End-to-End Adaptive Error Control for a Resilient and Disruption Tolerant Transport Protocol

Kamakshi Sirisha Pathapati

Submitted to the graduate degree program in Electrical Engineering & Computer Science and the Graduate Faculty of the University of Kansas School of Engineering in partial fulfillment of the requirements for the degree of Master of Science

Thesis Committee:

Dr. James P.G. Sterbenz: Chairperson

Dr. Gary J. Minden

Dr. Bo Luo

Date Defended: 07/20/2012

The Thesis Committee for Kamakshi Sirisha Pathapati certifies
that this is the approved version of the following thesis:

End-to-End Adaptive Error Control for a Resilient and Disruption Tolerant Transport Protocol

Committee:

Chairperson: Dr. James P.G. Sterbenz

Date Approved: 07/27/2012

Abstract

While there has been an increase in the growth of networking technologies to suit the current demand of applications and users, networks are still susceptible to disruption. Communication networks operating in any domain come with inherent challenges that make end-to-end connections harder to maintain. This argument calls for protocols that are disruption tolerant and can offer resilience. Res-TP is a new transport protocol that directly addresses the challenges posed by challenged networks. It offers QoS (quality of service) and varying degrees of reliability depending on the class of data being communicated. Apart from a reliable-connection mode which offers full end-to-end reliability using ARQ (automatic repeat request), the protocol also includes quasi-reliable mode that offers statistical reliability by using end-to-end FEC (forward error correction) codes, unreliable-connection mode that does not implement either ARQ or FEC but relies on link-layer FEC, and unreliable datagram that transparently passes UDP traffic. While the fully-reliable connection mode offers closed-loop error control, the quasi-reliable mode offers open-loop error control. This leaves a gap within the spectrum to analyze the benefits of employing both closed-loop and open-loop error control. Hyrbid error control, with both FEC and ARQ, is a simple way to offer high end-to-end reliability and performance during moderate error conditions. In this thesis, the reliable-connection mode (ResTP-ARQ) along with hybrid error control mechanisms (ResTP-NACK and ResTP-NACK+MACK) are implemented. The protocols are simulated using ns–3 (network simulator-3) and are compared against the quasi-reliable mode to examine their tradeoffs. They are also compared against the traditional TCP and UDP protocols in an identical network scenario.

I like to dedicate this work to my parents, Krishna Mohan Pathapati and Manjula Rani Pathapati, and my brother, Venkata Bharat Vineeth Pathapati, for their constant support and encouragement throughout the process.

Acknowledgements

I would like to thank to my comittee members for their valuable insights and guidance. I would like to specially thank Dr. James P.G. Sterbenz for his guidance, support, and encouragement. I would also like to thank former PhD student Dr. Justin P. Rohrer for mentoring, advising, and collaborating with me on most of the design and implementation of the protocol. I would like to thank PhD student Egemen Cetinkaya for his valuable advise and insightful discussions during our group meetings. Finally, I would like to thank all the members of the ResiliNets reasearch for their constant help and support without which much of the accomplishment would not have been possible.

Contents

List of Figures

List of Tables

Page left intentionally blank.

Chapter 1

Introduction and Motivation

The Internet has become an indispensable part of today's society. Services that enable gathering and sharing information, performing business transactions, and social networking are provided by the Internet. However, when these services are disrupted due to the network's vulnerability to failures, the outcome can be severe. To provide acceptable level of service even during the face of disruption, the protocols that operate within the networks should provide a greater degree of resilience [1].

Protocols designed to meet the demands of present day users and applications face multiple challenges posed by the network environments they operate in. For example, in airborne networks disruptions are caused due to challenges such as episodic connectivity, mobility, delays, bandwidth constraints, corruption, and interference. Maintaining end-to-end connections in such challenged environments is often difficult and requires more robust design. At the transport layer this means that a connection must be able to adapt quickly to the changing network conditions. To some extent these challenges can be addressed by enabling cross-layering such that the different layers can communicate implicitly with each other to gain a better idea of the network conditions [2].

End-to-end reliability is provided by using sequence numbers, error detecting and correcting codes such as checksums and forward error correction (FEC) schemes respectively, and retransmitting the data packets that were unable to reach the destination [3]. This involves in dealing with two integral issues: implementing connection management paradigms and error control procedures. Providing reliability is a challenging task as it involves an overall increase in packet exchange overhead between the source and the destination, increased protocol processing at both ends, and state retention [4]. With the advent of wireless networks, transport protocols faced new challenges as most of the network characteristics changed. Unlike the wired networks, wireless networks consist of mobile nodes and unstable links causing episodic connectivity, packet losses due to corruption, delays due to handoffs and rerouting of packets, and network partitioning. Providing a full range of services efficiently in such an environment means making fundamental changes to the way protocols operated and in some cases making fundamental changes to the way the transport layer functions [5].

A portion of research in this area has been previously done by a number of students in the ResiliNets research group under the supervision of our advisor. Under the iNET (Integrated Network Enhanced Telemetry) program the group has developed a suite of protocols to address the issues posed by the current telemetry networks. The suite of protocols referred to as ANTP (Aeronautical Network and Transport Protocols) employ cross-layered architectural framework to enable efficient end-to-end communication [2,6]. The protocol suite consists of AeroTP that is a TCP-friendly transport protocol designed to provide both reliable and unreliable end-to-end communication depending on the type of data being transmitted by the nodes in the network. AeroNP is an IP-compatible network protocol that employs addressing and forwarding for the aeronautical environment and AeroRP is a routing protocol that offers several alternatives in absence of location information caused by short contact times between nodes [6–16]. The AeroTP protocol is a domain-specific subset of ResTP (Resilient Transport Protocol) and it employs many of the transport protocol features of ResTP such as multi-mode reliability structure. The difference between the two protocols lies in the design specificity. While ResTP is designed based on path diversification in a multi-path environment, the AeroTP protocol is designed for a single-path aeronautical environment.

This thesis will focus on the design and implementation of the ResTP protocol to address the end-to-end reliability issues associated with challenged networks. We will consider the implementation of various error control schemes and analyze their performance. The rest of this chapter is organized as follows: Section 1.1 presents an example of a disruption-tolerant environment, Section 1.2 briefly introduces the drawbacks of TCP and UDP protocols, Section 1.3 provides the problem statement and brief motivation, Section 1.4 presents solution for the thesis statement, and Section 1.5 lists the contribution of this thesis.

1.1 Highly Dynamic Airborne Network

In this section, to gain a perspective on the environments that these challenges might be present in, we give an example of a highly-dynamic airborne network as a challenged network.

The airborne network scenario introduces highly mobile nodes with intermittent connectivity and dynamic topology causing efficient end-to-end communication to be difficult. The network shown in Figure 1.1 consists of three types of nodes: Ground Stations (GS), Airborne Nodes (AN), and Relay Nodes (RN).

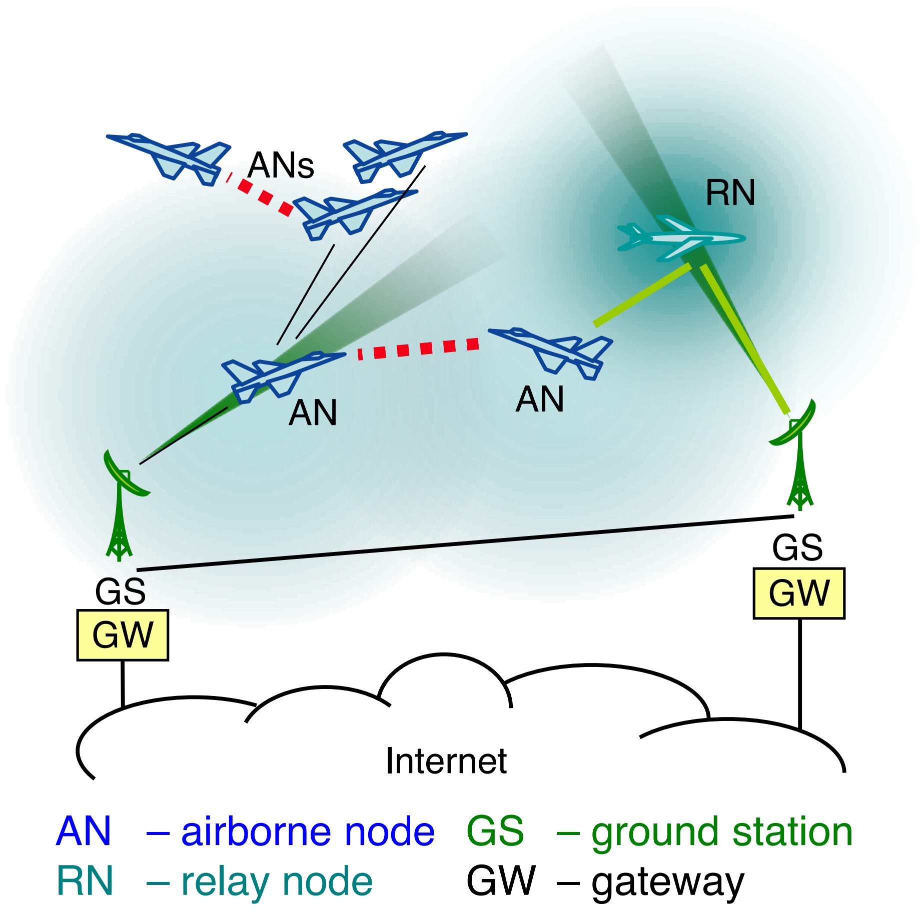


Figure 1.1. Test and evaluation environment [2, 8]

The GSs have gateways (GW) that provides an interface between the airborne network and internet applications running at the GS. ANs are airborne nodes, for example fighter jets, that contain devices running on IP such as cameras, monitors and other control equipment that collect data from other nodes and GSs. The airborne nodes move with velocities as high as Mach 3.5 causing time-varying connectivity between the nodes. The contact durations can be as short as 15 seconds at Mach 7 closing speed leading to network partitioning [8]. GSs have large steerable antennas with higher transmission range than a AN. They are capable of tracking ANs; however, due to narrow beam width GSs can track only one AN at a given time. They can also interact amongst each other during a soft handoff of a AN. The RNs are also airborne nodes that are used to provide better connectivity in the network, thus, they are assumed to have higher transmission range and larger buffer space than a AN to queue data. The primary data flow within the network is from a AN to GS, but the command and control data flows in the reverse direction.

The nature of this network environment poses many challenges to the current communication protocols. Mobility and dynamic topology are caused because of high speed nodes. Intermittent connectivity arises due to short contact durations, limited transmission range of ANs, interference, and jamming. Bandwidth constraints arise because of limited spectrum available for high volume of data that is sent from AN to GS.

1.2 Drawbacks of TCP and UDP Protocols

Transport layer protocols such as the transmission control protocol (TCP) and the user datagram protocol (UDP) are popularly used for communication in the Internet. In spite of their popularity and use in wide range of applications, they perform poorly in a wireless environment. The performance degradation is attributed to a number of drawbacks of both the TCP and UDP protocols. These drawbacks of TCP and UDP will be discussed in detail in Chapter 2.

1.3 Problem Statement

Popular transport protocols such as TCP and UDP exhibit degraded performance in challenged networks. They do not provide any adaptive measures to address the challenges posed by these networks. TCP employs a fixed error control scheme to address losses due to congestion without considering the current network conditions. It is also unable to discriminate corruption-based losses causing severe performance degradation in the face of challenges. UDP is unable to detect any errors. Both do not offer any QoS (quality of service).

Thesis Statement:

> To address the challenges posed by disruption-tolerant environments, transport protocols need to be adaptive and resilient to constantly changing network conditions. Providing various error control mechanisms to address end-to-end reliability issues depending on current network error conditions is important to enhance performance.

1.4 Proposed Solution

ResTP is a new transport protocol designed to address the challenges posed by challenged networks while being TCP-friendly to allow seamless splicing with conventional TCP at the network edge. The transport layer needs in these environments are dynamic resource sharing, differentiated level of precedence or QoS, real-time data service, best-effort connections, and best-effort datagrams [8]. Hence the protocol uses a multi-mode reliability architecture to provide QoS. It also employs multiple error control schemes depending on the nature of the application and error conditions prevailing in the network. The defined transport layer functions include connection-set up and management, transmission control, and error control [8]. Error control is performed using mechanisms such as ARQ (automatic repeat request) and FEC (forward error correction) at both the transport and the link layers. The protocol also implements hybrid-ARQ, both ARQ and FEC, using linear-block codes such as Reed-Solomon to improve performance in moderate error conditions. The hybrid-ARQ modes employ two different mechanisms: hybrid-ARQ using only NACKs (negative acknowledgments) and hybrid-ARQ using both NACKs and ACKs (positive acknowledgements). To support QoS, ResTP employs different operational modes: reliable connection mode, nearly-reliable connection mode, quasi-reliable connection mode, unreliable connection mode, and unreliable datagrams. The reliable connection mode performs end-to-end reliable data transfer at the transport layer using ARQ. The nearly-reliable connection uses a custody transfer mechanism to provide high reliability but does not fully guarantee delivery. The quasi-reliable connection provides statistical reliability using open-loop error recovery mechanism such as FEC coding. The unreliable connection relies on link-layer (FEC or ARQ) to preserve data integrity but does not perform any error correction at the transport layer. The unreliable datagram passes UDP traffic with no AeroTP connection management involved [6].

1.5 Contribution

This thesis provides and analyzes the implementation of ResTP reliable mode and ResTP hybrid-ARQ modes in ns-3. The reliable mode, as mentioned earlier, provides guaranteed data delivery. The mode must preserve end-to-end acknowledgment semantics from source to destination in order to perform guaranteed delivery [17]. The implementation is supported by results that show the performance of ResTP in terms of average goodput (PDR), average delay, total data delivered, and cumulative overhead.

As developing this protocol involved design from our previous work some of the implementation is performed by other members from the ResiliNets research group. To summarize, the major contributions of this thesis are:

- modification of ResTP's segment structure and state machine from the original design.
- implementation of ResTP connection management scheme
- implementation of ResTP reliable mode, ResTP-ARQ, with selective repeat ARQ mechanism in ns-3 [18]
- optimizing the selective-repeat ARQ algorithm by employing aggregated ACKs called MACKs (pronounced em-ACK).
- understanding the effect of aggregating ACKs on the performance ResTP
- implementation of type-I hybrid-ARQ modes: ResTP-NACK and ResTP-NACK+MACK
- examining the performance tradeoffs of a pure closed-loop end-to-end ARQ and a pure open-loop end-to-end FEC error control schemes with hybrid error control scheme that fills in the gap between the open-loop and closed-loop error control
- analyzing the benefits of end-to-end ARQ (ResTP-ARQ), hybrid-ARQ (ResTP-NACK and ResTP-NACK+MACK), end-to-end FEC (ResTP-FEC) and comparing them against each other and with traditional TCP and UDP protocols

Note:

> The quasi-reliable mode, ResTP-FEC, implementation was done by Justin P. Rohrer. He is also one of the principal contributors to the original design of the ResTP protocol [9].

1.6 Organization of Thesis

The thesis is organized as follows: Chapter 2 provides a review on background necessary for understanding and implementing this protocol. It discusses various ARQ mechanisms and applications in detail. Chapter 3 presents the design of ResTP and its transport layer functions such as connection management and its error control schemes. Chapter 4 describes the ns-3 implementation of ResTP and its error control schemes. Chapter 5 describes the simulations scenarios employed and provides an analysis on the performance of ResTP modes. It also compares their performance against TCP and UDP. Chapter 6 concludes the thesis by discussing what has been accomplished so far and future goals.

