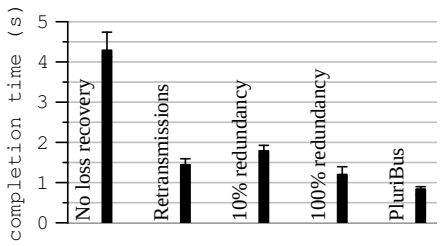


Figure 8: Performance of various loss recovery mechanisms. The graph plots the mean and (the top end of) 95% confidence interval for completion time. [Emulation]

6.2 Other loss recovery mechanisms

Having seen that loss recovery brings significant benefits in the vehicular environment, we now compare the use of OEC in PluriBus to other loss recovery mechanisms. We consider both retransmission based and erasure coding based loss recovery.

In retransmission-based loss recovery, the receiving proxy reports to the sender which packets have not been received, at which point the sender retransmits them. Both original packets as well as retransmissions are sent along the path that we currently estimate as offering the least delay. This policy provides an upper bound on policies such as pTCP [15] that do retransmission-based loss recovery because it uses the least delay channel and does not reduce the sending rate in response to losses.

The second loss recovery method that we consider is the capacity-oblivious erasure coding. We implement a code with K% redundancy by sending a coded packet after every $\frac{100}{K}$-th pure packet. Each coded packet codes over packets in the current unacknowledged window since the last coded packet. Thus, when K=10, every 11^{th} packet is coded. This code is identical to $(K, 1)$ Maelstrom code [2]. Both coded and pure packets are sent over the path with the least estimated delay.

The experiments in this section are based on emulation of the characteristics of wireless paths that we observe in our deployment. As described earlier (§2, §5.2), we have collected detailed traces to study the loss rate, delay and capacity of the wireless links in our testbed. We drive the emulation by updating emulated link's delay, loss, and capacity every second, as observed in the traces. The workload is as before, based on our traces.

Figure 8 shows the results. Notice that the "No Loss Recovery" and "PluriBus" bars are similar to those from deployment experiment (Figure 7(a)), which suggests that our emulation methodology is able to recreate the essential characteristics for these links.

We see that OEC-based loss recovery in PluriBus outperforms loss recovery based on both retransmissions and capacity-oblivious erasure coding. Compared to retransmissions, OEC's mean completion time is lower by 0.6 seconds (reduction factor of 1.7) because its loss recov-

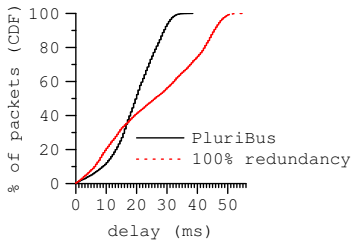


Figure 9: Delay experienced by data for two loss recovery methods. The graph plots the observed delay minus the minimum observed. [Emulation]

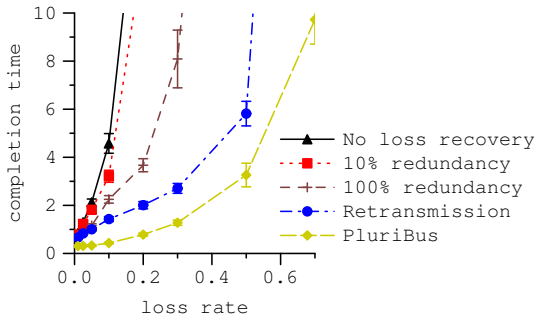


Figure 10: Performance of different loss recovery methods as a function of loss rate. The graph plots the mean and 95% confidence interval for completion time. [Emulation]

ery is faster. Compared to using erasure coding with 10% redundancy, its mean completion time is lower by 0.95 seconds (reduction factor of 2). This level of redundancy does poorly because it does not recover from many losses. Even though the average loss rate is low, the loss process is bursty and in periods of higher loss rates, using 10% redundancy is not sufficient. Data show that the post-recovery loss rate is 1.5%.

Compared to erasure coding with 100% redundancy, the mean completion time of PluriBus is lower by 0.4 seconds (reduction factor of 1.4). This level of redundancy is able to recover from most losses in our environment. Data show that the post-recovery loss rate is 0.5%. But by not being opportunistic, it imposes a higher queuing delay on data packets. This effect is shown in Figure 9, which plots the one-way delay, in addition to the minimum observed, for the two methods. We see that the 100% redundancy imposes a much higher delay on data.

The poorer performance of both ends of the redundancy levels relative to PluriBus, for different reasons, underscores the challenge in extracting good performance with capacity-oblivious erasure coding.

Impact of path loss rate: The results above demonstrate that OEC outperforms other loss recovery schemes under realistic path conditions. We now evaluate if the performance advantage of OEC persists in a range of settings with different loss rates.

