Implementation of the AeroTP Transport Protocol in Python

Santosh Ajith Gogi, Dongsheng Zhang,
Egemen K. Çetinkaya, Justin P. Rohrer, and James P.G. Sterbenz
Department of Electrical Engineering & Computer Science
Information and Telecommunication Technology Center
The University of Kansas
Lawrence, KS 66045
{santoshag,dzhang,ekc,rohrej,jpgs}@ittc.ku.edu

ABSTRACT

The aeronautical transport protocol AeroTP addresses the challenges of end-to-end communication in the highly dynamic airborne telemetry network environment. The protocol has multiple modes: reliable, near-reliable, quasi-reliable, unreliable connection, and unreliable datagram. We present our Python implementation of AeroTP. The results of preliminary experiments conducted on Linux systems using AeroTP quasi-reliable mode are comparable to previous simulation results.

I. INTRODUCTION AND MOTIVATION

Airborne telemetry networks poses many challenges to end-to-end communication across the network. Traditional transport and network protocols do not perform well in this environment [1]. A domain-specific solution Airborne Network Telemetry Protocol (ANTP) suite was designed to optimize performance in this tactical environment [2]. This also aims to provide edge-to-edge compatibility with legacy TCP/IP-based Internet architecture. The protocol suite consists of *AeroTP* – a TCP-friendly transport protocol [3] with multiple QoS (quality of service) modes for the TmNS (telemetry network system), *AeroNP* – an IP-compatible network protocol (addressing and forwarding), and *AeroRP* – a routing protocol [4] that exploits location information to mitigate the short contact durations of high-velocity airborne nodes. The protocol suite is designed to perform well in an environment in which rapidly-changing topology prevents global routing convergence, as well as those in which persistent stable end-to-end paths do not exist. The protocols have been verified using the ns-3 simulator and analysis has shown significant performance improvement compared to traditional transport protocols [5]. Having verified these protocols, the next step towards deployment of the ANTP suite is developing a cross-platform implementation of protocols. The software architecture and implementation of the ANTP suite in Python was presented previously [6].

This paper presents the implementation architecture of AeroTP (aeronautical transport protocol) with emphasis on the reliability modes providing *end-to-end error control*. Closed-loop retransmission is provided by the ARQ (automatic repeat request) mechanism in reliable and near-reliable modes. Reliable mode preserves end-to-end acknowledgment semantics from source to destination as the only way to guarantee delivery. Near-reliable mode is highly reliable, but does not guarantee delivery since the gateway [7] uses split ARQ and immediately returns TCP ACKs to the source [3]. An open-loop error recovery is achieved using the FEC (forward error correction) mechanism in quasi-reliable mode. We also present

preliminary analysis of quasi-reliable mode in an emulated environment. AeroTP provides two additional modes that do not provide any reliability: unreliable connection and unreliable datagram. Unreliable connection mode does not use any error correction mechanism at the transport layer and relies exclusively on reliability at the link layer if available. Unreliable datagram mode is intended to pass UDP traffic transparently to maintain edge-to-edge compatibility. In this paper, we only consider operational modes providing error control that are applicable to the telemetry network.

The rest of the paper is organized as follows: Section II introduces the background on AeroTP with previously conducted tests using simulation techniques and related work. Section III presents the Python implementation of the AeroTP protocol focusing on the ARQ and FEC based reliability modes. In Section IV, performance analysis of quasi-reliable mode using FEC is presented. Section V concludes and gives future directions for this work.

II. BACKGROUND AND RELATED WORK

AeroTP is a *TCP-friendly* domain-specific transport protocol designed to meet the needs of the airborne network environment: dynamic resource sharing, QoS support for fairness and precedence, real-time data service, and bidirectional communication [2, 3]. AeroTP has several operational modes that support different service classes: reliable, nearly-reliable, quasi-reliable, unreliable connection, and unreliable datagram. The first of these is fully TCP compatible, the last fully UDP compatible, and the others TCP-friendly with reliability semantics matching the needs of the mission and capabilities of the network. AeroTP uses opportunistic connection establishment as an alternative to the three-way handshake used by TCP.

