Achieving High Throughput Ground-to-UAV Transport via Parallel Links

Chit-Kwan Lin, H. T. Kung, Tsung-Han Lin, Stephen J. Tarsa, Dario Vlah
Harvard University
Cambridge, MA 02138
{cklin, htk, thlin, tarsa, dario}@eecs.harvard.edu

***Abstract*—Wireless data transfer under high mobility, as found in unmanned aerial vehicle (UAV) applications, is a challenge due to varying channel quality and extended link outages. We present FlowCode, an easily deployable link-layer solution utilizing multiple transmitters and receivers for the purpose of supporting existing transport protocols such as TCP in these scenarios. By using multiple transmitters and receivers and by exploiting the resulting antenna beam diversity and parallel transmission effects, FlowCode increases throughput and reception range. In emulation, we show that TCP over FlowCode gives greater goodput over a larger portion of the flight path, compared to an enhanced TCP protocol using the standard 802.11 MAC. In the process, we make a strong case for using trace-modulated emulation when developing distributed protocols for complex wireless environments.**

***Index Terms*—antenna beam diversity; 802.11; link layer; transport layer; UAV; network coding**

I. Introduction

In this paper, we focus on improving wireless data transfer from the ground to an unmanned aerial vehicle (UAV), a generic task of rising importance and interest for many UAV applications, e.g., in sensing and emergency rescue operations. Data transmission is challenging in this environment [1] since the ground-to-UAV wireless channel can experience long-lived outages and unpredictable, transient fades due to UAV mobility. Thus, one of our goals is to achieve high throughput and provide greater reception range under such adverse conditions. One approach would be to tailor a clean-slate protocol stack or modify existing protocols to fit these environments. We present an alternative: *FlowCode*, a link layer designed for the UAV environment that does not require modification of higher-layer protocols. This solution is near-term deployable, and supports existing transport protocols such as TCP (which was not designed to operate in such conditions, but is relied upon by many current applications of interest, e.g., database applications and MapReduce data processing). As an added advantage, our link layer approach is simple and requires no recasting of complex transport-layer semantics (e.g., TCP congestion control/fairness). Furthermore, our solution is easily deployable since it requires no tight synchronization amongst transmitters or receivers.

FlowCode exploits the antenna beam diversity present in multi-transmitter/multi-receiver (multi-Tx/Rx) systems. Despite wireless being a shared medium, each transmitter-receiver pair in these systems can act as a parallel, independent link [2], thereby increasing aggregate throughput. Such situations arise when the signal strengths from multiple transmitters differ at each receiver (due to geometry and differing antenna orientations), enabling different receivers aboard the UAV to capture different transmitters. Furthermore, diversity in antenna orientation can extend radio range: at a distance where the channel between one Tx-Rx pair is incoherent, another pair's antenna orientations may still permit reception. To realize these gains, we employ random linear network coding (1) to allow opportunistic packet delivery over *any* Tx-Rx link, (2) to ensure that every packet delivered is useful (i.e., *innovative*) with high probability, and (3) to do this without substantial control overhead.

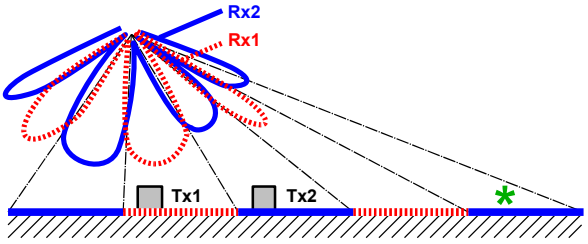

Fig. 1: An idealized scenario showing how antenna patterns and orientation can allow parallel transmissions to be received.

Developing and debugging our solution directly in our intended operating environment—where the UAV flies at 20m/s and covers a >800m range, in unpredictable weather and wind conditions—is untenable. Instead, we adopt trace-modulated emulation to aid development and devote precious flight time to collecting packet traces for use by an emulator.

The main contributions of this paper are as follows: (1) we present FlowCode, a link-layer solution that exploits the spatial diversity in multi-Tx/Rx systems to increase reception range and goodput of higher layer transport protocols; (2) we show a significant performance boost when using FlowCode underneath TCP in ground-to-UAV bulk data transfers; and (3) we demonstrate the advantages of trace-modulated emulation in distributed wireless protocol development/evaluation and present several lessons learned. While we focus on ground-to-UAV data transport in this paper, we plan to adapt our approach to the air-to-ground direction in the future.