To isolate the impact of loss rate, we perform emulation experiments with a single link between the two proxies. The link has a one-way delay of 75 ms and capacity of 1.5 Mbps. The loss rate on the link is varied from 1% to 70%. We show results using the Gilbert-Elliot (GE) loss model that induces bursty losses. Simpler loss models in which each packet has the same loss probability yield qualitatively similar results [27]. The GE model has two states, a good state with no (or low) loss and a bad state with high loss. The model is specified using the loss rate in the two states and state transition probabilities. We set both the transition probabilities to 0.5 and vary the loss rate of the bad state.

Figure 10 shows the results as a function of loss rate. We see that PluriBus outperforms other loss recovery methods across the board. These results also show that OEC performs better than capacity-oblivious erasure coding even if the coding overhead of these methods is adapted to loss rate. If we were to tune the overhead to expected loss rate, the overhead of the two erasure coding methods that we study must be suitable for some loss rate, but we see that OEC is better in the entire range. The reason is as explained earlier. Consider, for example, adding 100% redundancy. When less than half of the channel capacity is being used by data packets, OEC adds more than 100% redundancy and thus provides better protection, especially to the loss of many packets in a short time window. When more than half the channel is being used, OEC adds less than 100% redundancy. For the same amount of (coded+data) traffic that successfully reaches the receiver, the OEC traffic has more data than 100% redundancy traffic. The combined effect is that OEC tends to perform better than any capacity-oblivious redundancy method across a broad range of loss rates.

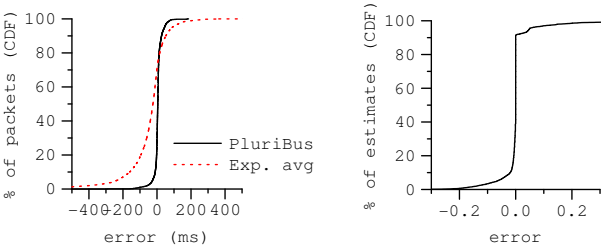
6.3 Understanding PluriBus in detail

We have studied the behavior of PluriBus in detail. In this paper, we report on the extra data sent by PluriBus due to coding and the accuracy of our delay and loss estimators. We defer to [27] other investigations such as the impact of inaccuracies in loss and delay estimates on performance, specific coding and decoding strategies we use, and fine-tuning of their parameters.

6.3.1 Amount of coded packets transmitted

Given that PluriBus is logically capacity filling, how many coded packets does it actually generate? Using data from the experiment in Figure 7(b), we find that the average percentage of coded packets is 54%. At scaling factors of 1, 2, 4 and 8, the percentage of coded packets is 67, 60, 57 and 35. Thus, as expected, PluriBus reduces the fraction of coded packets as workload increases because there are fewer opportunities to send coded packets.

We also find that while PluriBus logically fills the pipe,



(a) Path delay (b) Path loss rate

Figure 11: Error in estimating path delay and loss in PluriBus. [Deployment]

the actual amount of coded traffic is much lower because it codes over only data packets that arrive in the last RTT. At the scaling factor of 1, the average packet transmission rate of PluriBus is 258 Kbps, which is much lower than the combined capacity of the two links.

6.3.2 *Accuracy of path delay estimation*

Various factors, including estimates of path capacity, queue length, and propagation delay, impact the delay estimate of PluriBus. For good performance, the accuracy of this estimate is important. We evaluate accuracy by comparing the estimated delivery time at the sender to the actual delivery time at the receiver. This comparison is possible even with asynchronous clocks because our estimate of propagation delay includes the clock skew.

Figure 11(a) shows delay estimation error (i.e., estimate minus actual) in our deployment. It includes all load scaling factors; results are similar across all factors. The curve labeled PluriBus shows that our estimate is highly accurate, with 80% of the packets arriving within 10 ms of the predicted time. This is encouraging, especially considering the inherent variability in the delay of WWAN paths [22]. As a result of this accuracy, we find that fewer than 5% of the packets arrive out of order at the receiver and 95% of the out-of-order packet have to wait less than 10 ms in the sequencing buffer.

The curve marked "Exp. avg." shows the error if delays were estimated simply as an exponential average of observed delays, rather than our more precise accounting based on estimated capacity and queue length. Note that it tends to significantly underestimate path delay. We find that this underestimation significantly degrades performance, to a level that is sometimes worse than not using any loss recovery.

6.3.3 *Accuracy of loss rate estimation*

PluriBus uses an estimate of loss rate to estimate r, the probability of a packet being at the receiver, which is used in greedy encoding. Given the dynamics of the vehicular environment, loss rate maybe hard to estimate. Figure 11(b) shows that we obtain accurate estimates of loss rate in our deployment. It plots the difference in the loss rate for the next twenty packets minus the current running average of the loss rate that we use to predict future loss rate. Over 90% of the time, our estimate is within ± 0.1.