AeroTP has been designed and evaluated previously using the ns-3 simulator [8, 5]. In a lossy network environment, AeroTP reliable-mode performs significantly better compared to traditional TCP. With a higher BER (bit error rate), end-to-end delay in TCP increases by orders of magnitude whereas AeroTP has less than an order of magnitude increase. AeroTP quasi-reliable mode has much less loss due to the FEC recovery mechanism compared to UDP that drops packets with increasing error rates. Furthermore, AeroTP reliable mode has less overhead compared to TCP, while quasi-reliable mode does not cause increased delay but has significant overhead.

Many designs and implementations of network and transport protocols have been carried out. A transport layer protocol that relies on end-to-end mechanisms to detect network state was implemented in the Linux kernel and results were also verified with simulations [9]. The significant impact of a wireless multihop channel on TCP throughput and loss gives insight why TCP cannot be used in wireless networks [10]. A survey of many real-world experiments conducted in ad-hoc networks with emphasis on MANET (mobile ad-hoc network) routing protocols has been done [11]. Our work is different as AeroTP is a domain-specific transport protocol designed for end-to-end communication in airborne telemetry networks. The implementation of the AeroRP routing protocol and AeroNP network protocol upon which AeroTP is designed to run is presented in our companion paper [12].

III. IMPLEMENTATION OF AEROTP

The development and testing of AeroTP is carried out in the high-level scripting language Python. It provides a number of data structures and libraries, in particular, networking libraries such as sockets,

packing binary data, and CRC (cyclic redundancy check). Python-based implementations require less programming time leading to fast prototyping of applications [13].

AeroTP is implemented using a centralized top-level controller module that guides the flow setup, termination, and data transfer as shown in Figure 1, and employs supplemental modules designed for specific services. The controller runs independently on a POSIX operating system environment as a dæmon on both source and destination nodes to provide services to the application layer for end-to-end communication. The current implementation uses UDP sockets for compatibility with the security restrictions in PlanetLab testbed [14] systems, which is one of the emulation environments using the GpENI testbed.

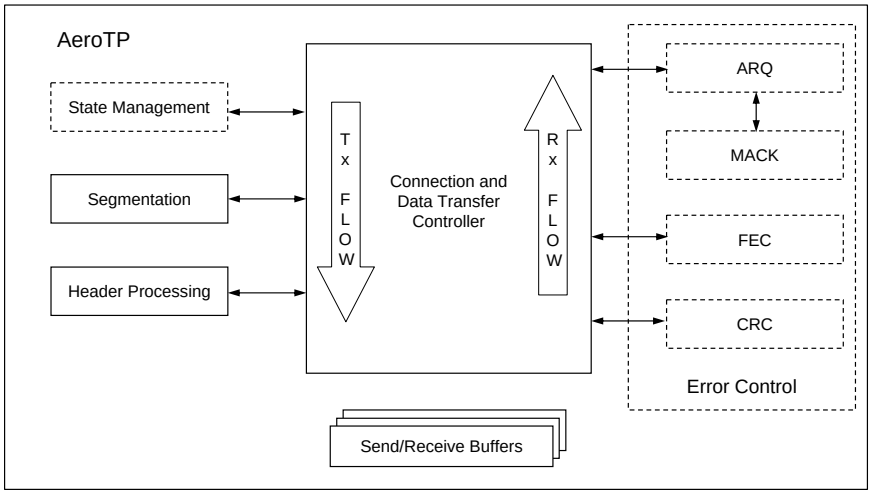

Figure 1: AeroTP implementation architecture

The controller flow at the source begins with connection setup that uses the state machine in connection-oriented modes. During the connection establishment, the overhead of a three-way handshake is eliminated by an opportunistic connection approach that overlaps data with control. Segmentation is followed with optional FEC error-correcting codes added to the payload. Header processing involves the application layer QoS and error control requirements embedded into the `AeroTP_Header` object. Depending on the class of service used by the application, error controlling mechanisms are employed in the subsequent steps. These functions are complimented appropriately at the destination by decomposing the segment based on service mode, maintaining state, and aggregating ACKs followed by the connection teardown. In addition to these modules, a consistent and efficient buffer-handling mechanism augments the main controller using sender and receiver buffers.