Page left intentionally blank.

Chapter 2

Background and Related Work

This chapter discusses some basic and necessary concepts related to transport protocols while introducing some of the earlier and popular protocols. One of the focus areas of this chapter is the concept of reliability: how it is achieved, its advantages, and the issues it imposes on the network. Reliability impacts other important concepts such as connection management and error control. Hence, the types of errors and error-handling mechanisms specifically involving ARQ algorithms such as stop-and-wait, go-back-N, fast retransmit, selective-repeat, SACK (selective acknowledgments), and SNACK (selective negative acknowledgments) are also discussed. The other focus area is understanding the protocols that implement reliability and the behavior of these protocols in different network environments.

2.1 Transport Protocols

The transport layer of the network architecture is responsible for delivering application data between end-system hosts through network switches or routers.

The services offered include selecting a data abstraction type such as byte streams, messages, and packets; reliability; flow control; error control; connection-oriented or connectionless; and congestion control services. Transport protocol designs primarily depend on the applications for which they are providing the service and the network environment in which they operate. Some of the examples of applications are client-server applications, request-reply, peer-to-peer file sharing, and multimedia streaming. Network environments broadly are classified into wired and wireless. However, regardless of the application a transport protocol is designed for or the environment in which it is going to operate, the goal is to achieve efficient and secure communication. Efficiency in transport protocols is often measured in terms of packet exchange overhead that affects the overall delay and throughput, complexity of algorithms implemented, and the size of the state space. In general, to improve the performance of transport protocols and to address issues unique to a particular environment, improvements to the existing protocol mechanisms are made through better implementation choices and eliminating or redistributing existing functionality [4].

2.1.1 End-to-End Reliability in Transport Protocols

One of the features that distinguishes transport protocols from one another is the provision of end-to-end reliability. A reliable transport protocol guarantees correct delivery of data to the intended destination. Introducing this feature changes the design of transport protocols in terms of error control, resource management, and a few other services. Reliability is achieved through messages called acknowledgments (ACKs), a defined connection management mechanism, and an efficient error control mechanism. ACKs are messages sent by the communicating pair to one another to notify that an expected packet has either arrived indicating a successful delivery (positive ACKs) or not arrived indicating a loss (negative ACKs). Connection management involves state retention that primarily involves allocation, synchronization, and deallocation of state between the source-destination pair [3, 4]. Error control involves the detection of and recovery from lost, damaged, missequenced, and duplicated packets. Error detection involves the use of error control identifiers such as sequence numbers to detect lost packets, while recovery involves either the use of error correcting codes (FEC codes) or retransmissions (ARQ). Flow control is another aspect of reliability that deals with controlling the rate at which the transport protocol sends the packets down to the network layer.

Reliability is a expensive mechanism to implement since it involves state retention, redundant information, data copying, buffer management, and packet exchange overhead [4]. In some cases using ARQ mechanism to achieve reliability gives rise to asymmetry, which means the channel is not fully utilized in the reverse path. The goal when designing an efficient transport protocol is to send as few number of packets as possible so that in extremely lossy conditions, losses do not cause excessive overhead and eventually degrade the throughput and delay performance of the protocol. While using any of the reliable mechanisms, connection identifiers such as sequence numbers are used to ensure ordered delivery and to make sure there is no duplication. While using sequence numbers, it is important to make sure for every communication session the identifiers are unique and have a bound on how long they can exist in the network.

Two of the earliest and the most commonly used protocols are TCP and UDP. TCP and UDP with modified or added functionality became the basis for many protocols to offer better services and perform efficiently in challenged environments, and are used as the basis for many of the developing transport protocols.

2.1.2 Transmission Control Protocol

TCP is a full-duplex, connection-oriented, end-to-end reliable protocol between hosts in a multi-host or multi-network environment that provides a byte-stream service. TCP was designed to operate for a wide range of communication systems with either hard-wired connections or packet-switched networks [19]. It offers its services to the application protocols that belong to the upper layer and it requires addressing, forwarding, and routing services from the lower layer Internet protocol (IP). To provide a standard communication service between two processes in a network, TCP was designed with multiple features. TCP's operation offers a reliable data service that allows the transmitted data to recover from damaged, lost, duplicated, or out-of-order TCP segments during the segment's transmission through the network. It achieves reliability by using cumulative ACKs from the receiver. It retransmits each segment if an ACK is not received in a set period of time called a timeout. It is a connection-oriented protocol that uses a 3-way handshake mechanism to explicitly establish a connection between two hosts and terminate it when the transmission is completed. It takes an extra RTT to set up the connection between the sender and the receiver after which actual data transmission begins. It implements flow control as an end-to-end mechanism that limits the amount of data transmitted by the sender at a given time to avoid choking the receiver. The TCP's congestion control mechanism prevents the sender from injecting too much data into the network causing congestion [20]. Congestion overloads the switches or routers in the network and causes the performance to degrade drastically.

2.1.3 User Datagram Protocol

UDP is a protocol with minimal end-to-end message delivery mechanism for the application programs [21]. UDP is unreliable which means there is no guarantee that the data sent will be delivered to the destination. It does not implement flow control, congestion control, or ordered delivery.

2.1.4 Drawbacks of TCP and UDP Protocols

Although TCP and UDP are the most commonly used transport protocols they fail to perform efficiently in a challenged wireless environment. In any network packet losses are inevitable, the reasons including link outages, lossy channel characteristics, unstable connectivity, delays, and congestion. A wireless channel is often subjected to bit-errors, interference, and channel fading, resulting in packet loss and packet corruption. TCP assumes every packet loss is caused by congestion in the network and invokes a congestion control algorithm. This decreases the congestion window by a fraction each time reducing the congestion window's size and thus, resulting in inefficient use of bandwidth. Schemes such as split-TCP connections and local retransmissions were developed to circumvent the problem caused by TCP's assumption of congestion being the only cause for packet loss [22].

TCP uses acknowledgment messages (ACK) to provide reliable data transmission and retransmissions. The source retransmits a TCP segment to the destination when a timeout occurs while waiting for an ACK. A connection setup is performed through a three-way handshake between the source and the destination pair of nodes. This takes up an extra round-trip time (RTT) that causes significant performance degradation in networks suffering from intermittent connectivity. By using a slow start algorithm, TCP may take RTTs to exit slow start before it can fully utilize the bandwidth.

TCP does not efficiently perform flow control in a network with asymmetric links since it requires a highly reliable ACK stream. Because of dynamic topology, link outages are common. The congestion control algorithm is invoked during short link outages causing an increase in number of retransmissions. The connection is terminated in case of longer link outages. This causes difficulty in restoring links and finding alternate paths to the destination [6]. TCP also does not provide any QoS for prioritizing the type of data being transmitted.

UDP is a simpler protocol than TCP and it does not offer any guarantee for guaranteed data delivery, so it is unreliable. Unlike TCP, UDP does not have a connection set-up mechanism and does not provide congestion control or flow control or data retransmissions. UDP also does not provide differentiated levels of precedence or QoS for the classes of data available in networks.

2.1.5 Optimizations for Mobile Wireless Networks

Researchers in the past laid much emphasis on developing algorithms for TCP to adapt to congestion issues in a network [23–25]. As networks evolved from simple point-to-point links to wired mesh networks to wireless and heterogenous (both wired and wireless) networks, high link error rates, channel fading, interference, long propagation delays, and noisy channel conditions increasingly became the reason for packet losses. Hence, decreasing the congestion window size in case a loss occurred in the network caused overall throughput degradation in the network.

Active research to find out other ways to deal with losses in a wireless network led to the development of newer algorithms [26]. TCP Peach [27] and TCP Westwood [28] are two such algorithms that were developed for wireless networks. TCP Peach was developed as a congestion control scheme for satellite IP networks. Satellite networks are often characterized by long propagation delays and high error rate channels. It was necessary that the algorithm could differentiate when the loss occurred due to congestion and when it did not. It introduced two new algorithms, sudden start and rapid recovery, along with traditional congestion avoidance and fast retransmit algorithms. TCP Peach performed better in terms of throughput and also provided an overall fair share of network resources compared to traditional TCP algorithms [27]. TCP Westwood was developed to improve TCP performance in both wired and wireless environments. TCP Westwood made use of end-to-end bandwidth estimation to discriminate the cause of packet loss in the network. It calculated the rate of connection continuously at the TCP sender side and computed the congestion window threshold and slow start threshold. It estimated the connection rate by monitoring the rate of returning ACKs [28]. The main advantage was that the only modifications made to TCP were at the source. Improvements in throughput and fair usage of link capacity were other advantages.

Other techniques were proposed to improve the performance of TCP in wireless environments [22]. The techniques are applied at different points in the network. The first technique involves a direct end-to-end protocol implementation where the sender is responsible for error recovery. The error recovery is performed using TCP SACK and explicit loss notification (ELN) mechanisms. The second technique provides link-layer reliability and the third technique implements a split-connection protocol, in which the end-to-end connection breaks at the base station. The results show providing local reliability at the link-layer that is TCP aware improves TCP's performance in wireless networks.

Wireless networks are characterized by asymmetric links in which conserving bandwidth in the lower capacity link becomes important. To address the effects of asymmetry on performance of TCP in a network, techniques such as ACK congestion control (ACC), ACK filtering (AF) used to control the frequency of ACKs, TCP sender adaptation (SA), ACK reconstruction (AR), and scheduling data and ACKs were developed to minimize the number of ACKs [22].

With the rapid increase in wireless technologies, high bandwidth-×-delay product networks are becoming increasingly common. These networks pose new challenges that worsen when the network is highly asymmetric. Asymmetry arises when there is a difference in the power used by the communicating entities for transmitting information. In these asymmetry can arise because of mismatched bandwidths. The central transmission unit, such as a base station, has a higher transmission power compared to the individual mobile units in order to reduce power consumption. With asymmetry in which there is bi-directional traffic flow, it becomes much more difficult to attain optimal performance.

The performance of the TCP protocol also was evaluated in a network with high bandwidth-×-delay product and random loss [29]. TCP showed deterioration in the throughput performance when random losses occurred and was unfair towards connections with larger RTTs while multiple connections share a bottleneck link.

2.1.6 Space Communications Protocol Standards

Space Communications Protocol Standards (SCPS-TP) is an extension to TCP, used particularly for satellite communications, developed to address problems posed by asymmetric links [30]. A satellite network experiences problems that are similar to a highly-dynamic airborne network. Both share problems arising due to limited spectrum, bandwidth constraints, resource constraints, asymmetric links, intermittent connectivity, and higher error/loss rates. The following paragraphs in this section describes the mechanisms used to cope with these issues.

The error control and recovery in the SCPS-TP protocol is based on TCP Vegas [24]. The enhancement on the part of SCPS-TP is that it identifies the source of loss, as opposed to TCP that automatically assumes congestion. It broadly classifies the sources of packet losses as being congestion-induced, corruption-induced, and losses due to link outage.

In the case of congestion, the protocol at the receiver end or at the next immediate router makes use of explicit congestion notification (ECN) bit in the packet header to notify the sender about any congestion related loss [30]. SCPS-TP then backs off using the congestion control algorithm. SCPS-TP also modifies the TCP Vegas implementation of congestion avoidance by doubling the TCP window size every other RTT as opposed to every RTT, achieving a more linear growth of the TCP window. In the case of corruption and link outage, the protocol sends corruption-experienced and link outage ICMP messages correspondingly to the destination to indicate existing corruption on the link or link outage. In the case of link outage, the sender protocol suspends the timers and does not transmit any new data. It only sends probe packets to check the status of the link until a link-restored ICMP message is received. Normal data transmission is resumed and data packets are sent from the point before which the link outage occurred.

In order to tackle the issues related to asymmetry arising from the volume of TCP acknowledgments generated in the reverse channel, the protocol uses a delayed ACK mechanism. The number of ACKs the receiver decides to delay is depending on the RTT estimation [30]. The protocol employs header compression and selective negative ACK (SNACK) option to work when the link capacity is constrained and limited.

In spite of the similarities in the issues that SCPS-TP addresses it is not suitable for applications in which network conditions change rapidly. This is because it relies on channel condition information which is either pre-configured or learnt during transition through the network [2].

2.2 Error Handling and Automatic Repeat Request

Early work on designing general-purpose protocols to provide end-to-end transport services was based on assumptions that no loss or corruption of packets would occur during an end-to-end transmission between hosts, and no missequenced packets or duplicate packets would arrive at the destination [4, 31]. However, in wireless networks these set of assumptions result in improper behavior. These networks are frequently prone to losses caused by channel characteristics, network topology, and traffic conditions.

Error control involves two phases: error detection and error recovery. Error detection involves identifying errors caused by corruption, loss, duplication, and missequencing of information. This is achieved through the use of checksums, FEC codes, and acknowledgements. Error recovery involves using retransmissions, FEC codes, or both to recover from the error. These mechanisms often come with drawbacks that affect the performance of the protocol. For example, while ARQ provides high reliability it results in low throughput in high BER (bit error rate) conditions. Moreover, while recovering losses using acknowledgments and retransmissions, it is important to consider the overall delay caused due to retransmissions. Using a purely-ARQ based error control scheme results in higher delay in high BER conditions. An alternative to using ARQ is using FEC codes to correct transmission errors. FEC codes involve adding extra bits to each data segment that are then used to correct errors at the receiver. Although FEC does not significantly increase overall delay and maintains a fairly high throughput compared to ARQ in high BER conditions, adding additional bits to every data segment increases the packet overhead [32–34]. Furthermore, using FEC codes alone provides only statistical reliability since it does not recover beyond the capability of the code, nor recover from losses or missing data segments. When errors go undetected the data segment is still delivered to the application and this makes employing a good FEC code expensive as it increases the complexity of the decoding system. Taking into account the drawbacks of using an error control based purely on ARQ or FEC, hybrid error control combines both ARQ and FEC to obtain better performance in moderate error conditions [35–37].

2.2.1 ARQ Algorithms

Ideally, reliable data transfer transmits data end-to-end with no delay and with no errors or losses. However, transmission in a network is often prone to delay, limited bandwidth and multiple errors along the path towards the destination. Bit errors are the most common in wireless channels because of the channel's vulnerability to noise and interference. Packet errors are caused because of congestion, switching between multiple paths within the network, and packet-drops during the occurrence of bit errors in the packet. To avoid the errors caused by congestion, congestion control and avoidance algorithms are used. They reduce the TCP window size by a fraction each time congestion is detected. Packet drops at the receiver may be caused because of corrupted packets. Error recovery schemes are often a solution to correct the errors in the received data packet. ARQ uses ACKs and retransmissions to ensure all the lost packets are successfully delivered to the destinations.

ARQ algorithms improved over time achieving better performance in terms of bandwidth utilization, delay, and reliability [38–40]. The basic approach in achieving reliability is through the stop-and-wait algorithm. The algorithm employs a feedback mechanism where the sender is notified of the delivery of the packet. In this approach, the sender transmits a packet to the receiver and waits for an ACK. In case the sender does not receive an ACK for a packet, which might be because of lossy link characteristics or packet-drops, the sender waits for a timer to expire, after which it retransmits the un-ACKed packet. This causes the link to be idle for the entire time the sender is waiting for an ACK causing inefficient utilization of available bandwidth and a delay of one RTT per packet. An alternative is the go-back-N algorithm. Multiple packets are sent simultaneously to the destination and the sender waits for all the ACKs. Once the sender misses an ACK for a single packet, retransmission of all the packets since the lost packet occurs. Although this eliminates the delay caused by waiting for an ACK, it introduces the delay caused by retransmitting packets since the loss. Fast retransmit is an algorithm in which the sender retransmits packets even before the timer expires. Retransmission occurs when the sender receives more than a certain number of duplicate ACKs [41].