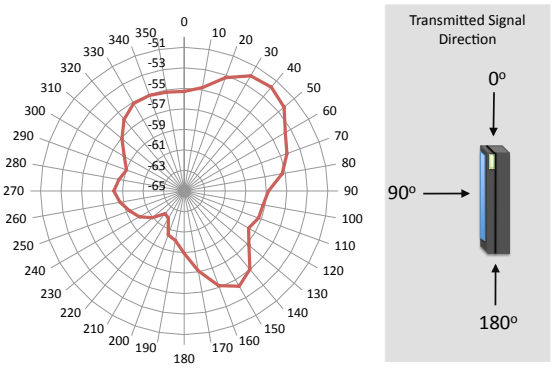

Fig. 2: The measured antenna radiation pattern of one of our UAV Rx nodes.

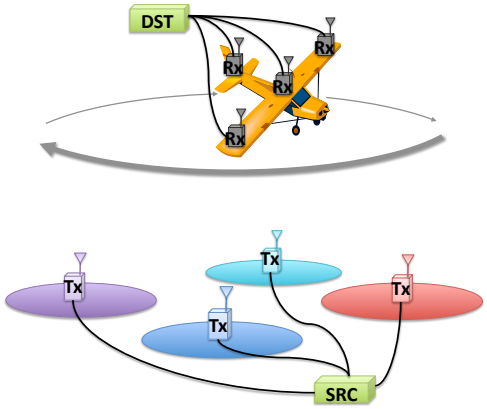

Fig. 3: Our canonical UAV scenario—a destination node is connected via wired Ethernet to four wireless receiver nodes on-board a low-altitude UAV. The UAV flies a circuit above a set of four wireless ground transmitter nodes connected to a source node via wired Ethernet.

II. Multi-Tx/Rx Antenna Beam Diversity

As mentioned in the previous section, the gains due to antenna beam diversity in our multi-Tx/Rx UAV system arise largely due to geometry and antenna patterns. To illustrate, Figure 1 shows an idealized scenario where two Rx nodes aboard a UAV pass over two ground Tx nodes. Since the lobes of the Rx antenna patterns exactly interleave, the received signal strengths of packets from Tx1 and Tx2 will differ significantly when the transmitters are separated by an appropriate distance. This enables the UAV to receive packets in parallel from Tx1 and Tx2. Though realistic scenarios are not as clean, the radiation patterns of the "omni-directional" antennas used in our field experiments (Figure 2) exhibit sufficiently different SNRs at various angles that a significant parallel transmission effect can be observed. Looking ahead, in Figure 6 (top, grey line), the parallel transmission effect allows throughput to momentarily exceed the 1Mbps single link rate, e.g., between 275s–300s. Note that this was achieved without purposeful placement or orientation of the Rx nodes on the UAV, which is in contrast to steerable antenna methods that require explicit synchronization and tuning. Our ad-hoc method is well-suited to the UAV environment where the dynamic channel makes it difficult to optimize antenna directions.

Besides increasing throughput, multi-Tx/Rx systems can also help extend reception range. Suppose that Tx2 in Figure 1 is instead located at the green asterisk (*). With Rx2, the UAV effectively extends its reception range to receive from a more distant Tx2; with only Rx1, Tx2 would have been out-of-range.

For the purpose of supporting transport protocols over multiple nodes, realizing the above two gains is insufficient. We must further address three issues in transporting a packet stream: (1) what to send, (2) when to send it, and (3) how to ensure it arrives. First, we perform random linear network coding [3] of transport layer flows (hence "FlowCode") so that *all* parallel transmissions will be useful with high probability, without explicit agreement between transmitters on which packets to send. This is important because parallel transmissions are only advantageous if they increase *goodput*, not simply throughput; we draw this distinction here and use the more precise goodput metric going forward. Second, transmissions should be coordinated to maximize parallelism. In the field, we have observed packet-level synchronization to be sufficient to induce the parallel transmission effect, and thus use a scheduling mechanism based on broadcast triggers. Finally, to improve delivery, we use an end-to-end retransmission scheme between the link-layer source and destination. Together, these three components comprise FlowCode and are detailed in the next section.

III. FlowCode Design and Implementation

To motivate FlowCode's design, we employ a canonical scenario that broadly captures various aspects of the UAV application environment. As shown in Figure 3, a source node on the ground is connected via wired Ethernet to four ground transmitters (Tx). To permit parallel transmissions, the ground nodes have CSMA disabled, either via software or through geographic separation. A UAV flying a circuit overhead carries a payload consisting of four wireless receiver (Rx) nodes connected to an on-board destination node via wired Ethernet. All nodes in the system speak the FlowCode protocol at the link layer, enabling the source to transfer data to the destination by relaying through the Tx's and Rx's.