A. *End-to-end Reliability with ARQ mechanism*

AeroTP reliable and near-reliable mode use ARQ mechanisms to provide full reliability within the TmNS [8]. MACK (multiple ACK) is used as the acknowledgement mechanism. It dynamically aggregates a number of ACKs hinging upon the transmission rate and loss rate. The source and destination use control messages (ASYN, ASYNACK, AFIN, AFINACK) for connection establishment and termination as shown in Figure 2. The source initiates an end-to-end connection by sending the ASYN `AeroTP_Header` object with the SYN flag set. A piggyback flag is set to provide options to use traditional three-way handshake or opportunistic connection establishment. Once the ASYN is received from the source, the connection state

is set to ESTABLISHED (Figure 5) at the destination and the sequence number of the ASYN is recorded in a MACK control message to be sent back to the source. The first acknowledged sequence number is inserted in the MACK header and the following acknowledged sequence numbers are appended as the payload. The structure of a MACK control message is shown in Figure 3 where n and m are the first and last acknowledged TPDU sequence numbers, respectively.

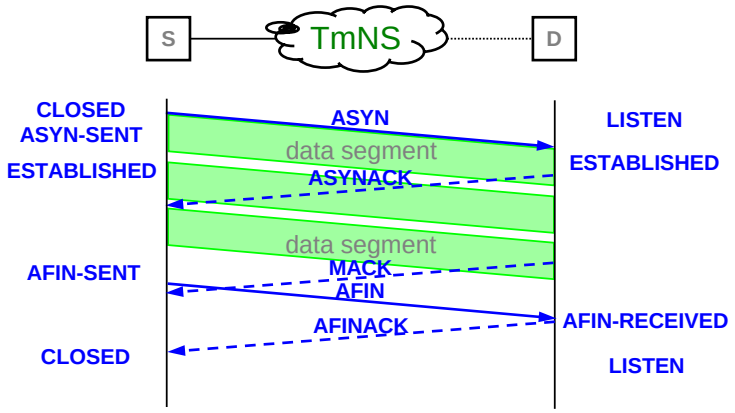

Figure 2: ARQ connection management

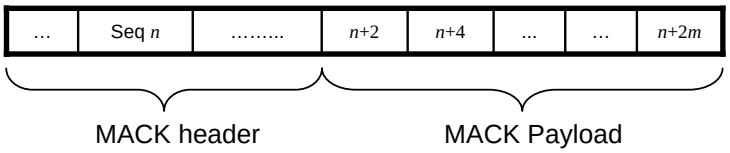

Figure 3: MACK header and payload

After the source sends out the ASYN, it moves to the ESTABLISHED state immediately. The source appends TPDUs (transport protocol data units) to the transmission buffer and sends them to the destination in FIFO (first in, first out) order. The AeroTP protocol employs a selective-repeat ARQ mechanism to provide a reliable edge-to-edge connection. Whenever the source sends out a TPDU, the sequence number and TPDU are stored in an additional buffer used for possible retransmission, implemented by using a hash table (`dict()` data structure in Python) with the sequence number as the keys. Figure 4 is an example showing the retransmission procedure. By checking the next MACK from the destination, the source will remove the sequence number with its counterpart TPDU sequence numbers (2, 6, 8, 10, and 12) in the retransmission buffer and append the unacknowledged TPDUs (sequence number 4) to sender buffer.