7. Additional related work

We now outline work that we build on, in addition to the work on combating path losses that we summarized earlier (§3).

Inverse multiplexing: Like PluriBus, many systems combine multiple links or paths into a single, high-performance communication channel. PluriBus differs primarily in its context and the generality of the problem tackled—our paths have disparate delays, capacities, and loss rates. Most existing works assume identical links [11], identical delays [39], or ignore losses [9, 34, 31].

A few systems, such as pTCP, R-MTP or MTCP, stripe data between end hosts across arbitrary paths by using TCP or a similar protocol along each path [15, 25, 5, 32]. Loss recovery is done via retransmissions. As we showed in §6, because of high path delays, this approach performs worse than PluriBus.

Delay-based striping, which we use to generalize OEC to multiple paths, was proposed by Chebrolu and Rao [9]. We combine it with loss recovery, which we find is important for it to be effective.

Improving connectivity for vehicles: Like us, MAR [34] and Horde [31] combine multiple WWAN links to improve vehicular Internet access. MAR showed the value of using multiple links using simple connection-level striping. It left open the task of building higher-performance algorithms. PluriBus employs one such algorithm (OEC). Horde [31] specifies a QoS API and stripes data as per policy. It requires that applications be re-written to use the API, while we support existing applications. Neither MAR nor Horde focus on loss recovery.

Some researchers have focused on improving WLAN (WiFi) connectivity to moving vehicles using lower-layer techniques such as rate adaptation and directional antennae [6, 29]. In contrast, we focus on WWAN links and on improving connectivity for applications by masking the deficiencies of connectivity provided by lower layers.

Erasure code: Numerous erasure codes have been proposed in the literature. The encoding used in OEC is a generalization of Growth codes [19] that were designed to transmit data in large sensor networks with failing sensors. Our generalizations include an explicit consideration of loss rate and data already at the receiver. The optimal degree of a coded packet (§4.2) is also derived by Considine [10]. However, that work does not address any of the systems issues (e.g., when to transmit packets).

8. Conclusions

Opportunistic erasure coding (OEC) is a new erasure coding scheme that varies the amount of coding overhead to fit the instantaneous spare capacity along a path. We built and deployed PluriBus, which applies OEC to a vehicular context, and found that it reduces the mean flow completion time by a factor of 4 for realistic workloads.

While we focused on the vehicular context, OEC is a general technique that can be used in other lossy environments where timely feedback is not available, e.g., wireless multicast and satellite links. Further, the two core mechanisms in OEC—opportunistic transmissions and greedy encoding—may be independently useful. Opportunistic transmissions can be used to transfer other kinds of low-priority data such that it uses only the capacity leftover by high-priority data. Greedy encoding can be used in other dynamic environments (e.g., wireless meshes) where the number of packets that will be received cannot be predicted in advance. We plan to study these possibilities in the future.

9. References

[1] H. Balakrishnan et al. Improving TCP/IP performance over wireless networks. In *MobiCom*, 1995.

[2] M. Balakrishnan et al. Maelstrom: Transparent error correction for lambda networks. In *NSDI*, 2008.

[3] P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *SIGMETRICS*, 1998.

[4] L. S. Brakmo and L. L. Peterson. TCP Vegas: End to end congestion avoidance on a global Internet. *IEEE JSAC*, 13(8), 1995.

[5] K. Brown and S. Singh. M-TCP: TCP for Mobile Cellular Networks . *CCR*, 27(5), 1997.

[6] J. Camp and E. Knightly. Modulation rate adaptation in urban and vehicular environments: cross-layer implementation and experimental evaluation. In *MobiCom*, 2008.

[7] R. Chakravorty, S. Katti, J. Crowcroft, and I. Pratt. Using TCP flow aggregation to enhance data experience of cellular wireless users. *IEEE JSAC*, 23(6), 2005.

[8] M. C. Chan and R. Ramjee. TCP/IP performance over 3G wireless links with rate and delay variation. *Wireless Networks (Kluwer)*, 11(1-2), 2005.

[9] K. Chebrolu and R. Rao. Bandwidth aggregation for real-time applications in heterogeneous wireless networks. *IEEE ToMC*, 4(5), 2006.

[10] J. Considine. Generating good degree distributions for sparse parity check codes using oracles, 2001.

[11] J. Duncanson. Inverse multiplexing. *IEEE Comm. Mag.*, 32(4), 1994.