The specific configuration presented above is easily generalized to accommodate more (or fewer) Tx or Rx nodes. The source and destination nodes can even be thought of as link-layer gateways in a larger network that are transparent to higher-layer protocols such as TCP. In the case of TCP, FlowCode offers an improved ground-to-UAV link without altering TCP semantics, a feature essential to applications that rely on TCP congestion control.

A. The FlowCode Protocol

With the above scenario in mind, we now describe FlowCode, a simple and lightweight link layer protocol comprised

of three mechanisms that work in concert: random linear network coding, transmission triggering, and link-layer retransmission. For brevity, we describe the protocol in the source-to-destination direction, but note that the reverse direction works the same way.

1) Random Linear Network Coding: Where multiple transmitters and receivers give rise to parallel links, we can gain from diversity: at the instant when one link is poor, another may be good. Independent random linear network coding at multiple transmitters, coupled with the broadcast nature of the wireless medium, allows us to automatically exploit these opportunities. Coded packets delivered along parallel network-coded links all have a high probability of increasing the rank of the decoding matrix at the destination, meaning links can individually convey useful information and together provide robustness against fluctuations in the channel.

At the source node, FlowCode groups k transport layer packets traveling outbound to the destination node into generations, each with a unique generation ID. These k packets are packed into a coding matrix, from which random linear combinations (RLCs) are generated by choosing random coefficients over a finite field. For each generation, each node transmits *coded frames* containing a coefficient vector along with the corresponding summation term. Network coding is carried out at all network nodes by choosing new random coefficients and combining RLCs. At the destination, once k linearly-independent RLCs have been received, the generation is decoded via Gauss-Jordan elimination to recover the transport-layer packets, which are then passed up the protocol stack. Transport packets moving in the opposite direction, from destination to source, are similarly encoded but in a different generation ID space. Thus, FlowCode does not seek the traditional network coding capacity gain due to intersecting packet streams [4]; we have found that behaviors of complex transport protocols like TCP (e.g., delayed ACK) present few opportunities to achieve this type of gain in practice.

2) Transmission Triggering: The random linear network coding scheme described above operates under the following transmission rubric. At the source, when k transport packets are available, a generation is packed immediately. When at least one but fewer than k packets are available, FlowCode waits a short time commensurate with the estimated round-trip time (RTT) before using dummy packets to fill the generation. This ensures that transport layer packets are not severely delayed so as to cause timeout for protocols like TCP. Once a generation is packed, the source broadcasts k coded frames, each containing a RLC, to the Tx nodes over its wired interface.

On arrival, RLCs are inserted into the corresponding generation's coding matrix. Reception of the broadcast RLC that gives the coding matrix full rank is simultaneous across Tx's and triggers each to send out s new RLCs from the same coding matrix over its wireless interface, where $s = \lceil k/n_{Tx} \rceil$ and n_{Tx} is the number of Tx nodes. That is, we (1) employ a jitter buffer of one generation at each Tx, ensuring that the RLCs generated are linearly independent with high probability and (2) in aggregate, we send as few RLCs as necessary to decode the generation at the destination, assuming no packet loss.

Rx nodes perform exactly the same coding operations as Tx nodes but are triggered to send a RLC to the destination for each one they receive from a Tx. While this one-to-one triggering can result in redundant RLCs arriving at the destination under low loss, it is necessary to ensure that innovative RLCs are pushed to the destination under high loss. At the destination, once enough RLCs are received to decode a generation, subsequent RLCs received for that generation are simply ignored.

3) Generation ARQ and Link-Layer Resend: To ensure that a transmitted generation is received at the destination, we employ two retransmission mechanisms. First, to recover from minor losses that antenna beam diversity and coding cannot mitigate, FlowCode uses a generation ARQ scheme, analogous to the 802.11 MAC ARQ. In FlowCode, a decoded generation is acknowledged via a link-layer ACK broadcast frame containing the decoded generation's ID; this ACK is relayed through the Rx and Tx nodes to the source. If the source does not receive an ACK within the ARQ timeout period, t_{ARQ}, it will trigger an additional RLC for the generation to be sent and will do so for up to kr RLCs, where r is the tunable maximum number of generation retries. Since FlowCode knows the maximum link rate and thus the frame transmission time, it knows to set t_{ARQ} to the transmission time of the number of generations in-flight (up to a maximum of f generations). We chose to have an end-to-end ARQ scheme between the source and destination rather than one that spans the wireless hop because only these nodes have complete status information on generation delivery.