2.2.2 Selective-repeat ARQ

An alternative to the fast retransmit algorithm is the Selective Repeat ARQ. In this algorithm, retransmission of only those packets for which the sender did not receive an ACK takes place. When the packets are retransmitted they arrive out of sequence. Hence, the receiver maintains a buffer to store the packets to rearrange them at the end of the entire session. The algorithm's complexity increases since both the sender and the receiver have to maintain a consistent state throughout the session and the receiver must have an increased buffer size. It also fails in the case multiple packet losses occur during transmission. Another version of Selective Repeat ARQ sends ACKs for a group of packets instead of a single packet each time. This reduces the overall complexity at the receiver buffer space and corrects the behavior of the protocol during multiple packet losses. SNACK (selective negative acknowledgment) is an alternative to SACK in which the receiver sends negative ACKs requesting a damaged or lost packet. The receiver explicitly notifies the sender which packets were lost or corrupted and thus may need to be retransmitted. TCP SNACK was originally implemented in satellite communications in which the end-to-end delay was long [30]. SNACK provides the sender with a complete view of receiver's buffer when the sender receives an ACK specifying damaged or lost packets. The sender aggressively sends packets that are lost without waiting for a timeout. In this case TCP's congestion control is not invoked and utilization of bandwidth is improved.

2.3 Estimating Retransmission Timeout

In the previous section we discussed how retransmissions can be used to recover lost or corrupted TPDUs (transport protocol data unit). Retransmissions are triggered by either detecting a missing ACK or by relying only on timeouts. It is understood that the timeout must be set larger than the RTT to prevent unnecessary retransmissions. However, determining how long the timeout should accurately be is important [20, 42]. Since, networks can be prone to various delays, RTT is seldom constant. Hence, updating the RTT information can help determine what the RTO (retransmission timeout) must be set to. In this thesis we follow RFC 2988 specifications in estimating RTO. For estimating RTT it is important to collect a sample RTT for every ACK received. Hence, RTT samples, *sampleRTT*, are recorded when an ACK arrives carrying a sequence number for a corresponding TPDU. The sampleRTT varies for every TPDU and hence an average of these samples are taken, denoted as *estRTT* calculated using the following:

$$\text{estRTT} = (1 - \alpha) \times \text{estRTT} + \alpha \times \text{sampleRTT}$$

The deviation in RTT is also necessary to estimate the RTO and it is denoted as *devRTT*, calculated as:

$$\text{devRTT} = (1 - \beta) \times \text{devRTT} + \beta \times |\text{sampleRTT} - \text{estRTT}|$$

In the above equations recommended $\alpha = 0.125$ and $\beta = 0.25$ [42]. From the above equations the timeout is calculated as:

$$\text{RTO} = \text{estRTT} + 4 \times \text{devRTT}$$

It is important to note that RTT is estimated only for outstanding data segments (TPDUs have not been retransmitted) unless the protocol header provides an option to record a timestamp for every transmitted TPDU [42].

2.4 Hybrid Error Control

Hybrid error control schemes, commonly referred as hybrid-ARQ, involve FEC within an ARQ scheme. The idea is to use FEC error recovery capability to reduce the number of retransmissions. This addresses the performance drawback of ARQ in high BER conditions. In the case that the FEC is unable to recover any errors, the receiver requests retransmission using negative acknowledgments instead of delivering the corrupted data segment to the receiver. This improves the reliability of the scheme compared to using only using FEC codes [33]. There are two types of hybrid ARQ schemes: type-I and type-II [32, 35, 43, 44]. Type-I hybrid-ARQ scheme is an approach that is designed for simultaneous error detection and correction. When a received data segment is detected with errors, the redundant information within that segment attempts to correct them. Upon successfully correcting the errors the segment is delivered to the application. In case the correcting code is unable to correct the errors, the receiver discards the segment and requests a retransmission. The received retransmitted segment undergoes the same error check and decoding process. When unsuccessful, retransmission is requested and this repeats until the destination receives a segment that can be successfully decoded. Type-I schemes are suited for channels with a relatively constant error rate. When an efficient error correcting code is chosen, this approach can reduce the number of retransmissions significantly and improve the overall delay and performance. This scheme comes with drawbacks, especially at the extremes of channel error conditions. During fairly low error rate the addition of extra bits increases the overhead, while during high error conditions the error code word used may not be sufficient to correct all the errors, thereby increasing the numbers of retransmissions and reducing the throughput [32]. Type-II hybrid-ARQ schemes are suited for channels with varying error rates [36]. These schemes often use adaptive hybrid-ARQ approach in which the extra bits are added depending on the channel conditions. The retransmission strategy involved in using this approach has varied from only retransmitting the extra bits to correct the corrupted original segment to retransmitting the entire segment [45]. In this thesis we explore type-I hybrid-ARQ schemes using a Reed-Solomon linear block code.

2.5 Summary

This chapter presented the general transport layer functions and end-to-end reliability concepts. It also described the challenges employing reliability mechanisms pose to the network. The popular transport protocols TCP and UDP were reviewed along with their drawbacks that contributes to degraded performance. This chapter then presented some of the TCP optimizations developed for general wireless ad hoc networks and more specifically in satellite communications using SCPS-TP. This chapter also discussed how error control can be achieved and the types of error control mechanisms used. In the next chapter we will talk about the design of ResTP and the transport layer functions it supports.

Chapter 3

Design of ResTP

This chapter presents the design of the ResTP protocol and discusses its operation in the reliable mode and hybrid-ARQ modes. The design aspects of the ResTP protocol include the header formats, connection management paradigms, and the error control strategies. Following the design features of the protocol in the reliable-mode and hybrid-ARQ modes, we consider the differences between the error control schemes. The chapter is organized as follows: Section 3.1 discusses briefly the overview of ResTP protocol and its five operational modes, Section 3.2 introduces the header formats for both ResTP data TPDU and MACK TPDU, Section 3.3 explains the connection management phases of ResTP protocol along with the state machine employed. Finally we consider various error control schemes used in this protocol and their differences in Section 3.4.

3.1 ResTP Overview

ResTP is a resilient transport protocol designed to address the challenges posed by a disruption-tolerant network. The AeroTP protocol previously developed as a domain-specific subset of ResTP, is a part of the ANTP suite of protocols that were developed specifically to address the issues posed by highly-dynamic airborne networks [6]. ResTP offers differentiated levels of precedence or QoS depending on the type of data being communicated, thus involves several operational modes. Transport layer functions that must be performed by the protocol include connection management, transmission control, and error control. The connection management involves setting up connections, and terminating them by synchronizing the sender and receiver using a defined state machine. The protocol uses ARQ and FEC for error control and it is fully decoupled from rate control [2].

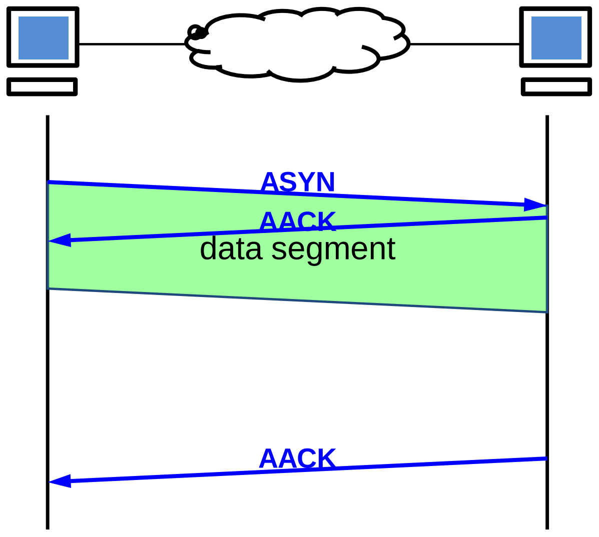


Figure 3.1. ResTP reliable connection mode

There are four operational modes: reliable connection, quasi-reliable connection, unreliable connection, and unreliable datagram [6, 8].

- ***Reliable connection*** mode uses end-to-end acknowledgment semantics from source to destination to guarantee correct data delivery as shown in Figure 3.1. Hybrid-ARQ enhances reliable connection mode to reduce the number of retransmissions needed.

- ***Quasi-reliable connection*** mode completely eliminates ACKs and ARQ. It uses open-loop error recovery mechanisms such as FEC and erasure coding to achieve statistical reliability.

- ***Unreliable connection*** mode uses no error correction mechanism at the transport layer. It relies exclusively on the FEC of the link layer to preserve data correctness.

- ***Unreliable datagram*** mode is a stateless mode which provides no assurance of reliable delivery.

3.2 ResTP Segment Structure

A ResTP segment (shown in Figure 3.2 and Figure 3.3) is capable of converting to the TCP/UDP format at the realm boundaries in the cases of interworking with TCP/IP. The ResTP protocol maintains two different segment structures depending on whether the segment is a TPDU or an ACK or a MACK. The MACK segment structure is very similar to the data segment structure except in the place of payload there is a variable 32-bit field that carries the ACK number. Both the segment structures are in conjunction with the formats specified in our previous work [2].

- ***Source Port Number***: 16 bits

  The source port number (unsigned int) of the ResTP header is used by the receiver to identify the correct flow to which the ResTP segment belongs.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Source Port Number        |     Destination Port Number   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|resv |  Mode   |ECN|   Flags   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         MACK Length           |        Optional fields        /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/ Optional fields for FEC, Erasure Coding, ... /  TP HEC CRC-16 /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                            Payload                            /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/             Payload FEC Parity Trailer (Optional)             /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Payload CRC-32 (Optional)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.2. ResTP data segment structure

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Source Port Number        |     Destination Port Number   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence (ACK) Number 0                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|resv |  Mode   |ECN|   Flags   |         Payload Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         MACK Length           |        Optional fields        /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/ Optional fields for FEC, Erasure Coding, ... /  TP HEC CRC-16 /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          ACK Number 1                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          ACK Number ...                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          ACK Number N                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
/             Payload FEC Parity Trailer (Optional)             /
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Payload CRC-32 (Optional)                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.3. ResTP MACK segment structure

- ***Destination Port Number***: 16 bits

  The destination port number (unsigned int) of the ResTP header is used by the sender to identify the correct flow to which the ResTP segment needs to be delivered.

- ***Sequence Number***: 32 bits

  The sequence number (unsigned int) uniquely identifies ResTP segments for reordering them at the receiving edge and for error-control purposes.

- ***Timestamp***: 32 bits

  The timestamp (unsigned int) records the time of the latest ResTP segment transmitted.

- ***Reserved***: 3 bits

  These bits are reserved for future use.

- ***Mode***: 5 bits

  The mode bits indicate the ResTP reliability mode currently in use. The modes are: Reliable, Quasi-Reliable, Unreliable Connection, and Unreliable Datagram modes.

- ***Flags***: 8 bits

  The flag bit (unsigned int) indicates the type of ResTP segment transmitted. Some of the bits are the same as the TCP flag bits to enable translation of TCP segments. The flag bit used in the ResTP protocol are:

  - ASYN: indicates the ResTP connection setup request
  - ACK: indicates a single ACK for an ResTP segment
  - NACK: indicates a single NACK for an ResTP segment
  - MACK: indicates multiple ACKs for multiple ResTP segments

  - PSH: used to pass TCP PSH flag transparently
  - RST: used in ResTP for future use
  - URG: used to pass TCP URG flag transparently
  - AFIN: indicates the ResTP connection termination request

- ***Payload Length***: 16 bits

  The payload length (unsigned int) indicates the length of each payload unit or data that is sent by the application.

- ***MACK Length***: 16 bits

  The MACK length (unsigned int) indicates the number of ResTP ACKs contained in the ResTP MACK header.

- ***Optional Fields***: variable length

  The optional field is associated with the mode that the ResTP protocol operates in. For example, in a quasi-reliable mode which uses FEC bits for error recovery the optional field indicates the FEC strength used (FEC strength field). Hence, in case the payload gets corrupted, ResTP performs FEC on the payload. The HEC (header error check) field performs a strong CRC on the header to detect bit errors caused by wireless channel, thus making sure the packet is correctly transmitted to the destination.

- ***Payload***: Variable length

  The payload carries the application data.

- ***ACK Number***: Variable length

  The ACK number (unsigned int) represents the sequence number of a correctly received data segment.

- ***Payload CRC-32***: 32 bits

  The payload CRC (unsigned int) for error correction depending on the transfer mode used.

3.3 ResTP Connection Management

For a reliable-transport protocol, it is essential to define and maintain consistent states at the sender and the receiver to establish a connection for data transfer. The states either remain the same or evolve to another depending on the events and actions that happen within the protocol during a communication session [46]. Figure 3.4 shows the ResTP reliable mode packet flow-diagram, in which S is the source, D is the destination. The diagram depicts the basic operation of the reliable mode. A more detailed end-to-end connection setup phase can be seen in Figure 3.5.

The ResTP protocol is connection-oriented in all the operational modes except for the unreliable datagram mode. The ResTP connection management scheme consists of a connection setup or connection establishment phase and a connection termination phase. The protocol uses control messages such as ASYN, AFIN, ASYN_ACK, and AFIN_ACK to setup and terminate connections. To maintain uniformity and reduce complexity, all the modes that are connection-oriented use an identical state machine. Figure 3.6 shows the ResTP state transition diagram and Table 3.1 contains description for various states and events in the ResTP protocol.

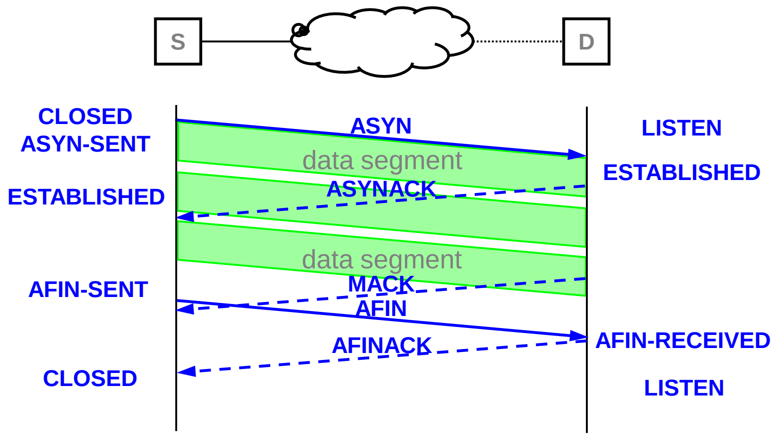


Figure 3.4. ResTP connection management

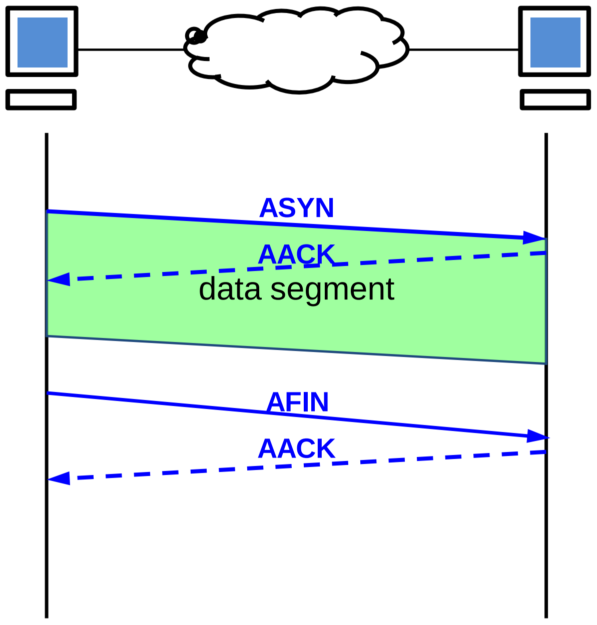


Figure 3.5. ResTP connection setup

3.3.1 Connection Establishment

As shown in Figure 3.5, the connection establishment phase in the ResTP protocol is similar to that of TCP's connection establishment. ResTP uses an opportunistic connection-establishment mechanism in which data and control overlap.