After the transmission buffer is empty, the source will send out an AFIN immediately to signal the end of the current connection. However, it is possible that some of the last sent several TPDUs are lost during the transmission. Therefore, an alternative retransmission mechanism is exploited. When the last MACK arrives at the source, if all the sequence numbers of the last set of TPDUs are acknowledged, the source will just wait for AFINACK to terminate the connection; if some sequence numbers are missed, it triggers a

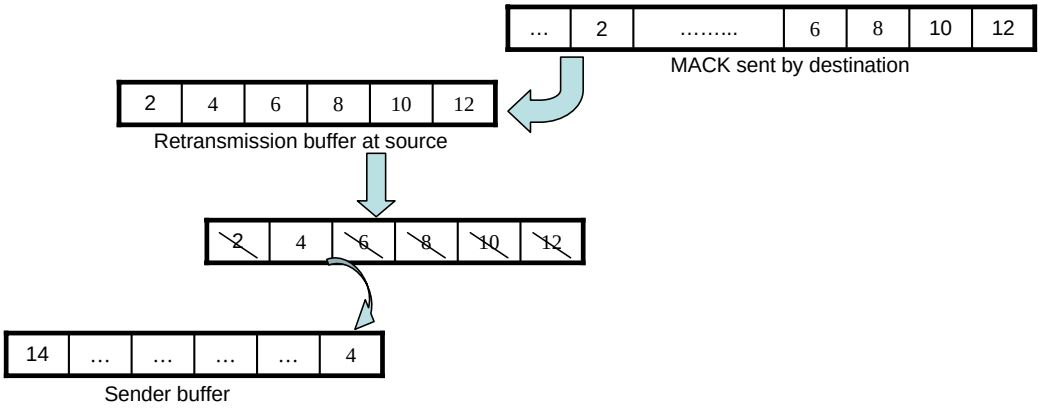

Figure 4: AeroTP ARQ retransmission mechanism

time-out mechanism for the retransmission of the last set of TPDUs. In this case, the destination will send an ACK back to the source immediately for each single TPDU so that each TPDU is acknowledged individually. A time-out thread is employed for each TPDU by using the Python class `threading.Timer`, and when the timer expires it will trigger automatic retransmission of that TPDU. When AFINACK reaches the source and no TPDUs are in the retransmission hash table, the source moves to the CLOSED state and the destination moves to the LISTEN state. The state management module described next, is used for effective event transitions.

The state management module is a FSM (finite state machine) based module [15] that effectively manages the state transitions in the case of AeroTP connection-oriented modes. The state transition diagram is shown in Figure 5. The module maintains AeroTP states and possible event transitions with action handlers. A set of tuples each representing event transitions (`input_symbol`, `current_state`, `action`, `next_state`) with optional `input_symbol` are initialized that vary depending on the AeroTP reliability mode. Then, based on an event occurrence, the state transitions from `current_state` to `new_state` and takes corresponding action. The events are listed in Table 1.

B. Quasi-Reliability with FEC mechanism

Due to the inherent behavior of bit-errors resulting from noise and interference in wireless links with dynamic mobility, relying alone on retransmissions to efficiently provide reliable data transfer is not sufficient. An open-loop error recovery mechanism such as FEC achieves an arbitrary level of statistical reliability. This also means it does not guarantee absolute delivery of data, while eliminating the need of ACKs and ARQ completely. We use the RS (Reed-Solomon) class of error correcting codes that are predominantly used in applications such as RAID-like systems, optical disk storage devices, and optical communication technologies. Quasi-reliable mode also manages a subset of the state transitions discussed earlier.

RS error correcting codes are linear block codes that can detect and correct multiple burst of errors. They represents data as polynomials and oversample them. An (n, k) code encodes k source data bytes into n codewords, with the ability to correct at most $(n - k)/2$ errors. We use the publicly available RS error correcting Python library [16] for our FEC mechanism.