The second retransmission mechanism is engaged in an extended outage, after ARQ retries have been exhausted, e.g., when the UAV flies out of range. This link-layer resend scheme seizes the first reception opportunity the moment the UAV returns within range by repeating, in sequence, un-ACKed generations at the full link rate whenever no new generations can be packed (e.g., when the transport layer is blocked or timed out). Such responsiveness can be important to the transport layer—e.g., after an extended outage, completing and decoding a resent generation brings TCP out of timeout. We can adopt this aggressive resend strategy because, in our scenario, there is no contention for the medium by multiple clients, as in a traditional 802.11 network. Instead, clients obtain fair share in relaying through the ground Tx nodes.

B. Implementation and Tuning Hints

We implemented FlowCode as a Linux userspace daemon, with an instance running on each node. At the source and destination, we use `ipqueue` to intercept and redirect transport layer packets from kernel space into the FlowCode daemon for encoding. Coded frames are implemented by encapsulation inside UDP broadcast packets sent out via the 802.11 wireless

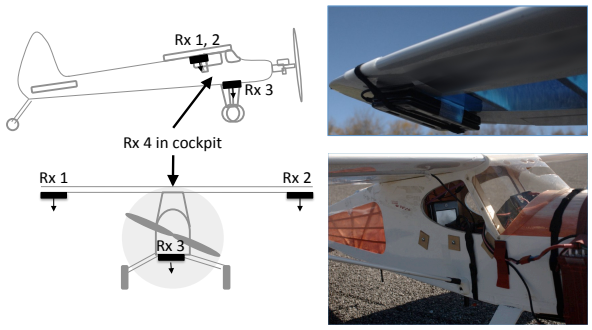

Fig. 4: Mounting positions of four receiver nodes on the UAV. All four nodes have antenna radiation patterns in different orientations.

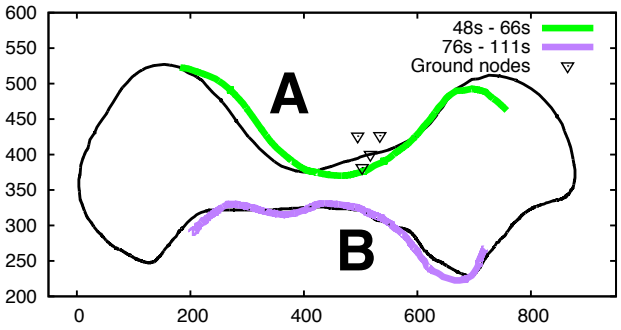

Fig. 5: A clockwise flight path (black) gives two regions of connectivity, A and B, as the UAV flies over the ground nodes.

interface. At the destination, decoded packets are injected into the network stack through a raw socket.

FlowCode's tunable parameters should be set as follows. Small generation sizes have the benefit of shorter decoding times and higher packing efficiency (i.e., fewer dummy packets used for padding); we use a generation size $k = 4$ throughout our experiments. Next, the maximum number of generations in flight f should be set depending on the generation RTT, which in turn depends upon processor speed of the nodes, since slow Gaussian elimination can cause the RTT to increase dramatically. Roughly, f should equal the number of generations that can be sent within the RTT at the aggregate link rate ($f = t_{RTT} n_{Tx}/t_{pkt} k$, where t_{pkt} is transmission time of one packet at a Tx's individual link rate). Finally, the maximum number of generation retries r should be set according to the expected loss characteristics of the receiving regions of the UAV flight path. For our flights, described in Section IV-A, $r = 10$ is sufficient.

IV. Trace-Modulated Emulation for Protocol Development

Distributed protocol development for the UAV environment is a special challenge because the high complexity of such protocols (due to timing, bottlenecks and faults) is compounded by the environment and logistics. First, flight tests are at the mercy of nature: poor weather grounds UAVs and wind can blow them off course. Second, UAVs require significant manpower to keep aloft and flights are limited in duration due to power constraints (our platform gives a maximum of 15min [5]). We therefore adopt trace-modulated emulation to facilitate protocol development and evaluation, and opt to devote precious flight time to packet trace acquisition instead.

Packet traces capture all the complexities of the wireless channel during flight, e.g., fades, collisions and capture effects. In trace-modulated emulation, such traces are used to modulate a packet stream, providing a controlled yet realistic emulation environment for protocol development and evaluation. By carefully designing acquisition campaigns, traces can capture a broad range of conditions, allowing for a simple emulator implementation. This is in contrast to model-based simulation [6], [7] and emulation [8], [9], where the complexity of the underlying model governs the realism of the emulation.

A. Trace Acquisition in the Field and Emulator Design

For trace acquisition, we use nodes with Marvell SD8686 802.11b/g radios set at the 1Mbps modulation. Figure 4 shows the mounting positions of the Rx nodes on our UAV. Figure 5 (black) shows the UAV flight path and the ground Tx locations (these were separated by ∼45m, effectively disabling CSMA). The UAV follows the flight path in a continuous loop passing through two regions of connectivity (A and B, in Figure 5) near the ground nodes. More details on the UAV, environment, and GPS/autopilot systems can be found in our previous work [5].