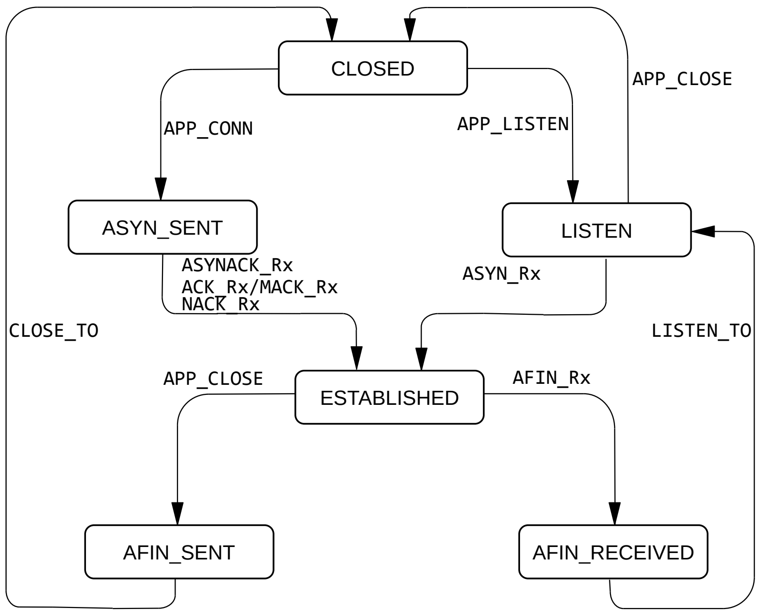


Figure 3.6. ResTP state transition diagram

This differs from the TCP three-way handshake, which requires an extra RTT. An ResTP connection is initiated by the sender upon receiving an APP_CONNECT message from the application while in the CLOSED state. The sender initiates connection by sending an ASYN message. This involves setting the ASYN bit in the ResTP header. The setup message is then transmitted with or without data depending on the data being available in the send buffer. The sender moves to the ASYN_SENT state. The receiver moves to the LISTEN state upon receiving an APP_LISTEN message from the application. Upon receiving the connection setup message the receiver sends an ACK for the ASYN by setting both the ASYN and the ACK bits in the ResTP header. The receiver moves to the ESTABLISHED state. In a wireless network with multiple paths to a destination a segment can easily arrive out of order. Hence, the sender considers a successful connection establishment upon receiving an ACK to any ResTP TPDU instead of only waiting for an ASYN_ACK. However, the application can also chose to wait for an

Table 3.1. State transition definitions

State	**Description**
CLOSED	No connection exists and no TPDU is transferred
LISTEN	Destination is ready to listen to any incoming TPDU
ASYN_SENT	ASYN sent by the host initiating connection
ESTABLISHED	Steady state in which data transfer takes places
AFIN_SENT	AFIN sent indicating all outstanding TPDU have been sent
AFIN_RECEIVED	AFIN received; AFINACK sent acknowledging AFIN
Event	**Description**
APP_CONN	Request issued to the sender to initiate connection
APP_LISTEN	Request issued to the destination to LISTEN to incoming data
APP_CLOSE	Request to intiate closing a connection by sending AFIN
ASYN_RX	ASYN received, indicating a connection is requested
ASYNACK_RX	ASYNACK received, indicating connection request is granted
ACK_RX	Single ACK received
MACK_RX	Multiple ACKs received
NACK_RX	NACK received, indicating corrupted TPDU; requesting retx.
AFIN_RX	AFIN received; connection termination is initiated
AFINACK_RX	AFINACK received; connection termination has been notified
CLOSE_TO	Timer ensuring all data is transmitted before going to CLOSED
LISTEN_TO	Timer ensuring all data is received before going to CLOSED

ASYN_ACK to ensure a successful connection establishment. The sender moves to the ESTABLISHED state. In this state both sender and the receiver exchange TPDUs and ACKs.

3.3.2 Connection Termination

The ResTP protocol makes use of explicit messages, AFIN and AFIN_ACK, to ensure a successful termination. When all outstanding data has been transmitted the application issues an APP_CLOSE message. The sender then sets an AFIN bit in the ResTP header and transmits an AFIN message without any payload. The sender then moves to the AFIN_SENT state and starts a timer, CLOSE_TO, to move the CLOSED state. The timer ensures that all data has been acknowledged before closing the connection. On the other end, upon receiving the AFIN message, the receiver sends an AFIN_ACK by setting AFIN and ACK bits. The receiver also starts a timer, LISTEN_TO, to move to the LISTEN state once all data has been acknowledged [31, 46, 47].

3.4 ResTP Error Control Schemes

In this section we discuss the various end-to-end error control schemes of the ResTP protocol implemented in ns-3. The ResTP protocol in the connection oriented mode uses a pure ARQ based error control scheme, a pure FEC based error control scheme, and two type-I hybrid ARQ error control schemes that use both ARQ and FEC to provide reliability. The differences between these schemes are briefly mentioned below.

- ***ResTP-ARQ***: A pure ARQ based mode using selective-repeat ARQ to provide complete reliability. There are two retransmission strategies employed in this mode, one that uses only RTOs (retransmission timeouts) to trigger retransmission and the other that uses a fast recovery technique based on missing ACKs. The protocol in this mode also features aggregating ACKs in which multiple ACKs are aggregated to conserve bandwidth in the reverse channel. The protocol also enables the user to control the number of retransmission per packet.

- ***ResTP-FEC***: A pure FEC based mode that uses Reed-Solomon error correcting code to provide statistical reliability.

  NOTE: This mode was originally developed and implemented by Justin P. Rohrer, a former PhD student of the ResiliNets research group. It is a part of our previous work involving design and simulation of AeroTP protocol [6,8,10,11]. It is used in this thesis to integrate with the ARQ code to implement hybrid-ARQ modes.

- ***ResTP-NACK***: A hyrbid-ARQ mode that uses only NACKs to indicate the receipt of a corrupt packet and to request retransmissions.

- ***ResTP-NACK+MACK***: A hyrbid-ARQ mode that uses both ACKs and NACKs to indicate the receipt of successfully received packets, the receipt of corrupted packets, and to request retransmissions.

Depending on the error control scheme chosen, the protocol offers various embedded features. For example, a default case in the ResTP-ARQ mode involves acknowledging each data segment, which means sending an ACK for each data packet. To decrease the amount of packet exchange overhead and in order to conserve bandwidth in the reverse channel, the protocol aggregates multiple ACKs [48, 49]. The protocol is able to determine that an incoming acknowledgment is a single ACK or a multiple ACK depending on the flag that is set in the ResTP header. The mode also offers two retransmission strategies; one based on only retransmission timeout and the other based on fast recovery. We consider this difference further in this section.

To better understand the behavior of these mechanisms, we make use of packet flow diagrams. The following cases show the behavior of each of the above discussed schemes when a TPDU is either lost, corrupted, or both. In this thesis, a corrupted TPDU is referred as a TPDU that has been delivered but detected with errors that are beyond correction and a lost TPDU is referred to as a TPDU that has been lost during transmission never arriving at the receiver.

3.4.1 Comparison of Pure-ARQ vs. Hybrid-ARQ Modes

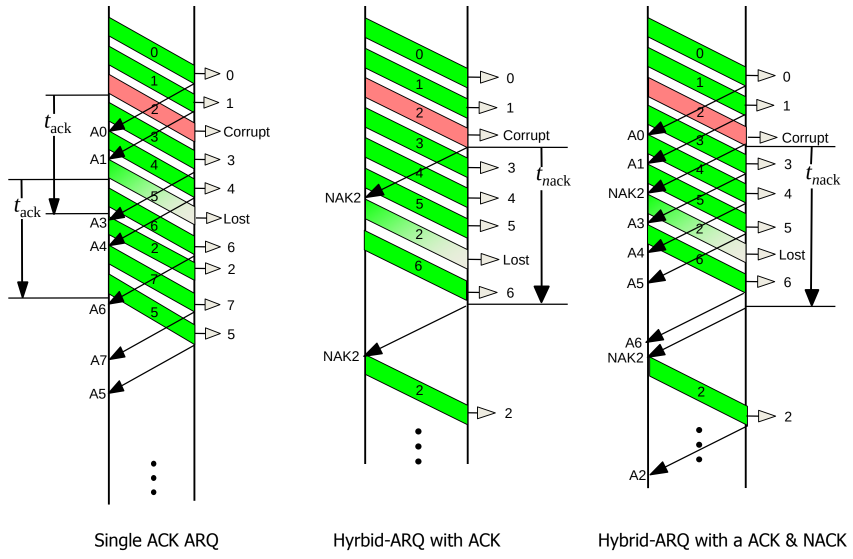


Figure 3.7. Difference between ResTP modes

In this case we compare the behavior of three schemes in the event of a corrupted and a lost ResTP TPDU. This case provides a base to understand the primary difference between these three schemes. We consider specific cases in each error control scheme to better understand the modularity of the design. Figure 3.7 shows a pure selective-repeat ARQ based ResTP-ARQ mode with the single ACK acknowledgment strategy, a hybrid-ARQ mode with only NACKs, and a hybrid-ARQ mode with both ACKs and NACKs. The receiver in a pure-ARQ mode sends an ACK for every successfully delivered TPDU. This implies that ACKs are not sent in the case of lost or corrupted TPDUs. Recovery in this case is done by retransmitting the affected TPDU. In the hyrbid-ARQ modes, an FEC check is performed on every TPDU that arrives at the receiver. If the

TPDU is corrupted beyond the FEC's capability to correct as determined by the CRC, the TPDU is discarded and a NACK is sent requesting retransmission of the affected TPDU. This strategy applies for both the hybrid-ARQ modes. The difference is that lost TPDUs cannot be recovered using only NACKs as seen in the case of a lost TPDU, for example TPDU 5 in Figure 3.7. In hyrbid-ARQ mode that uses both NACKs and ACKs the lost TPDU is recovered in a way similar to that of a pure ARQ based mode.

NOTE: The packet flow diagrams used in this thesis are based on EECS 780 lecture notes [50].

3.4.2 ResTP-ARQ Mode

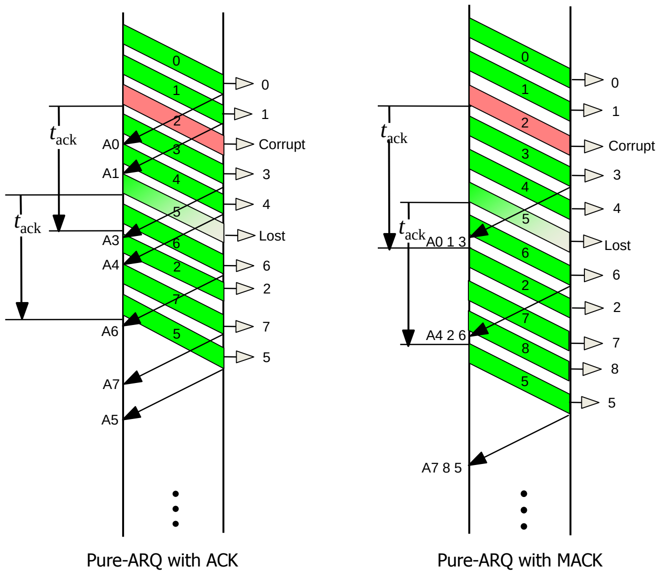


Figure 3.8. ResTP-ARQ using timeouts

Figure 3.8 shows the operation of ResTP-ARQ mode using only timeout to retransmit packets. Here the corrupted or lost TPDUs are only retransmitted when an RTO expires. Every transmitted TPDU is scheduled for a retransmission after time t_{ack}. This timer is canceled in case an ACK, for example A0, arrives indicating that the TPDU 0 has been correctly delivered. TPDU 2 and 5 do not receive ACKs indicating that they have not been delivered correctly. Hence, they are retransmitted after time t_{ack} and the timer is reset. This does not affect the flow of the other TPDUs. On the receiver end, TPDUs that arrive out of order are buffered until the next expected TPDU arrives. In the pure-ARQ with MACK case, the receiver waits to aggregate certain number of ACKs before sending it. The number of ACKs to aggregate is a user-defined parameter. If any TPDU is lost or corrupted, the receiver continues to wait for the next TPDU to arrive until a MACK is transmitted, for example A013 indicates an MACK carrying ACKs for TPDUs 0,1,3. The sender scans the sequence numbers to note which TPDUs have been ACKed and cancels the timer on the corresponding TPDUs. The sender waits for TPDU 2 to be ACKed and after t_{ack} the sender retransmits the TPDU.

Figure 3.9 shows the operation of ResTP-ARQ mode using fast recovery. In this example the sender schedules a retransmission for every TPDU like in the previous example. However, when a TPDU is not ACKed, for example TPDU 2, the sender immediately retransmits it without waiting for t_{ack} to timeout. This implies that the sender has to keep track of the next expected ACK which in this case after A0, A1 is A2 and after A3, A4 is A5. The expected ACK must updated every time an ACK is received to ensure that the sender does not unnecessarily retransmit the TPDUs.