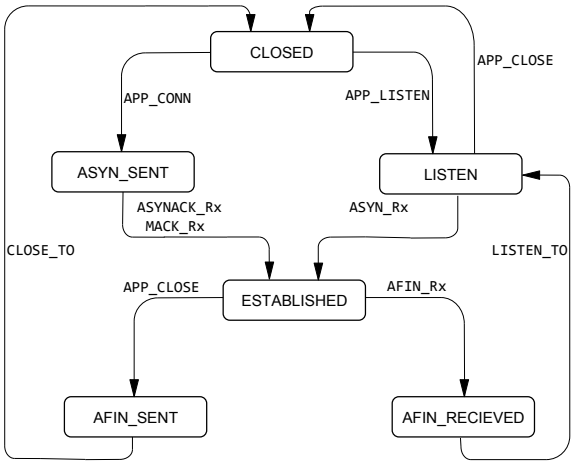

Figure 5: AeroTP state machine

A widely used RS code uses 8-bit symbols as it facilitates byte-oriented systems and is computationally less intensive. This corresponds to the number of symbols in the encoded block to be $n = 2^8 - 1 = 255$. To provide different error correcting capabilities, k can be varied. To account for a higher payload size in the TPDU, a mechanism to iteratively encode and decode codewords is incorporated.

- During encoding at source, chunks of codewords are computed with specified FEC strength and the default n being 255. Two conditions are possible: the sum of the lengths of codewords computed exceeds the length of the maximum payload size, or the length of the codewords is less than the block size of 255. In the former, n is calculated based on the strength whereas k is computed for blocks of smaller size in the latter case.
- During decoding at the destination, each chunk of the payload is decoded individually which involves computation of k. To simulate different noise levels in the network during analysis stage, a user-specified BER based `RateBasedErrModel` is developed. This introduces burst errors into the payload calculated using its length and BER.

An object of RS based error correcting class is created using a constructor with `n` and `k` arguments computed as discussed previously. FEC adds redundancy for error correcting capability that overcomes the necessity of retransmissions.

C. Supplemental modules

While the centralized controller provides the main functionality, supplemental modules serve as utilities to simplify operations and provide reusable modules across different AeroTP modes. These modules make use of the object-oriented features of Python. They act as independent libraries, instantiated by the centralized controller.

Segmentation and header processing modules are a prerequisite for processing related to size and alignment of control and payload data during network communication. The segmentation module accepts application data and segments into chunks for transmission. In contrast, the destination side decapsulates the segment from the receiver buffer `RECV_BUFF`. Segmentation also provides the sequence

Table 1: State transition definitions

State	Description
CLOSED	State in which no connection exists and no data is transferred
LISTEN	State in which a destination is ready to listen to any incoming data
ASYN_SENT	ASYN message sent by the host initiating connection
ESTABLISHED	Steady state in which data transfer takes place
AFIN_SENT	AFIN message sent to indicate no new data being sent
AFIN_RECEIVED	AFIN message received and AFINACK is sent as an acknowledgement
APP_CONN	Request issued by the application to initiate connection by sending ASYN
APP_LISTEN	Request issued to the receiving host to move to the LISTEN state
APP_CLOSE	Request to initiate closing a connection by sending AFIN
ASYN_RX	ASYN received, indicating a connection has been requested
ASYNACK_RX	ASYNACK received, indicating connection request has been granted
MACK_RX	Single or multiple packet ACK received
AFIN_RX	AFIN received, indicating end of any new data, initiating connection close
CLOSE_TO	A timeout allowing outstanding retransmissions before going to CLOSED state
LISTEN_TO	A timeout to go to LISTEN state so that the destination can receive all data packets

numbering mechanism, which numbers each TPDU to permit resequencing. Header processing at the source helps in constructing the header in network byte order using the binary packing library method `struct.pack(format, v1, v2, ...)`, which can easily be decomposed at the destination. The arguments `v1,v2,..` representing header fields are packed according to `format`. This module provides extensive support in terms of handling header field attributes.