[12] J. Eriksson et al. Feasibility study of mesh networks for all-wireless offices. In *MobiSys*, 2006.

[13] CIRA 3G mobile broadband router. http://www.feeneywireless.com/products/routers/cira/cira.php.

[14] N. Graychase. Greyhound launches in-bus Wi-Fi. http://www.wi-fiplanet.com/news/article.php/3736816.

[15] H.-Y. Hsieh and R. Sivakumar. ptcp: An end-to-end transport layer protocol for striped connections. In *ICNP*, 2002.

[16] N. Hu et al. Locating Internet bottlenecks: Algorithms, measurements, and implications. In *SIGCOMM*, 2004.

[17] M. Jain and C. Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with TCP throughput. In *SIGCOMM*, 2002.

[18] K. Jang et al. 3G and 3.5G wireless network performance measured from moving cars and high-speed trains. In *MICNET*, 2009.

[19] A. Kamra et al. Growth Codes: Maximizing Sensor Network Data Persistence. In *SIGCOMM*, 2006.

[20] R. Kohavi et al. Practical guide to controlled experiments on the web: Listen to your customers not to the hippo. In *SIGKDD*, 2007.

[21] S. Krishnamurthy, M. Faoutsos, and S. Tripathi. Split TCP for Mobile AdHoc Networks. In *Globecom*, 2002.

[22] X. Li et al. Experiences in a 3G network: interplay between the wireless channel and applications. In *MobiCom*, 2008.

[23] Y. Liao, Z. Zhang, B. Ryu, and L. Gao. Cooperative robust forwarding scheme in DTNs using erasure coding. In *MILCOM*, 2007.

[24] M. Luby. LT Codes. In *FOCS*, March 2002.

[25] L. Magalhaes and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *ICNP*, 2001.

[26] R. Mahajan et al. Eat all you can in an all-you-can-eat buffet: A case for aggressive resource usage. In *HotNets*, 2008.

[27] R. Mahajan et al. E pluribus unum: High performance connectivity on buses – extended version. MSR-TR-2009-76, June 2009.

[28] Does wi-fi on transit attract riders? http://www.masstransitmag.com/interactive/2010/11/04/does-wi-fi-on-transit-attract-riders/.

[29] V. Navda et al. Mobisteer: using steerable beam directional antenna for vehicular network access. In *MobiSys*, 2007.

[30] V. Paxson. *Measurements and Analysis of End-to-end Internet dynamics*. PhD thesis, UC Berkeley, 1997.

[31] A. Qureshi and J. Guttag. Horde: separating network striping policy from mechanism. In *MobiSys*, 2005.

[32] C. Raiciu, M. Handley, and D. Wischik. Coupled multipath-aware congestion control. IETF draft draft-raiciu-mptcp-congestion-01.txt, Mar. 2010.

[33] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *SIAM*, 8(2), June 1960.

[34] P. Rodriguez et al. MAR: A commuter router infrastructure for the mobile Internet. In *MobiSys*, 2004.

[35] F. Salmon. Did wifi cause a rise in bus ridership? http://blogs.reuters.com/felix-salmon/2011/12/26/did-wifi-cause-a-rise-in-bus-ridership/, 2011.

[36] S. Sanghavi. Intermediate performance of rateless codes. In *Information Theory Workshop*, 2007.

[37] S. Savage et al. The end-to-end effects of Internet path selection. In *SIGCOMM*, 1999.

[38] V. Sharma et al. MPLOT: A transport protocol exploiting multipath diversity using erasure codes. In *INFOCOM*, Apr. 2008.

[39] A. Snoeren. Adaptive inverse multiplexing for wide-area wireless networks. In *Globecom*, 1999.

[40] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. In *SIGCOMM*, 2003.

[41] V. Subramanian et al. An end-to-end transport protocol for extreme wireless network environments. In *MILCOM*, 2006.

[42] C. Tsao and R. Sivakumar. On effectively exploiting multiple wireless interfaces in mobile hosts. In *CONEXT*, 2009.

[43] WiFi incompatibility on Seattle metro? http://www.internettablettalk.com/forums/archive/index.php?t-18539.html.

[44] Sound transit riders. A mailing list for Microsoft employees.

[45] Google's buses help its workers beat the rush. http://www.nytimes.com/2007/03/10/technology/10google.html.

[46] Metro bus riders test county's first rolling WiFi hotspot. http://www.govtech.com/e-government/King-County-Metro-Bus-Riders-Test.html.

[47] Microsoft WiFi-enabled system will debut this month. http://seattlepi.nwsource.com/business/330745_msfttranspo07.html.

[48] Soundtransit – Wi-Fi. http://www.soundtransit.org/Rider-Guide/Wi-Fi.xml.