For each flight, Tx's continuously broadcast 1472-byte UDP packets at 1Mbps. Rx's log the sender, reception time, and sequence number of each packet received. In post-processing, these logs are converted into traces (essentially, binary strings: 1 indicates reception and 0 indicates loss). We collected traces from two configurations: one ground Tx broadcasting to one UAV Rx (1x1), and four ground Tx's broadcasting simultaneously to 4 UAV Rx's (4x4). With the 1x1 configuration, we used the best Tx (determined from previous flights) and collected four traces, one for each UAV Rx.

As the complexities of the ground-to-UAV channel are already captured in traces, our emulator is simple and consists of just three components: emulation machines, redirectors, and a modulator. Each emulation machine corresponds to a node in our field setup. A *redirector* daemon on each emulated node captures FlowCode's outgoing frames and redirects them to a central server running the *modulator* daemon. The modulator maintains, for each emulated link, a schedule with time slots corresponding to the packet transmission times in our traces. Frames arriving at the modulator are *modulated*, i.e., scheduled on the appropriate link if that link's corresponding trace allows. The modulator advances all schedules at the emulated link rate, forwarding scheduled packets to the intended recipients. This procedure is accurate when the modulated frames are equal in size to the packets in the trace; in our ensuing experiments, frames sent from the ground to the UAV are sized to fit this requirement. As a simplification, the modulator does not subject small packets (e.g., link-layer ARQ ACKs or transport layer control packets) to trace modulation. Instead,

small packets sent within Regions A and B, in either direction, are always successfully delivered; outside of these regions, they are always dropped.

V. Evaluation

The design philosophy behind FlowCode is to improve the link layer to support existing, unmodified transport protocols such as TCP in the UAV environment. As a point of comparison, we adopted the contrasting approach and implemented a minimalist modification to TCP that enables it to work over a single ground-to-UAV link, using the standard 802.11 MAC. Aside from misinterpreting wireless link loss as a congestion signal [10], standard TCP cannot cope with the extended outages in our UAV scenario because exponential back-off increases TCP's retry timeout so much that transmission opportunities are missed when the UAV returns within range. We solve this problem by having TCP repeat the last un-ACKed packet at a low rate (e.g., 10kbps) whenever TCP releases no new sequence numbers. We call this enhancement *TCP with low-rate retry* (TCP+LRR).

Our evaluation compares the performance of unmodified, standard Linux TCP over FlowCode against that of TCP+LRR over the standard 802.11 MAC with ARQ. For TCP over FlowCode, we use parallel links in a 4-Tx/4-Rx configuration and set the following FlowCode parameters: $n_{Tx} = 4, k = 4, f = 2, r = 10$. For TCP+LRR, we use a single link setup and set the maximum number of 802.11 MAC retries to 15. In both cases, we emulate the wireless links with the corresponding collected trace—for TCP over FlowCode, we use the 4x4 trace and for TCP+LRR, we use the best 1x1 trace out of the four collected. As a load generator drives the two experiments with infinite offered load, we measure the following metrics: (1) *goodput*; (2) *range*, or the length of the flight path segment over which the UAV can receive; and (3) *efficiency*, or the percentage of the maximum achievable goodput achieved over the entire flight. The maximum achievable goodput for each second of flight time is derived by calculating the total UDP goodput over that 1s window in the UDP packet trace (or, in the 4x4 case, across multiple traces).

Figure 6 shows the measured per-second goodput of TCP over FlowCode (top) and TCP+LRR (bottom) over 4.5 laps of the emulated flight, as compared to the maximum achievable goodput calculated from the traces. Here, we should keep in mind that the maximum achievable goodput is an optimistic measure; since TCP is sensitive to specific sequences of packet runs and gaps, it may not be able to always achieve the maximum goodput in practice.

First, observe that parallel transmissions boost the maximum achievable goodput in the 4-Tx/4-Rx configuration above 1Mbps, the modulation rate of a single link. Second, in both runs, each burst of delivery is labeled with the corresponding portion of the flight path (Regions A or B, in Figure 5). Note that, in both panels, almost all bursts commence with a large spike in goodput; this is due recovery from loss and subsequent in-order delivery of a large number of buffered packets. More importantly, observe that the burst lengths for TCP over FlowCode are noticeably longer than those for TCP+LRR, meaning FlowCode improves receiving range.