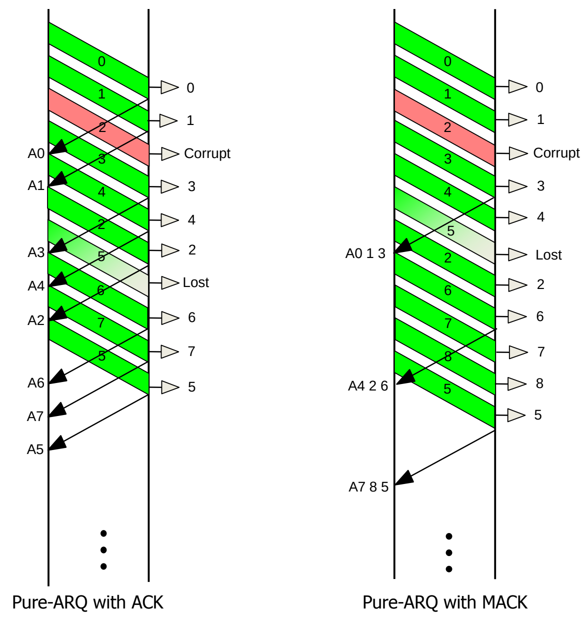


Figure 3.9. ResTP-ARQ using fast recovery

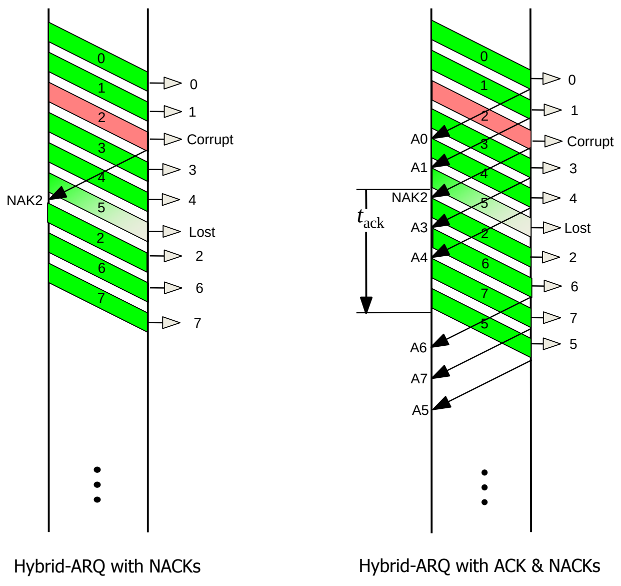


Figure 3.10. Comparison of hybrid-ARQ modes

3.4.3 ResTP Hybrid-ARQ Modes

In this section we consider the behavior of hybrid-ARQ modes with only NACKs and with both ACKs and NACKs. In Figure 3.10, the sender in ResTP-NACK mode assumes that a packet is correctly delivered if it does not receive any NACK. For example, when TPDU 0 arrives at the receiver, the receiver performs an FEC check on the TPDU. If the TPDU is error free the receiver delivers it to the application. The receiver does not send an ACK indicating that the TPDU has been delivered correctly. If a TPDU is corrupted, for example TPDU 2, and the number of errors detected is greater than the FEC's ability to correct them, the TPDU is dropped and a NACK is transmitted. In this case NACK 2 is transmitted requesting retransmission of TPDU 2. When the sender receives NACK 2, TPDU 2 is retransmitted. While this approach recovers corrupted TPDUs it fails to recover any lost TPDUs. However, when a retransmitted TPDU is lost, it can be recovered up to a certain number of retries. For example, in Figure 3.11, when TPDU 2 is lost after retransmission the receiver retransmits NACK 2 after time t_{nack}. However, in this thesis we limit the number of NACK retries to 5. The behavior of ResTP-NACK+MACK mode is similar to that of the ResTP-NACK mode while dealing with corrupted TPDUs. However, since the mode also employs ACKs to acknowledge correctly delivered TPDUs and maintains RTOs, lost TPDUs can be recovered (Figure 3.10). This has an advantage over ResTP-NACK in terms of achieving higher reliability.

3.5 Summary

In this chapter we presented the design of ResTP. The chapter discussed a brief overview of the protocol and its development in our previous work. The

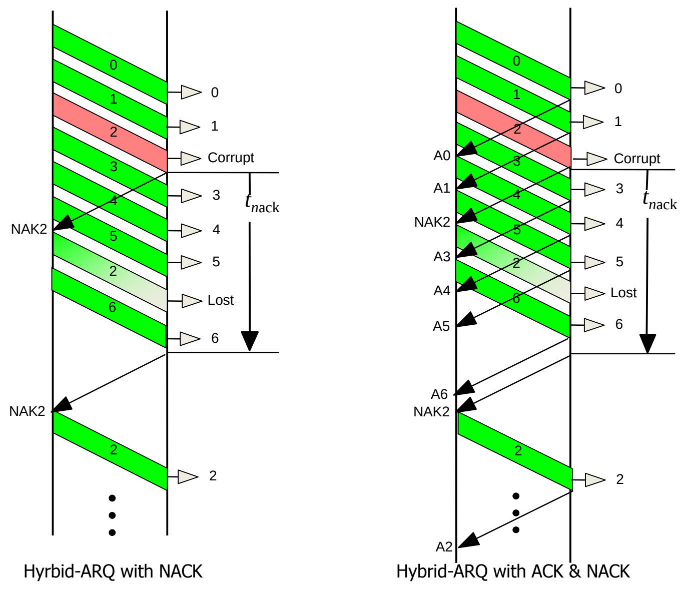


Figure 3.11. Comparison of hybrid-ARQ modes

ResTP segment structures for both TPDUs, ACKs, and MACKs were discussed in detail. We also discussed the connection management strategy used by the protocol which involved both connection setup and connection termination. We discussed the various error control schemes implemented in this protocol and their behavior in the case of corrupted and lost TPDUs.

Chapter 4

Implementation of ResTP

Following the design features of the protocol in the reliable-mode and hybrid-ARQ modes from the previous chapter, we describe the implementation of the protocol in ns-3. The details are discussed using a class interaction diagram to understand various classes used during implementation. The chapter is organized as follows: Section 4 introduces the building blocks of the transport layer in ns-3 and how various classes involved in the implementation interact, Section 4.1.1 discusses the modularity of the ResTP code. ResTP provides multiple error control schemes and features unique to each. This enables us to choose a combination of mechanisms for different scenarios. Section 4.2 presents the algorithm and flow of ResTP error control schemes using transmit and receive for both the sender and the receiver.

4.1 ResTP Implementation in ns-3

Implementing ResTP in ns-3 primarily involved understanding and learning how the application, transport, and routing layer were structured in it. From a transport layer perspective it meant building sockets that communicated directly with the application. Much of the protocol is implemented within this class. The ns3::Socket is an API that deals with creating sockets, performing sending and receiving operations, and initiating and terminating a connection with a remote host [18]. Although the sockets perform sending and receiving operations, sockets do not directly send to the routing layer. There is a layer in between sockets and the IP, ns3::Ipv4L4Protocol, which allocates endpoints objects and performs header checksum operations on the packets. For example, the packets are only forwarded to the endpoints depending on the checksum results.

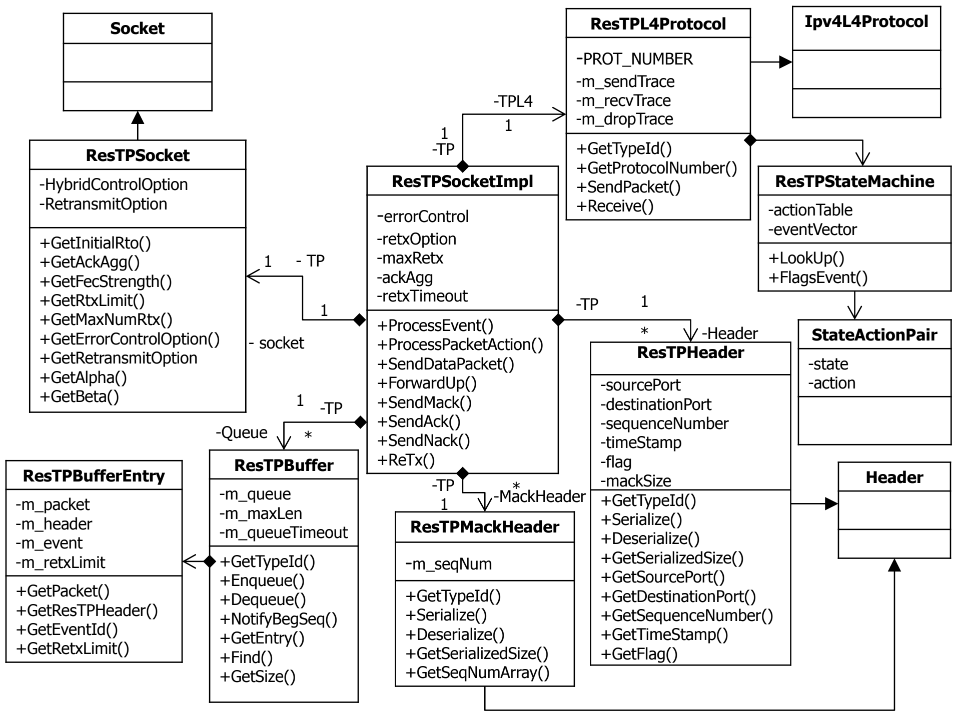


Figure 4.1. ResTP class interaction diagram

With the majority of the functional placement determined, the rest of the implementation involved creating classes for headers, buffers, and state machines. Figure 4.1 shows a class interaction diagram that briefly describes the attributes and methods involved in each class of the protocol. A detailed explanation of various classes used in implementing the ResTP protocol is presented in the following list.

- **ns3::ResTPSocketImpl**: A class that contains majority of the implementation of the protocol. This class implements the socket specific operations such as initiating and terminating a connection and receive and deliver application data. The class is also responsible for building ResTP data headers and MACK headers. It is derived from the ns3::ResTPsocket class that in turn is derived from the ns3::socket class. This class also implements the state machine that is declared in ns3::ResTPStateMachine and defined in ns3::ResTPL4Protocol. The ResTP reliable-connection mode, ResTP-ARQ, and the ResTP hybrid-ARQ modes, ResTP-Nack and ResTP Nack+Mack, are implemented within this class.

- **ns3::ResTPL4Protocol**: This class implements socket independent logic. It is responsible for adding the ResTP header to the payload and initializing the header checksum. This class is responsible for determining if the received segment is intended for the right endpoint which is stored in a class called the ns3::Ipv4EndPointDemux. This is done by storing a tuple that contains the local port, local address, destination port, and the destination address. The class also maintains a lookup table for state-action pairs and flag value to determine the event.

- **ns3::ResTPSocket**: This is a parent class of ns3::ResTPSocketImpl. This class contains the ResTP socket attributes that can be assigned and changed over various simulation scenarios. This class defines the various error control options and retransmission strategies. The user can initialize the attributes to control the behavior of the protocol.

- **ns3::ResTPHeader**: This class implements the ResTP header discussed in Chapter 3.
- **ns3::ResTPMackHeader**: This class implements ResTP Mack header. The MACK header is a variable length header that is treated as a vector.
- **ns3::ResTPBuffer**: This is a buffer class that implements any buffer instance used in ResTP. It primarily enables the endpoints to perform enqueue and dequeue operations.
- **ns3::ResTPBufferEntry**: This class is used to create an entry in a buffer. It is indexed by sequence number of the TPDU. It also holds the ADU (application data unit), ResTP header, an event id, and the maximum number of retries for ARQ.
- **ns3::ResTPStateMachine**: This class is used to create a state machine using a event vector and state-action pair.

4.1.1 Modularity of the ResTP Protocol

As described in previous chapters, ResTP is an adaptive transport protocol that offers resilience by offering multiple reliability modes. Multiple error control schemes offer a good platform for studying the tradeoffs of each protocol in various challenge conditions. Before further describing ResTP operation, we consider various options ResTP has to offer, as shown in Table 4.1.

Table 4.1. Modularity in ResTP protocol

Features	Options
Error control scheme	ResTP-ARQ, ResTP-Nack, ResTP-Nack+Mack
Retransmission strategy	fast recovery, timeouts
ACK aggregation	single ACK, MACK
Retransmission limit	no limit, maximum number of retransmissions

4.2 ResTP Operation

The basic ResTP protocol operation can be divided into transmit events and receive events. Transmit events are those associated with explicitly sending an outstanding TPDU. Receive events are events associated with receiving of a TPDU or an ACK. In this implementation, the sender implements an identical transmit event regardless of the error control scheme chosen. However, the sender has different receive events depending on if ACKs, NACKs, or MACKs are received. The receiver implements different receive events based on the error control scheme chosen. It does not perform any transmit event in the state machine although it transmits an ACK. The following sections describe the flow of protocol operation.

4.2.1 Transmit Event

Figure 4.2 shows the transmit event of the protocol. The transmit event here is explained along with the a few states, events, and actions as seen in the state table. Initially the sender and the receiver are in the CLOSED state. A connection is initiated by the sender through an APP_CONNECT message from the application. The sender populates the ResTP header based on the current state and the error control scheme chosen. For example, in the case of ResTP-ARQ mode, the header will not carry any FEC strength information and in the hybrid-ARQ modes the header carries FEC strength information. Once the header is added to the payload (ADU in this case), the sender creates a buffer entry using the ns3::ResTPBufferEntry class. The entry stores a copy of the original TPDU and is then added to the transmit buffer by using the ResTPBuffer::Enqueue() function. A retransmission is scheduled to that entry after time *t*. Once an entry has been created the protocol then sends the original TPDU to ns3::ResTPL4Protocol in which the header checksum is added. The TPDU is then transmitted to the IP layer and subsequently lower layers. The ns3::ResTPSocket continues to receive ADUs from the application until no more data is available. The application then issues an APP_CLOSE message.

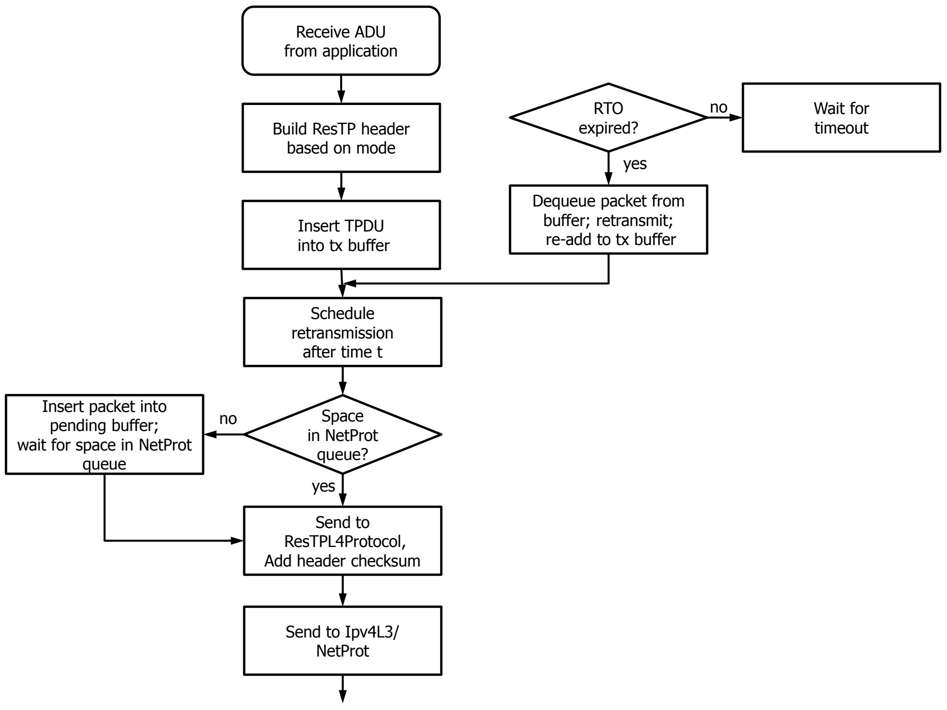

Receive ADU from application
Build ResTP header based on mode
Insert TPDU into tx buffer
RTO expired?
no
Wait for timeout
yes
Dequeue packet from buffer; retransmit; re-add to tx buffer
Schedule retransmission after time t
Insert packet into pending buffer; wait for space in NetProt queue
no
Space in NetProt queue?
yes
Send to ResTPL4Protocol, Add header checksum
Send to Ipv4L3/ NetProt

Figure 4.2. ResTP sender: Transmit event

4.2.2 Sender: Receive Event in ResTP-ARQ Mode

Figure 4.3 shows the sender's course of action when an ACK or an MACK is received. In case of a receive event, the sender processes the header to obtain the flag bit. The flag indicates if the received TPDU is an ACK or an MACK. In case of an ACK, the sender checks the sequence numbers contained in the header to determine if the ACK is expected. If the ACK is not the expected ACK, the sender then dequeues the corresponding TPDU from the transmit buffer and retransmits it. It then inserts the TPDU to the transmit buffer and schedules another retransmission. If the ACK is an expected ACK, then the sender cancels the retransmit event of the corresponding TPDU and dequeues it.