Various other modules serve as tools to validate and analyze the overall implementation. A utility module converts values between different numerical base formats and provides time related functions using the `datetime` library. A traffic generator module generates CBR (constant bit rate) application traffic at a specified rate, which emulates end-to-end application layer protocol behavior. The CRC protects the integrity of data end-to-end across the network. It can be used in conjunction with FEC to detect errors. The HEC (header error check) is a 16-byte strong CRC whereas the payload has a 32-byte CRC performed by the Python library module `binascii`.

IV. PERFORMANCE ANALYSIS

Functional testing of all AeroTP modes is initially conducted on the GpENI testbed [17, 18]. In this section, we analyze the performance of AeroTP quasi-reliable mode in a lossy environment. In order to test the accuracy of the FEC mechanism, we have simplified the testing on two systems operating with the Debian Linux distribution. 1 MB application data is transmitted from one of the systems to another during a single scenario. FEC strengths provided by RS codes ranging from $(255, 254)$ to $(255,74)$ are tested. Note that the higher the difference between n and k, the higher the strength. We measure the cumulative goodput and overhead of using different FEC strengths that provides statistical reliability. Emulation of errors with varying rates is done at the destination end receiving interface.

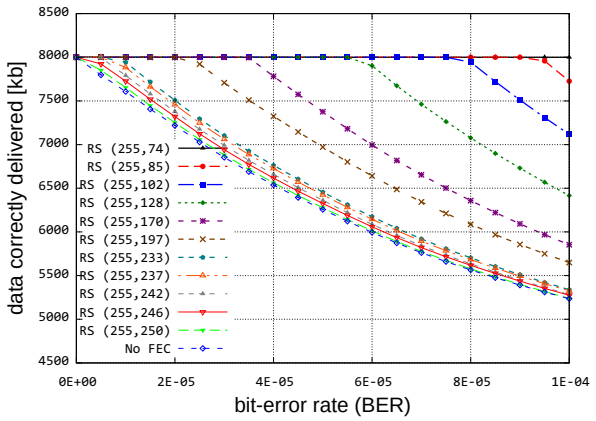

Figure 6: Cumulative goodput

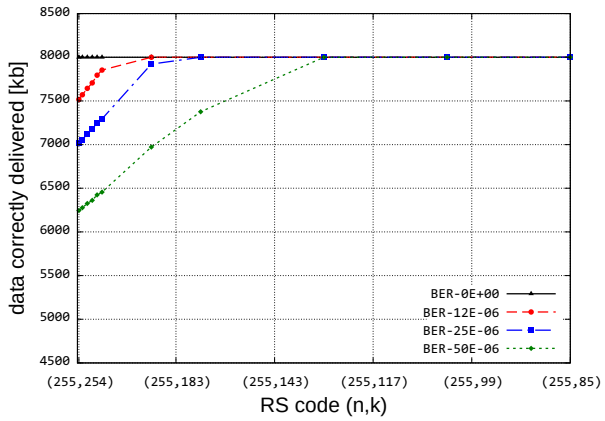

Figure 7: Cumulative goodput

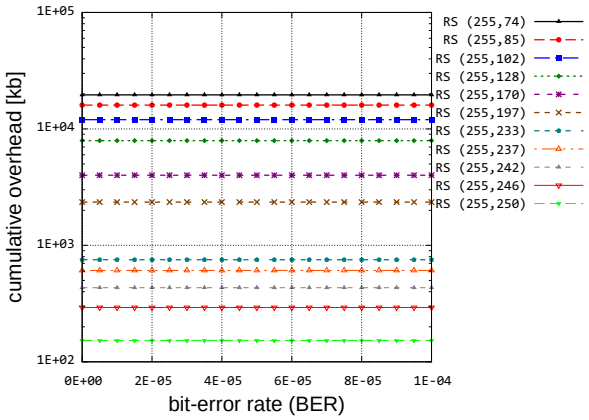

Figure 8: Cumulative overhead

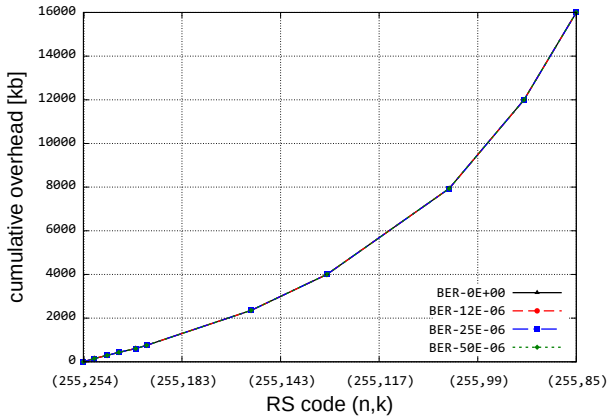

Figure 9: Cumulative overhead

Figure 6 shows the amount of data correctly delivered as BER increases with varying FEC strength. At the error rates tested, except for high strength RS (255,74) code, the total amount of correctly delivered data decreases as the error rate increases. The AeroTP quasi-reliable mode eliminates the need for ACKs and retransmissions in the case of lossy links that cause bit errors at the cost of not *guaranteeing* reliability. Due to increased noise and interference in the wireless channel, more bit errors reduce the amount of correctly delivered data. This can be compensated by error correction with sufficient FEC strength. Figure 7 shows the amount of data correctly delivered for different FEC strength with lower error rates. Each FEC strength has a threshold BER that can deliver entire data correctly. An FEC strength of RS (255,128) code is able to correct errors occurring at rate $\leq 5.0 \times 10^{-5}$.