To better quantify these differences in performance, we present average goodput and range statistics over Region A and Region B in Table I. TCP over FlowCode wins decisively over TCP+LRR in Region A in both goodput (+31%) and range (+28%), but shows just modest gains in Region B (goodput: +2%, range: +3%). This is due to asymmetry in the flight path. In Region A, the UAV experiences better link quality since it passes closer to the ground Tx nodes. This is further improved by FlowCode, which presents TCP with an even more stable link. In contrast, Region B is farther from the ground nodes and experiences greater signal attenuation. FlowCode gives modest gains here because it has less with which to work.

Closer inspection of the statistics in Table I reveals a seeming discrepancy. Even though the range covered in Region B (544.71m) is significantly shorter than that covered in Region A (639.12m), according to Figure 6 the time it takes to traverse Region B is significantly longer. This is explained by a forceful headwind that blew continuously from the west (left, in Figure 5) the day we collected the traces. Consequently, the UAV traveled slower in Region B and faster in Region A. That our emulator revealed this is not only evidence of its correctness but also of its utility.

Finally, Figure 7 shows that the median efficiency of TCP over FlowCode (72.40%) was comparable to that of TCP+LRR (76.35%). Still, there remain optimization opportunities for FlowCode. At present, FlowCode's major performance bottleneck is the Gaussian elimination performed whenever a new RLC is inserted into a coding matrix. These operations amount to a processing delay of 1.5ms per generation on our current hardware. At a 1Mbps link rate, this accounts for $\sim$12% less goodput. Hand optimization of the Gaussian elimination routine should remedy this.

In summary, TCP over FlowCode exhibited better performance than TCP+LRR. In some cases, it might seem that the gains were modest. However, recall that the trace we used for TCP+LRR was from the best Tx-Rx pair available. In practical applications, it is generally difficult to know *a priori* which Tx-Rx pair is best; choosing incorrectly can lead to disastrously poor performance. Thus, in a sense, FlowCode is attractive because it renders choosing unnecessary: the system automatically uses all links as best it can.

A. Emulation: Lessons Learned

We lack space to present a full validation of our emulator, but note that TCP goodput in Figure 6 tightly tracks the maximum achievable goodput from the traces. This is strong evidence of correctness since it can only be achieved at such a fine granularity when the emulator introduces no artifacts.

We found that emulation was an indispensable tool in developing FlowCode. In particular, the repeatability afforded by trace modulation was especially useful in flushing out

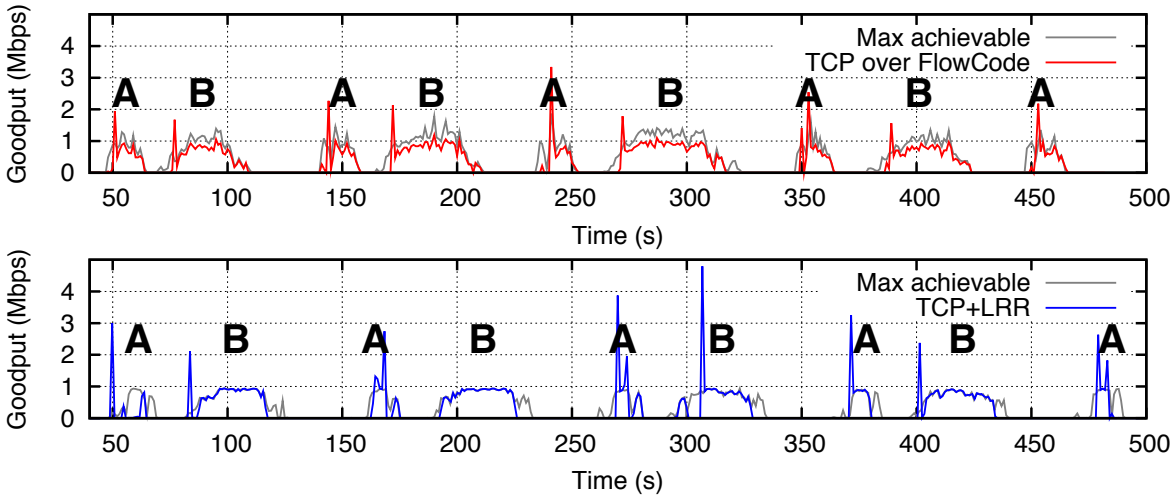

Fig. 6: Measured goodput as compared to the maximum achievable over emulated flights. Each burst of delivery occurs in the correspondingly labeled region of the flight path (see Figure 5). *Top:* TCP over FlowCode, using four Tx's and four Rx's. *Bottom:* TCP+LRR, using one Tx and one Rx.