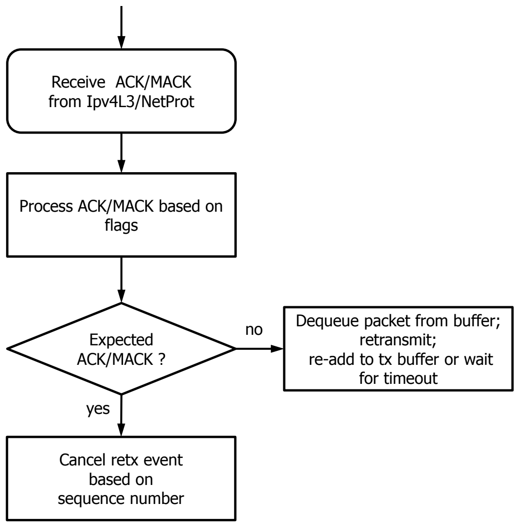

Receive ACK/MACK from Ipv4L3/NetProt
Process ACK/MACK based on flags
Expected ACK/MACK ?
no
Dequeue packet from buffer; retransmit; re-add to tx buffer or wait for timeout
yes
Cancel retx event based on sequence number

Figure 4.3. ResTP sender: Receive event for MACK/ACK

4.2.3 Sender: Receive Event in ResTP-NACK Mode

Figure 4.4 shows sender's behavior when a NACK is received. In the case of a NACK, the sender checks the sequence numbers contained in the header to determine which TPDU has been corrupted. It then then dequeues the corresponding TPDU from the transmit buffer and retransmits it. It then adds the TPDU to the transmit buffer and schedules another retransmission.

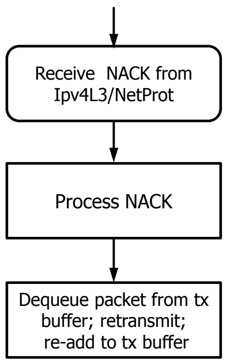

Receive NACK from Ipv4L3/NetProt
Process NACK
Dequeue packet from tx buffer; retransmit; re-add to tx buffer

Figure 4.4. ResTP sender: Receive event for NACK

4.2.4 Sender: Receive Event in ResTP-NACK+MACK Mode

When the hybrid-ARQ error control scheme involving both ACKs and NACKs is chosen, the sender checks for the flag bit in the header to determine if the received TPDU is an ACK, MACK, or a NACK as shown in Figure 4.5. If the flag indicates that the received segment is a NACK, then it functions similar to the ResTP-NACK receive event. In the case of an ACK or a MACK the sender functions similarly to the ResTP-ARQ receive event.

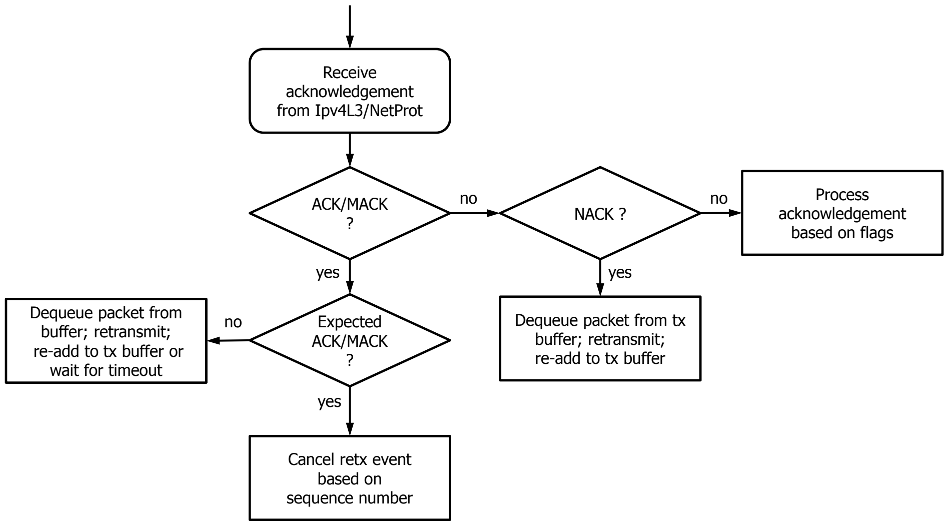
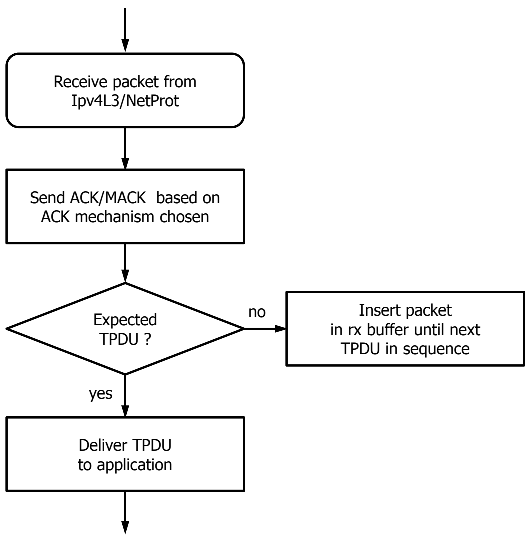

Receive acknowledgement from Ipv4L3/NetProt
ACK/MACK ?
no
NACK ?
no
Process acknowledgement based on flags
yes
yes
Dequeue packet from buffer; retransmit; re-add to tx buffer or wait for timeout
no
Expected ACK/MACK ?
Dequeue packet from tx buffer; retransmit; re-add to tx buffer
yes
Cancel retx event based on sequence number

Figure 4.5. ResTP sender: Receive event for MACK+NACK

4.2.5 Receiver: Receive Event in ResTP-ARQ Mode

Figure 4.6 shows the receiver end, which upon receiving a TPDU, sends either a single ACK or aggregates multiple ACKs based on the m_AckAgg value. If the value is set to zero then an ACK is sent for every correctly received TPDU, otherwise the receiver waits to aggregate the specified number of ACKs and transmits the acknowledgment. It then verifies if the sequence number in the received TPDU is in sequence to determine whether to deliver to the application or to wait for the next TPDU in sequence to arrive. While waiting the receiver inserts the TPDU in the receive buffer until the next TPDU in sequence arrives.

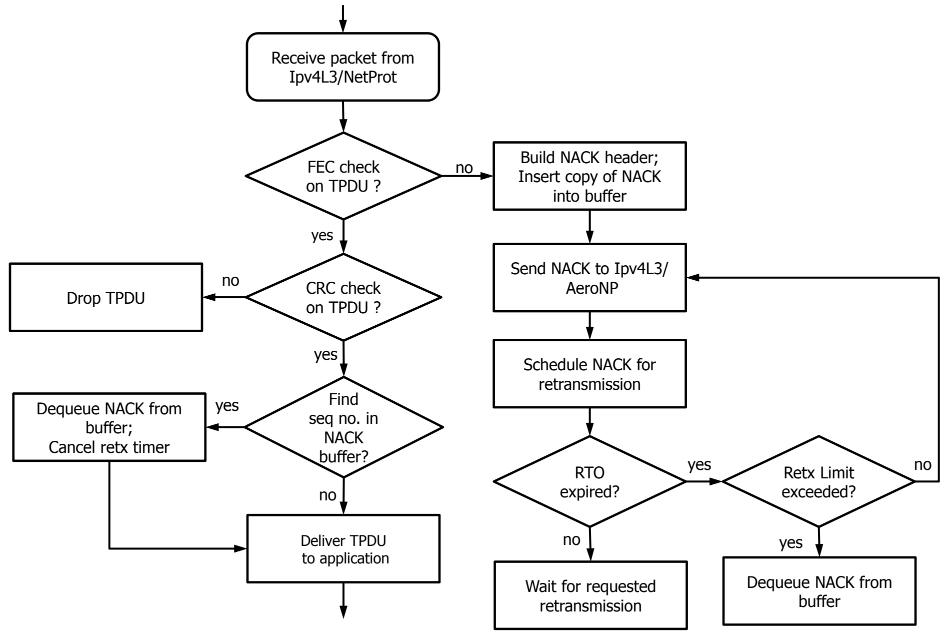


Receive packet from Ipv4L3/NetProt
Send ACK/MACK based on ACK mechanism chosen
Expected TPDU ?
no
Insert packet in rx buffer until next TPDU in sequence
yes
Deliver TPDU to application

Figure 4.6. ResTP receiver: MACK/ACK

4.2.6 Receiver: Receive Event in ResTP-NACK Mode

Figure 4.7 shows the receiver's behavior upon receiving a TPDU in ResTP-NACK mode. When a TPDU arrives, the receiver performs an FEC check on the TPDU to determine if the TPDU is correctly received. If the TPDU is corrupted then the receiver sends a NACK to the sender requesting retransmission of the original TPDU. The receiver maintains a NACK buffer to store the NACKs for retransmission. It then schedules the NACK for retransmission in case the original TPDU gets lost or the NACK gets lost during transmission. If the received TPDU is not corrupted then the receiver performs a CRC check on the TPDU. If the check fails the TPDU is dropped and cannot be recovered, otherwise the receiver checks the NACK buffer to verify if the received TPDU is a retransmitted TPDU or an outstanding TPDU. If it is a retransmitted TPDU, the receiver cancels the retransmission timer of the corresponding NACK and dequeues it. It then delivers the TPDU to the application.

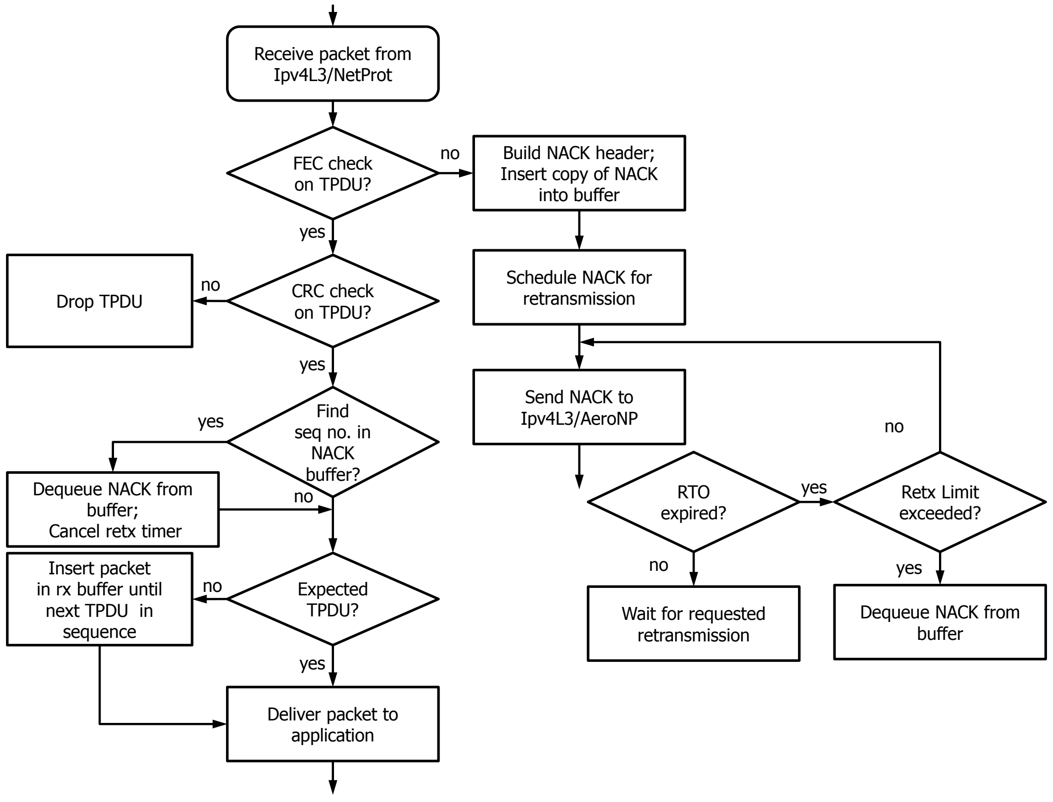


Receive packet from Ipv4L3/NetProt
FEC check on TPDU ?
no
Build NACK header; Insert copy of NACK into buffer
yes
Drop TPDU
no
CRC check on TPDU ?
Send NACK to Ipv4L3/ AeroNP
yes
Schedule NACK for retransmission
Dequeue NACK from buffer; Cancel retx timer
yes
Find seq no. in NACK buffer?
RTO expired?
yes
Retx Limit exceeded?
no
no
no
yes
Deliver TPDU to application
Wait for requested retransmission
Dequeue NACK from buffer

Figure 4.7. ResTP receiver: NACK

4.2.7 Receiver: Receive Event in ResTP-NACK+MACK Mode

Figure 4.8 shows the receiver's behavior upon receiving a TPDU in ResTP-NACK+MACK mode. When a TPDU arrives the receiver performs an FEC check on the TPDU to determine if the TPDU is correctly received. If the TPDU is corrupted then the receiver sends a NACK to the sender requesting retransmission of the original TPDU. The receiver maintains a NACK buffer to store the NACKs for retransmission. It then schedules the NACK for retransmission in case the original TPDU gets lost or the NACK gets lost during transmission. If the retransmission timer on the NACK expires, the receiver checks the retransmission limit. If the limit exceeds the maximum, which defaults to 5, the receiver dequeues the NACK from the buffer.



Receive packet from Ipv4L3/NetProt
FEC check on TPDU?
no
Build NACK header; Insert copy of NACK into buffer
yes
Schedule NACK for retransmission
Drop TPDU
no
CRC check on TPDU?
yes
Send NACK to Ipv4L3/AeroNP
yes
Find seq no. in NACK buffer?
no
Dequeue NACK from buffer; Cancel retx timer
no
RTO expired?
yes
Retx Limit exceeded?
Insert packet in rx buffer until next TPDU in sequence
no
Expected TPDU?
no
yes
Wait for requested retransmission
Dequeue NACK from buffer
yes
Deliver packet to application

Figure 4.8. ResTP receiver: MACK+NACK

Otherwise the NACK is retransmitted and the retransmission limit is updated to one less than the previous value. If the received TPDU is not corrupted then the receiver performs a CRC check on the TPDU. If the check fails the TPDU is dropped. The dropped TPDU is recovered through retransmission after the sender side timer on the TPDU expires or when the sender detects a missing sequence number in the ACKs. In case the checksum is correct, the receiver checks the NACK buffer to verify if the received TPDU is a retransmitted TPDU or an outstanding TPDU. If it is a retransmitted TPDU then the receiver cancels the retransmission timer of the corresponding NACK and dequeues it. Then the receiver checks to see if the TPDU in sequence. When in sequence it delivers the TPDU to the application, otherwise it stores the TPDU in the receive buffer and waits until the next TPDU in sequence arrives.

4.3 Summary

In this chapter we have discussed the implementation of ResTP protocol in ns-3. We showed the ns-3 classes used in implementing and developing this protocol. To gain a better understanding on the operation of ResTP we explained in terms of transmit and receive events for both the sender and receiver side. Finally, we have discussed in detail the operation of ResTP for all three error control schemes.

Page left intentionally blank.

Chapter 5

Simulations

This chapter presents the simulation scenarios used to test various modes of the ResTP protocol and compare them against each other. The simulations in this thesis are performed using open source network simulator-3 (ns-3) [18]. ns-3 is a discrete-event network simulator used to model, test, analyze, and simulate protocols over various network scenarios. Section 5.1 explains the various performance metrics used in the analysis of the transport protocols. Section 5.2 discusses the simulation scenario used to provide some baseline analysis.

5.1 Performance Metrics

We evaluate the performance of ResTP-ARQ, ResTP-NACK, ResTP-NACK+MACK, ResTP-FEC, TCP, and UDP protocols using the following metrics.

- ***Total data delivered***: Total number of received bytes that shows how much of actual unique data packets have been received in order. This metric is important to measure the total amount of application data delivered by a protocol while the network conditions might cause the connection between the source-destination pair to be terminated.