The next set of plots indicate the overhead added by the FEC mechanism. Figure 8 shows that as the FEC strength is increased, the overhead added by the RS codes increases (note the log y-axis scale). This means more packets are required to carry the application data. The reliability induced in quasi-reliable mode comes at the cost of additional FEC codewords added to the packet, reducing the application data in each packet. Note that error rate does not affect the overhead for a given strength. Figure 9 shows

the increase in the overhead with increased FEC strength, quantified by the number of packets sent. The results are comparable to the simulation results [5]. Significant overhead is added that can influence the FEC strength value to be used in AeroTP. The strength value can be adapted based on the error-rates, by cross-layering link characteristics with link and MAC (media access control) layer.

V. CONCLUSIONS AND FUTURE WORK

The implementation of AeroTP protocol extends the previous work of simulation to real-world systems. The performance analysis of quasi-reliable mode that is based on an open-loop error recovery mechanism shows comparable results to simulation results. Cross-layer optimizations can be considered in order to effectively use quasi-reliable mode, given the overhead of FEC. Future work includes deployment and performance analysis of ANTP suite on mobile devices in a mobile ad-hoc network environment. Furthermore, a hybrid protocol that uses ARQ mechanism as an extension to FEC mechanism [19] in the case of loss of packets and/or high error rates is being considered.

ACKNOWLEDGEMENTS

The authors would like to thank the Test Resource Management Center (TRMC) Test and Evaluation/Science and Technology (T&E/S&T) Program for their support. This work was funded in part by the T&E/S&T Program through the Army PEO STRI Contracting Office, contract number W900KK-09-C-0019 for AeroNP and AeroTP: Aeronautical Network and Transport Protocols for iNET (ANTP). The Executing Agent and Program Manager work out of the AFFTC. This work was also funded in part by the International Foundation for Telemetering (IFT). We would like to thank Kip Temple and the membership of the iNET working group for discussions that led to this work.

REFERENCES

[1] J. P. Rohrer, A. Jabbar, E. K. Çetinkaya, and J. P. Sterbenz, "Airborne telemetry networks: Challenges and solutions in the ANTP suite," in *Proceedings of the IEEE Military Communications Conference (MILCOM)*, (San Jose, CA), pp. 74–79, November 2010.