		Max Achievable		Emulated	
		Goodput (Mbps)	Range (m)	Goodput (Mbps)	Range (m)
Region A	TCP over FlowCode (4-Tx/4-Rx)	$\mathbf{0.74} \pm 0.04$	$\mathbf{748.99} \pm 49.80$	$\mathbf{0.55} \pm 0.04$	$\mathbf{639.12} \pm 59.97$
	TCP+LRR (1-Tx/1-Rx)	$\mathbf{0.49} \pm 0.07$	$\mathbf{673.30} \pm 81.99$	$\mathbf{0.42} \pm 0.12$	$\mathbf{497.54} \pm 119.39$
Region B	TCP over FlowCode (4-Tx/4-Rx)	$\mathbf{0.77} \pm 0.07$	$\mathbf{697.17} \pm 31.35$	$\mathbf{0.57} \pm 0.04$	$\mathbf{544.71} \pm 28.38$
	TCP+LRR (1-Tx/1-Rx)	$\mathbf{0.62} \pm 0.02$	$\mathbf{674.87} \pm 12.63$	$\mathbf{0.56} \pm 0.01$	$\mathbf{530.10} \pm 30.61$

TABLE I: Mean and standard deviation of goodput and range over Regions A and B.

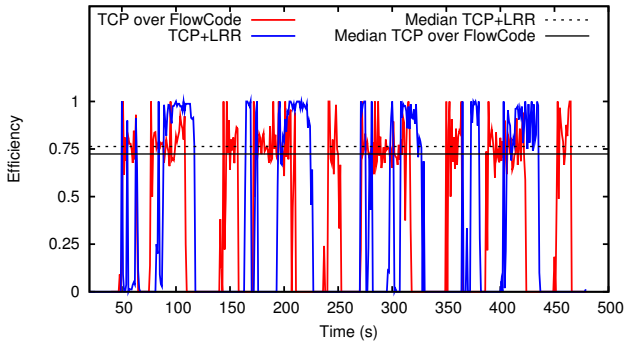

Fig. 7: Efficiency of TCP over FlowCode vs. TCP+LRR over their respective emulated flights. TCP over FlowCode median efficiency = 72.40%. TCP+LRR median efficiency = 76.35%.

problems due to timing and unexpected bottlenecks. For example, FlowCode was initially unable to achieve instantaneous goodput above 300kbps, even though the maximum achievable was >1Mbps. Under repeated trace emulation, we ultimately discovered that our emulation hardware had slow processors which were spending 10ms on Gaussian elimination per generation. Moving to faster machines immediately boosted goodputs to 1Mbps. Had this been a flight test, we would not have even known that we were underperforming at 300kbps.

Yet another benefit of trace-modulated emulation is found in parameter tuning. We settled on a FlowCode generation size of $k = 4$ after testing TCP behavior under other values. Larger generation sizes, e.g., $k = 16$, gave poor goodput due to low packing efficiency. This occurred when TCP experienced loss and cut its sending window size such that an insufficient number of packets was available to fill a generation. The larger the generation size, the more likely this was to occur.

VI. Conclusions and Future Work

FlowCode is an easily-deployable link-layer solution that enables high throughput ground-to-UAV data transport by higher-layer protocols such as TCP. It exploits the antenna beam diversity in multi-Tx/Rx systems arising from geometry and differences in antenna orientations. FlowCode uses random linear coding to enable opportunistic transmissions without requiring substantial control overhead or complexity.

We show that with FlowCode, TCP achieves goodput that approximates best-case performance, despite the fact that channel loss is well-known to cause TCP to underutilize wireless links. FlowCode also extends the effective communication range and increases the duration of high-rate transmission.

Additionally, we have shown the effectiveness of trace-based emulation in distributed protocol development and evaluation, especially for scenarios where field experimentation is difficult and the environment is fast-varying. By capturing the

behaviors of the communication channel (fading, collisions, and capture effects) in traces, we simplified emulator design while retaining a realistic emulation environment.

Our design enables a wide range of airborne applications that rely on TCP for data transmission. In the near future, and as the weather improves in our flight areas, we plan to further improve and validate system performance by running TCP over FlowCode in flight tests. Finally, the channel stability introduced by FlowCode makes TCP performance more tractable, suggesting that, for the UAV scenario, we can develop models and predict TCP performance with high accuracy via traces.

VII. Related Work

Work related to ours can be categorized into three areas: (1) improving communication performance over wireless networks with network coding, (2) increasing throughput by using multiple routing paths in parallel, and (3) improving TCP performance over lossy wireless links by adding reliability mechanisms.

Area 1 saw a surge in activity recently, starting with the seminal work of Ahlswede et al. [11], and followed by many protocol design and system implementations [3], [4], [12]. Area 2 is a classic networking problem. Interest in Area 3 surged after adoption of wireless LAN communications, starting with early protocols such as Snoop [13].