- ***Average delay***: This metric indicates the average time taken by all the packets to reach the destination. This is an important parameter to study the effects of using the selective-repeat ARQ mechanism and delayed ACKs on the overall delay performance. Error recovery mechanism using an ARQ mechanism affect delay because of retransmissions.

- ***Average goodput***: The average goodput shows the average rate of successful delivery of the application data. It is the rate of application data transmitted over time. The metric shows efficiency of the protocol.

- ***Cumulative Overhead***: The cumulative overhead shows the total number of extra bytes it takes for the protocol to deliver data. This includes all the extra bytes in the header, retransmitted packets, acknowledgments, connection set-up messages, and any extra bytes added for error correction. This is used to understand the efficiency of protocol operation in terms of contention for wireless channel and congestion and its impact on lower layers.

5.2 Simulation Model

The simulation scenario used in thesis shows only one of many challenge conditions for which ResTP is designed. We take an example of a highly-dynamic airborne network mentioned in Chapter 1. The network in this simulation setup consists of two nodes communicating via a lossy link. One node is configured as a traffic generator, sending data at a constant data rate of 4.416 Mb/s (3000 pkts/s with an MTU of 1500 B), and the other as a traffic sink. The path consists of a

10 Gb/s link representing the LAN on end-system, a 5 Mb/s capacity link with a latency of 10 s representing the challenged network, and a second 10 Gb/s link representing the LAN on the other end-system. Bit errors are introduced in the middle link with a fixed probability for each run, and the performance for each probability of bit-errors is shown in the plots described in the next section. A total of 1 MB of data is transmitted during a single simulation between the two nodes. The link is made unreliable by introducing losses using an error model varying bit-error probabilities ranging from 0 to 10^{-4} for each of the protocols. Each simulation case is run 25 times and the results averaged to obtain the data needed for transport layer comparisons with 95 percent confidence-interval bars plotted. The general parameter space used in the simulations are shown in Table 5.1.

Table 5.1. Simulation parameters

Parameter	Value
Transport Protocol	ResTP, TCP, UDP
ResTP-Mode	ARQ, FEC, NACK, NACK+MACK
Retransmit Option	Fast recovery, Timeouts
ACK aggregation	0–365
Limit retransmissions	(boolean) 0, 1
ACK retransmit limit	5
NACK retransmit limit	5
α (EMWA smoothing factor)	0.125
β (RTT deviation smoothing factor)	0.25
Data Rate	4.416 Mbps
MTU	1500 B
Bandwidth	10 Gb/s LAN, 5 Mb/s constrained network
Delay on bottleneck link	10s
Maximum Packet Lifetime (MPL)	30 s
Bit Error Rate (BER)	0 to 0.0001
Total data transmitted	1 MB
Number of runs	25
Simulation Time	2000 s
FEC strength	0–255
TCP timeout	60 s

5.3 Performance Analysis

In this section we will analyze the performance of each ResTP mode by varying parameters that are uniquely tunable in that mode. This will enable us to understand the behavior of each mode under different BER conditions. Furthermore, we compare the performance of ResTP with TCP-NewReno and UDP protocols.

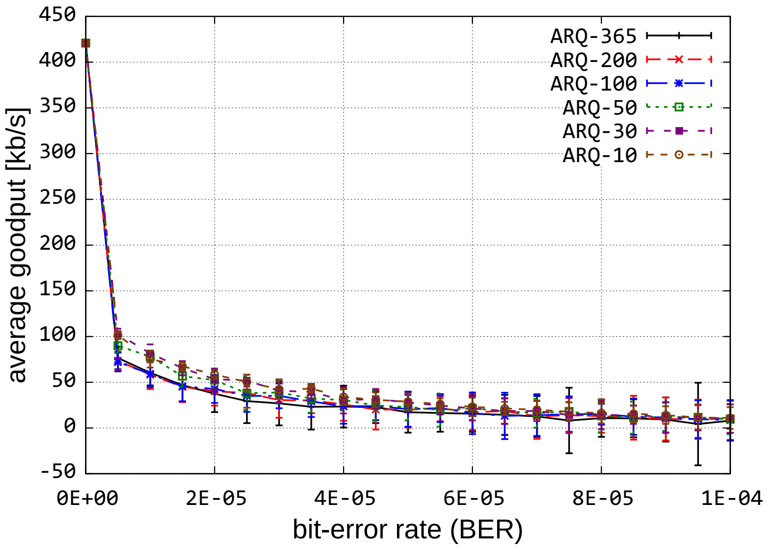


Figure 5.1. Fully reliable: Average goodput

5.3.1 Fully-Reliable Mode Performance

In the *fully-reliable* mode, ResTP uses ARQ to provide complete reliability. This trades off additional latency (in the case of lost or corrupted packets) and overhead of the reverse channel, against reliability. The advantage of this mechanism is that given enough time, every lost packet can be correctly delivered to the application.

The first set of simulations in this section show the effects of BER on ResTP-ARQ using MACKs. As discussed in Chapter 3, the ResTP reliable mode provides

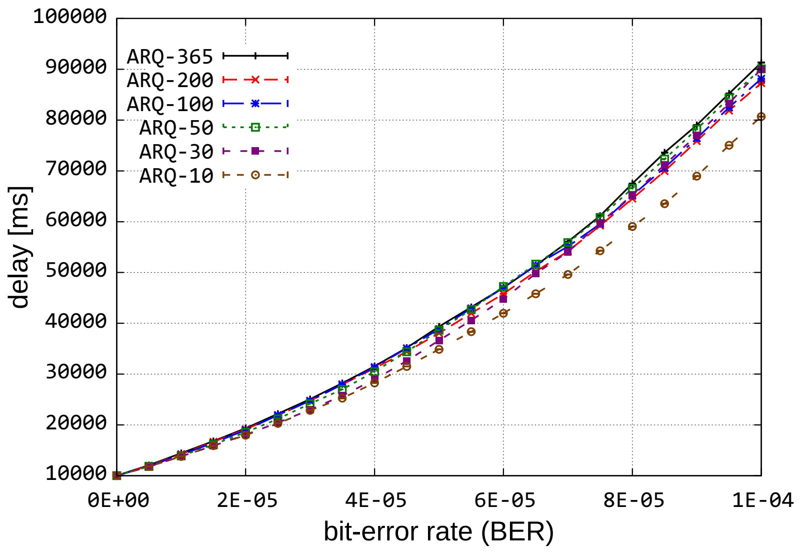


Figure 5.2. Fully reliable: Average delay

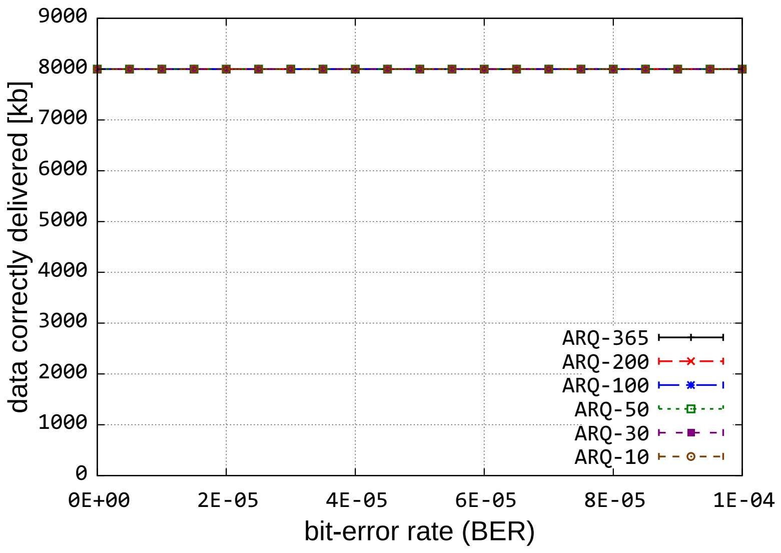


Figure 5.3. Fully reliable: Total data delivered

an option to aggregate multiple ACKs. With an MTU size of 1500 bytes we can aggregate up to 365 ACKs, in which each additional ACK adds 4 bytes to the 20-byte ResTP header. The simulation parameters specific to ResTP-ARQ mode set for this simulation are shown in Table 5.2. In this mode, the default retransmission

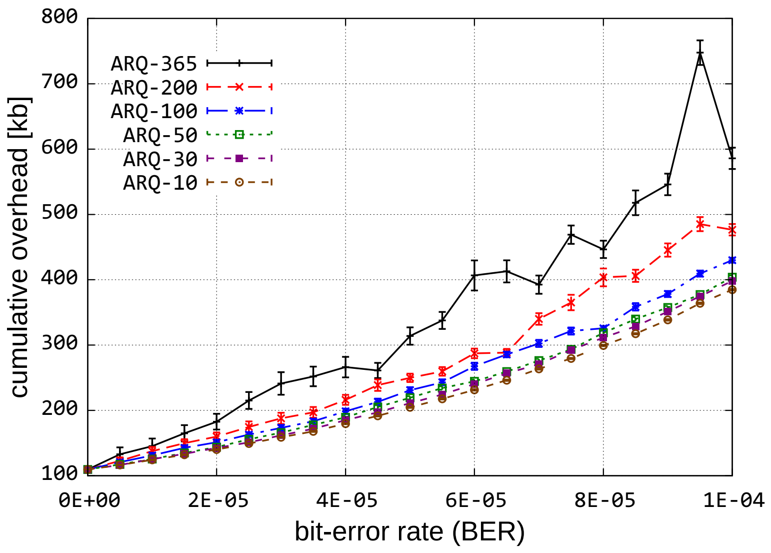


Figure 5.4. Fully reliable: Cumulative overhead

mechanism is set as timeouts.

Table 5.2. ResTP-ARQ with MACK enabled

Parameter	**Value**
Transport Protocol	ResTP
ResTP-Mode	ARQ using MACK
Retransmit Option	Timeouts
ACK aggregation	10, 30, 50, 100, 200, 365
Limit retransmissions	0

Figure 5.1 shows decreasing goodput as BER increases due to fewer data packets delivered per unit time. The steep decrease in the goodput initially is due to the large delay on the channel. The channel delay is set to 10 s and all of the data initially is sent within 1 s of the actual data transmission. As mentioned previously, this simulation scenario was taken as an example of a highly-dynamic airborne network for which AeroRP is used as a routing protocol. The store-and-forward delay for the AeroRP protocol was measured to be 10 s [13]. Figure 5.2 shows that the average delay in general increases across the range of BER due to an

increased number of retransmissions. However, there is no significant difference in the goodput or delay performance amongst different MACK values. We attribute this behavior to the amount of data transmitted being too small compared to the delay across the channel to see any significant difference in the performance. Figure 5.3 shows that all 1 MB of the data is delivered as expected, thus achieving full reliability, which is expected in closed-loop error recovery. Figure 5.4 shows increased overhead for ACKs aggregated above 200 due to the increased amount of retransmissions for every MACK lost. In the next set of plots, we examine

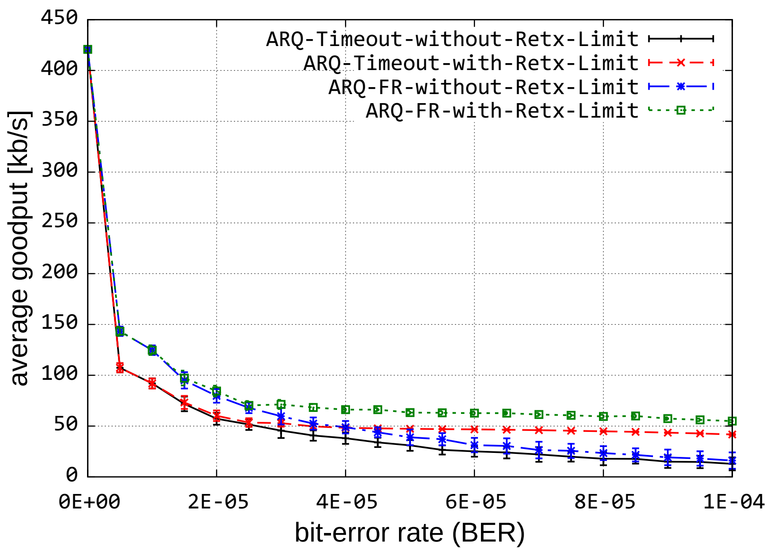


Figure 5.5. ResTP-ARQ: Average goodput

the effect of BER on the ResTP-ARQ mode while using both fast recovery and timeouts as the retransmission mechanism. We also test the performance of the protocol when a retransmission limit is set as opposed to when there are no limits on the number of retransmissions. Table 5.3 shows the parameters in this scenario.

Figure 5.5 shows that setting a limit on the number of retransmissions achieves better goodput using both fast recovery and timeouts. While during lower BER

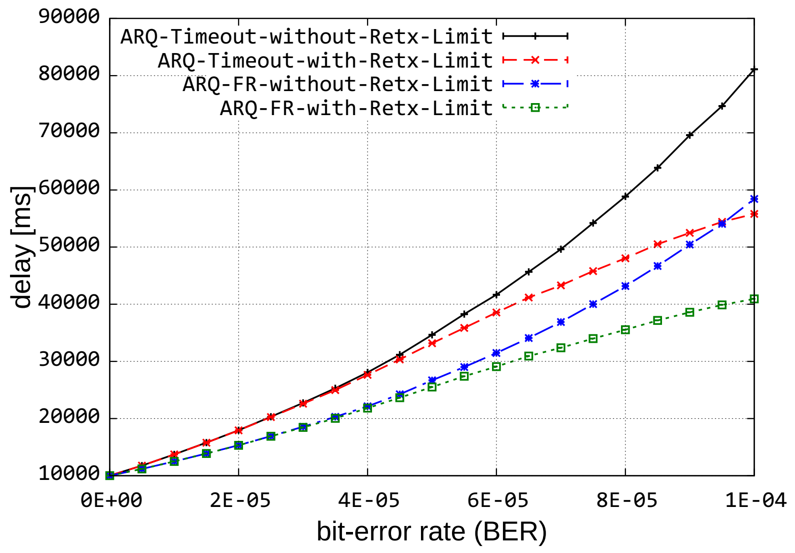


Figure 5.6. ResTP-ARQ: Average delay

Table 5.3. ResTP-ARQ using retransmissions

Parameter	**Value**
Transport Protocol	ResTP
ResTP-Mode	ARQ
Retransmit Option	Timeouts, fast recovery
ACK aggregation	0
Limit retransmissions	1
ACK retransmission limit	5
Link delay	10 s

retransmissions using fast recovery and timeout perform similarly, at higher BER using fast recovery achieves better goodput. This is because the the sender does not have to wait for retransmission timeouts to expire to send the missing data. Figure 5.6 shows that when TPDUs are retransmitted only based on RTOs the delay incurred is much higher. However, by limiting the number of retransmissions they can achieve lower delay at higher BER compared to fast recovery mechanism. Figure 5.7 shows that all the variations achieve full reliability at lower BER. For BER $> 6 \times 10^{-5}$ full reliability cannot be achieved with limiting the number of