[2] J. P. Rohrer, A. Jabbar, E. K. Çetinkaya, E. Perrins, and J. P. Sterbenz, "Highly-dynamic cross-layered aeronautical network architecture," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 47, pp. 2742–2765, October 2011.

[3] J. P. Rohrer, E. Perrins, and J. P. G. Sterbenz, "End-to-end disruption-tolerant transport protocol issues and design for airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2008.

[4] A. Jabbar, E. Perrins, and J. P. G. Sterbenz, "A cross-layered protocol architecture for highly-dynamic multihop airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2008.

[5] K. S. Pathapati, T. A. N. Nguyen, J. P. Rohrer, and J. P. Sterbenz, "Performance analysis of the AeroTP transport protocol for highly-dynamic airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2011.

[6] M. Alenazi, S. A. Gogi, D. Zhang, E. K. Çetinkaya, J. P. Rohrer, and J. P. G. Sterbenz, "ANTP Protocol Suite Software Implementation Architecture in Python," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2011.

[7] E. K. Çetinkaya and J. P. G. Sterbenz, "Aeronautical Gateways: Supporting TCP/IP-based Devices and Applications over Modern Telemetry Networks," in *Proceedings of the International Telemetering Conference (ITC)*, (Las Vegas, NV), October 2009.

[8] K. S. Pathapati, J. P. Rohrer, and J. P. G. Sterbenz, "Edge-to-edge ARQ: Transport-layer reliability for airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2010.

[9] Z. Fu, B. Greenstein, X. Meng, and S. Lu, "Design and Implementation of a TCP-Friendly Transport Protocol for Ad Hoc Wireless Networks," in *Proceedings of the 10th IEEE International Conference on Network Protocols*, pp. 216–225, November 2002.

[10] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The impact of multihop wireless channel on TCP throughput and loss," in *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 1744–1753, 2003.

[11] W. Kiess and M. Mauve, "A survey on real-world implementations of mobile ad-hoc networks," *Ad Hoc Networks*, vol. 5, no. 3, pp. 324–339, 2007.

[12] M. J. Alenazi, E. K. Çetinkaya, J. P. Rohrer, and J. P. G. Sterbenz, "Implementation of the AeroRP and AeroNP Protocols in Python," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2012.

[13] L. Prechelt, "An Empirical Comparison of Seven Programming Languages," *IEEE Computer*, vol. 33, no. 10, pp. 23–29, 2000.

[14] "PlanetLab." `http://www.planet-lab.org/`, November 2009.

[15] N. Spurrier, "FSM in Python." `http://www.noah.org/python/FSM`, 2002.

[16] A. Brown, "Reed-Solomon class of error correcting codes in Python." `https://github.com/brownan/Reed-Solomon`, 2009.

[17] J. P. G. Sterbenz, D. Medhi, B. Ramamurthy, C. Scoglio, D. Hutchison, B. Plattner, T. Anjali, A. Scott, C. Buffington, G. E. Monaco, D. Gruenbacher, R. McMullen, J. P. Rohrer, J. Sherrell, P. Angu, R. Cherukuri, H. Qian, and N. Tare, "The Great plains Environment for Network Innovation (GpENI): A programmable testbed for future internet architecture research," in *Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, (Berlin, Germany), pp. 428–441, May 2010.

[18] J. P. G. Sterbenz, D. Medhi, G. Monaco, B. Ramamurthy, C. Scoglio, B.-Y. Choi, J. B. Evans, D. Gruenbacher, R. Hui, W. Kaplow, G. Minden, and J. Verrant, "Gpeni: Great plains environment for network innovation." `http://wiki.ittc.ku.edu/gpeni`, November 2009.

[19] K. S. Pathapati, J. P. Rohrer, and J. P. Sterbenz, "Comparison of adaptive transport layer error-control mechanisms for highly-dynamic airborne telemetry networks," in *Proceedings of the International Telemetering Conference (ITC)*, (San Diego, CA), October 2012.