Combining Areas 1 and 2 are the CodeTorrent [14] and Rainbow [15] protocols, which employ network coding for data dissemination across wireless mesh networks, using multiple paths in the process. However, these protocols are tailored for a specific mode of transport and do not aim to support general transport layer protocols. There have also been more theoretical studies examining, e.g., optimal rate selection for network coding over parallel routing paths in wireless networks [16]. However, these works do not support non-trivial transport protocols like TCP.

Falling under Areas 2 and 3 are, e.g., the Multipath TCP [17] and PATHHEL [18] projects, which aim to allow TCP to take advantage of multiple routing paths. However, this comes at the expense of modifying TCP, either by terminating connections in the middle, or by changing the congestion control mechanisms to be multi-path aware. Alternative transport protocols specifically designed to aggregate multiple links [19] have also been put forth.

Finally, combining Areas 1 and 3 is work that couples network coding with TCP [20] by using equation rank instead of sequence numbers to identify segments and measure data transfer progress. To mask wireless network loss, the authors introduce an explicit retransmission mechanism underneath the network coding. Although the protocol improves performance, the tight coupling with TCP restricts its scope. Furthermore, the work does not explore the use of this mechanism over parallel transmission paths.

There exists a large body of work covered by strict subsets of the three areas. However, we are unaware of work that addresses all three, as we do.

Acknowledgments

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8750-10-2-0180. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. The authors would like to thank the Office of the Secretary of Defense (OSD/ASD(R&E)/RD/IS&CS) for their guidance and support of this research.

References

[1] T. Brown, B. Argrow, C. Dixon, and S. Doshi, "Ad Hoc UAV Ground Network (AUGNet)," in *AIAA 3rd "Unmanned Unlimited" Tech. Conf.*, 2004.

[2] J. Li, C. Blake, D. S. J. D. Couto, H. I. Lee, and R. Morris, "Capacity of ad hoc wireless networks," in *MobiCom*, 2001.

[3] P. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Proc. Ann. Allerton Conf.*, 2003.

[4] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard, and J. Crowcroft, "XORs in the air: practical wireless network coding," *ACM TON*, vol. 16, pp. 497–510, 2008.

[5] H. T. Kung, C.-K. Lin, T.-H. Lin, S. J. Tarsa, D. Vlah, D. Hague, M. Muccio, B. Poland, and B. Suter, "A location-dependent runs-and-gaps model for predicting tcp performance over a uav wireless channel," in *MILCOM*, 2010.

[6] http://www.opnet.com.

[7] S. McCanne, S. Floyd, K. Fall, K. Varadhan *et al.*, "Network Simulator ns-2," 2000.

[8] G. Judd and P. Steenkiste, "Repeatable and realistic wireless experimentation through physical emulation," *SIGCOMM Comp. Comm. Rev.*, vol. 34, pp. 63–68, 2004.

[9] B. D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz, "Trace-based mobile network emulation," in *SIGCOMM*, 1997.

[10] R. Caceres and L. Iftode, "Improving the performance of reliable transport protocols in mobile computing environments," *IEEE J. Sel. Areas in Comm.*, vol. 13, no. 5, pp. 850–857, 1995.

[11] R. Ahlswede, N. Cai, S. Li, and R. Yeung, "Network information flow," *IEEE Trans. Inf. Theory*, 2000.

[12] S. Chachulski, M. Jennings, S. Katti, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," in *SIGCOMM*, 2007.

[13] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, "Improving TCP/IP performance over wireless networks," in *1st Ann. Int'l Conf. on Mobile Computing and Networking*, 1995.

[14] U. Lee, J.-S. Park, J. Yeh, G. Pau, and M. Gerla, "Code torrent: content distribution using network coding in VANET," in *Mobishare*, 2006.

[15] C.-M. Cheng, H. T. Kung, C.-K. Lin, C.-Y. Su, and D. Vlah, "Rainbow: A wireless medium access control using network coding for multi-hop content distribution," *MILCOM*, 2008.

[16] X. Zhang and B. Li, "Optimized Multipath Network Coding in Lossy Wireless Networks," in *ICDCS*, 2008.

[17] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "MultiPath TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet," *ACM TON*, vol. 14, pp. 1260–1271, 2006.

[18] A. Baldini, L. D. Carli, and F. Risso, "Increasing Performance of TCP Data Transfers Through Multiple Parallel Connections," in *ISCC*, 2009.

[19] H. Hsieh and R. Sivakumar, "A transport Layer Approach for Achieving Aggregate Bandwidth on Multi-Homed Mobile Hosts," in *Mobicom*, 2002.

[20] J. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. Barros, "Network coding meets TCP," in *INFOCOM*, 2009.