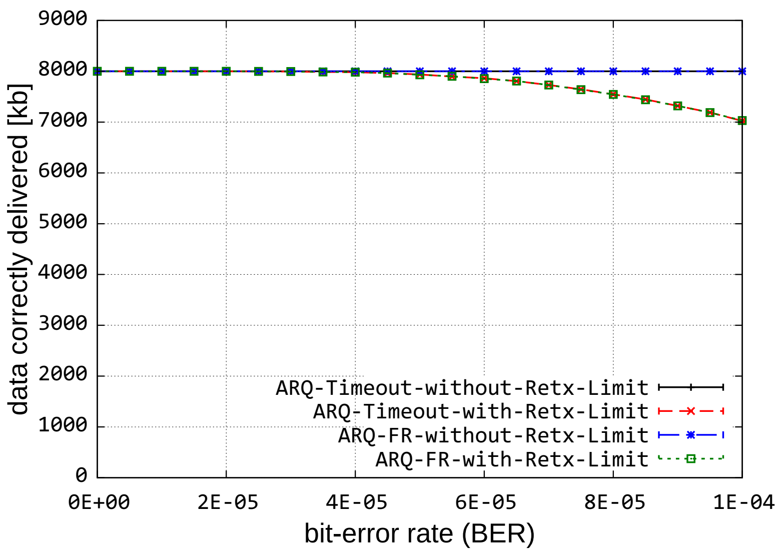


Figure 5.7. ResTP-ARQ: Total data delivered

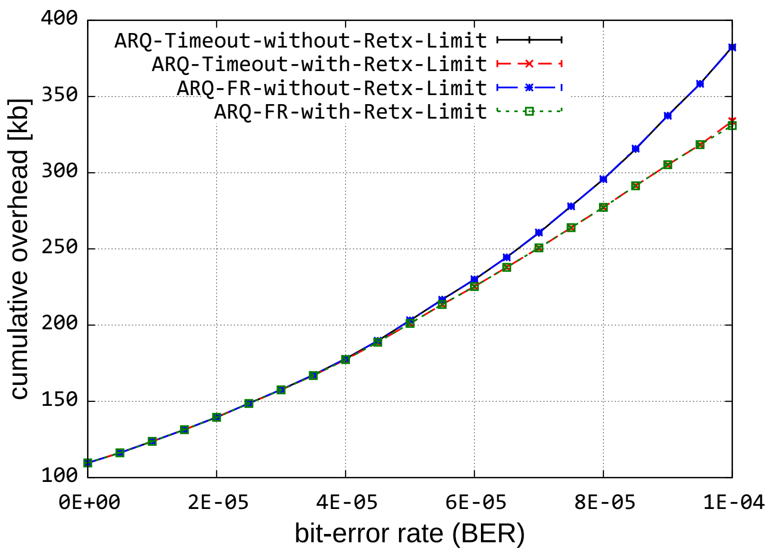


Figure 5.8. ResTP-ARQ: Cumulative overhead

retransmissions. This is because once the protocol exceeds a certain number of retries it closes the connection, thus not delivering outstanding TPDUs. While not setting a retransmission limit offers complete reliability the overhead incurred is also greater as seen in Figure 5.8. This is due to the increased amount of data transfer through retransmissions.

5.3.2 Hybrid-ARQ Mode Performance

In the hybrid-ARQ mode ResTP-ARQ uses both closed-loop and open-loop error control to recover losses. This trades off additional latency and overhead against higher end-to-end reliability. When compared to a pure open-loop error recovery, such as ResTP-FEC mode, using hybrid error control can achieve higher end-to-end reliability. In the ResTP-NACK+MACK mode, the protocol achieves full reliability and still gains better goodput than ResTP-ARQ.

The first set of simulations in this section show the effect of increasing FEC strength on ResTP-NACK mode's performance. We evaluate the performance over various error rates. The ResTP-NACK parameters set for this simulation are shown in Table 5.4

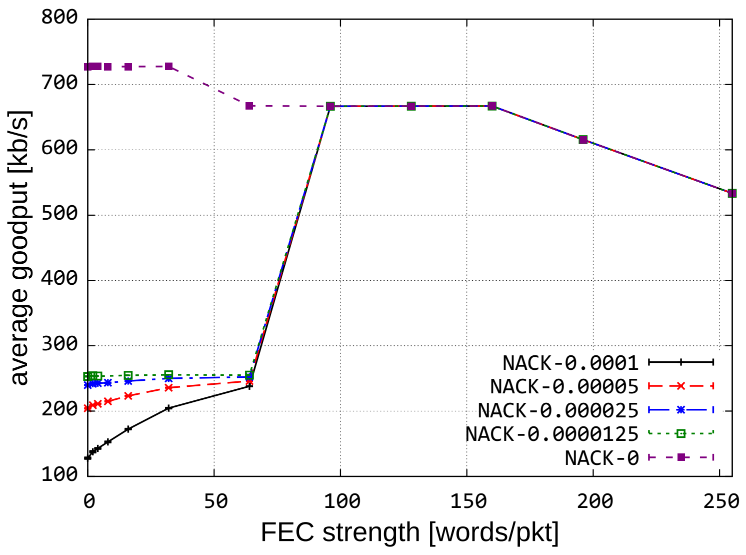


Figure 5.9. ResTP-NACK: Average goodput

The results from simulating ResTP-NACK show that for error rates > 0

Table 5.4. ResTP-NACK for varied strengths

Parameter	Value
Transport protocol	ResTP
ResTP mode	NACK
NACK retransmission limit	5
BER	0, 0.0000125, 0.000025, 0.00005, 0.0001
FEC strength	0,2,4,8,16,32,64,96,128,160,196,255

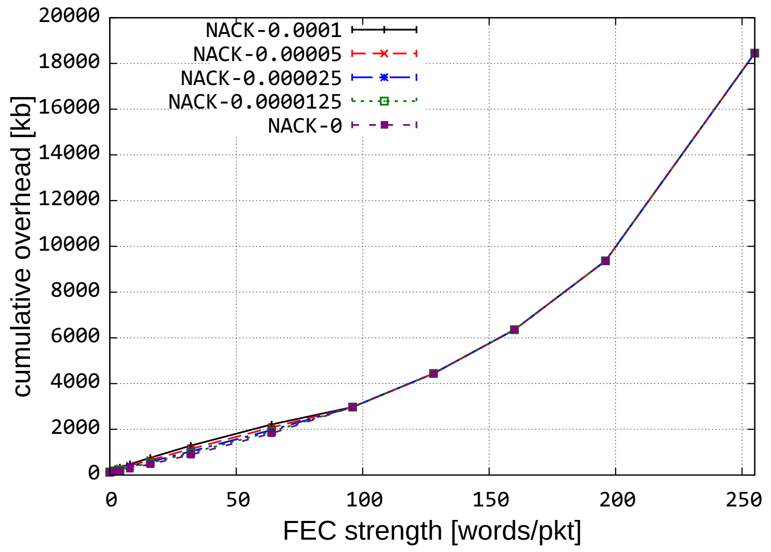


Figure 5.10. ResTP-NACK: Cumulative overhead

ResTP-NACK achieves significantly lower goodput when FEC strength is zero. Having no FEC strength implies that error correcting bits are not added to the payload. Hence any corrupted TPDU is dropped at the receiver. This also decreases the goodput by actually delivering less application data per unit time. However, as FEC strength increases the goodput continues to increases indicating much of the corrupted data is recovered. Figure 5.9 shows that for FEC strengths greater than 96 words/pkt the goodput begins to decreases since the number of FEC bytes added to every packet increases hence saturating the link. Furthermore, this also means the amount of actual application data in every packet decreases resulting in requiring more numbers of TPDUs to send the same amount of application data. This results in an increased overhead as seen in Figure 5.10.

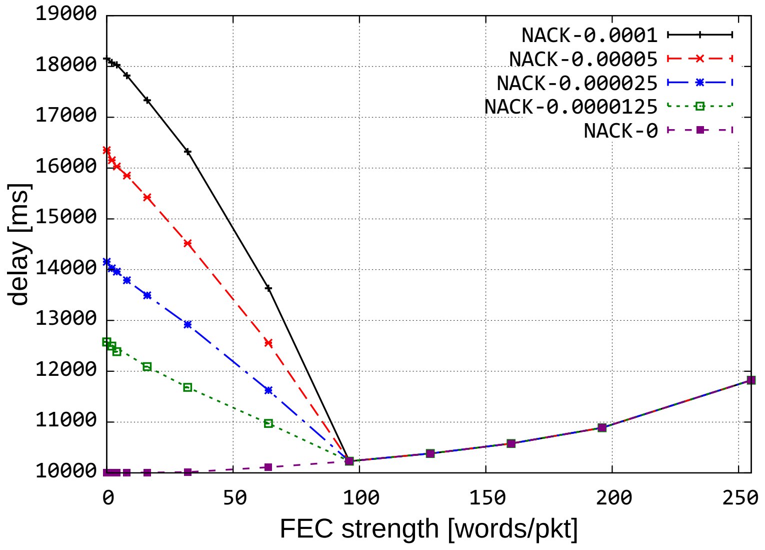


Figure 5.11. ResTP-NACK: Average delay

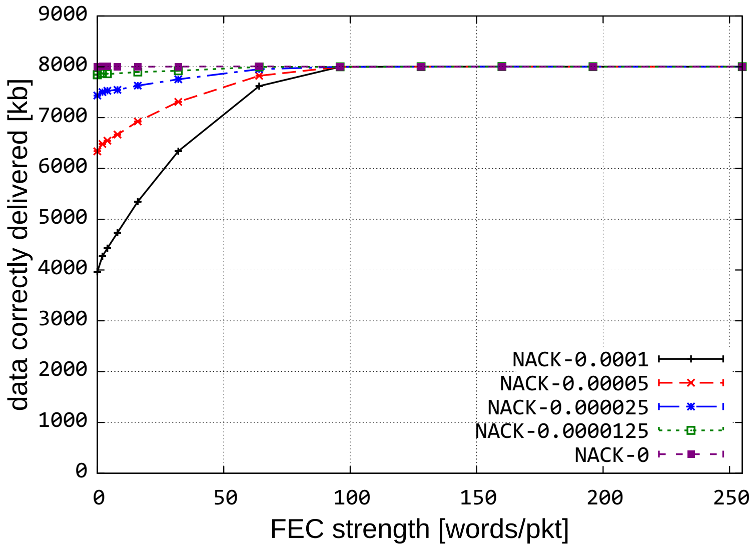


Figure 5.12. ResTP-NACK: Total data delivered

For error rates greater than zero, when no FEC bytes are added (FEC strength being zero) the number of corrupted TDPUs increases. This results in an increased number of NACKs sent by the receiver requesting retransmission. Although this results in some amount of application data being recovered the delay incurred increases. This is shown in Figure 5.11 in which ResTP-NACK for an error rate of 10^{-4} has the highest delay. However, the delay decreases up to a point as FEC strength increases. Futhermore, for FEC strengths greater than 96 words/pkt the delay increases since increased FEC words saturate the link. Figure 5.12 shows that for increased FEC strengths ResTP-NACK is able to deliver a higher percentage of data compared to when no FEC words are added. The significance of hyrbid error control can be seen in achieving higher end-to-end reliability for higher error rate under lower FEC strengths. Although a percentage of application data is not delivered when compared to a pure ResTP-FEC mode that uses open-loop error control, the percentage is significantly higher due to feeback from the receiver. The results for ResTP-FEC mode can be seen in our previous work [11].

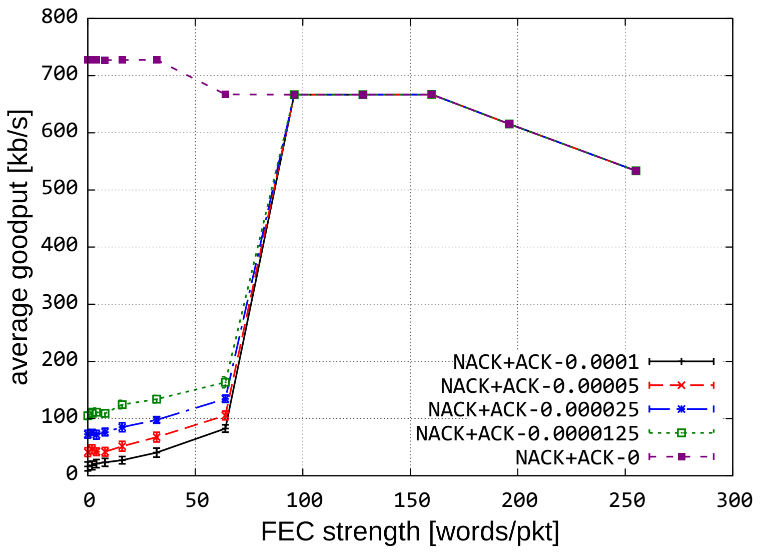


Figure 5.13. ResTP-NACK+MACK: Average goodput

The second set of simulations in this section show the effect of increasing BER for various error rates in the ResTP-NACK+MACK mode. The ResTP-NACK+MACK mode employs hybrid error control similar to the ResTP-NACK mode but differs by employing ACKs along with NACKs to achieve higher end-to-end reliability than ResTP-NACK and ResTP-FEC. ResTP parameters chosen for this set of simulation are shown in Table 5.5

Table 5.5. ResTP-NACK+MACK perf. over varied FEC strengths

Parameter	**Value**
Transport protocol	ResTP
ResTP mode	NACK+MACK
NACK retransmission limit	5
Limit retransmissions	0
BER	0, 0.0000125, 0.000025, 0.00005, 0.0001
FEC strength	0,2,4,8,16,32,64,96,128,160,196,255

The results from simulating ResTP-NACK+MACK show that for BER > 0 ResTP-NACK+MACK achieves significantly lower goodput when FEC strength is zero. This behavior is similar to ResTP-NACK, the difference is that lower goodputs are achieved due to an increased amount of retransmissions that not only saturate the link but also end up sending a lower amount of outstanding TPDUs. Having no FEC strength implies that error correcting bits are not added to the payload. Hence any corrupted TPDU is dropped at the receiver and NACKs are sent requesting retransmissions. Any lost TPDUs also increases the number of retransmissions. However, as FEC strength increases the goodput continues to increase indicating much of the corrupted data is recovered. Figure 5.13 shows that for FEC strengths greater than 96 words/pkt the goodput begins to decreases since the number of FEC bytes added to every packet increases, hence saturating the link. Furthermore, this also means the amount of actual application data in every

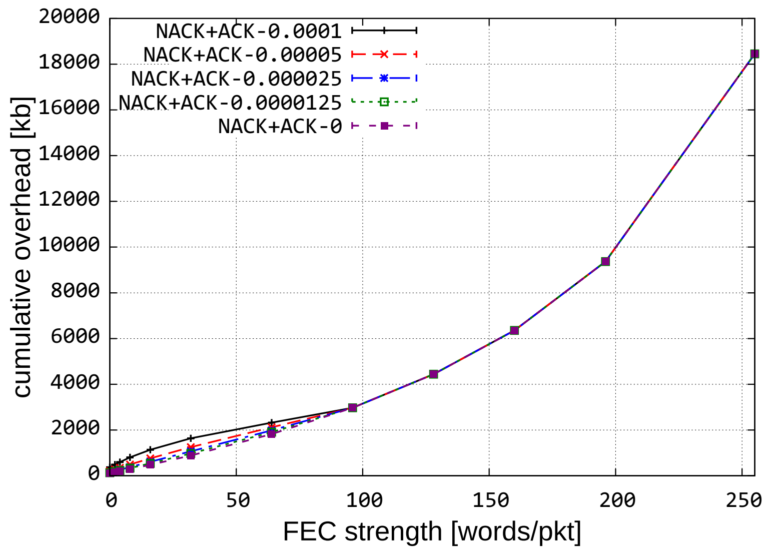


Figure 5.14. ResTP-NACK+MACK: Cumulative overhead

packet decreases resulting in requiring more TPDUs to send the same amount of application data. This results in increased overhead as seen in Figure 5.14.

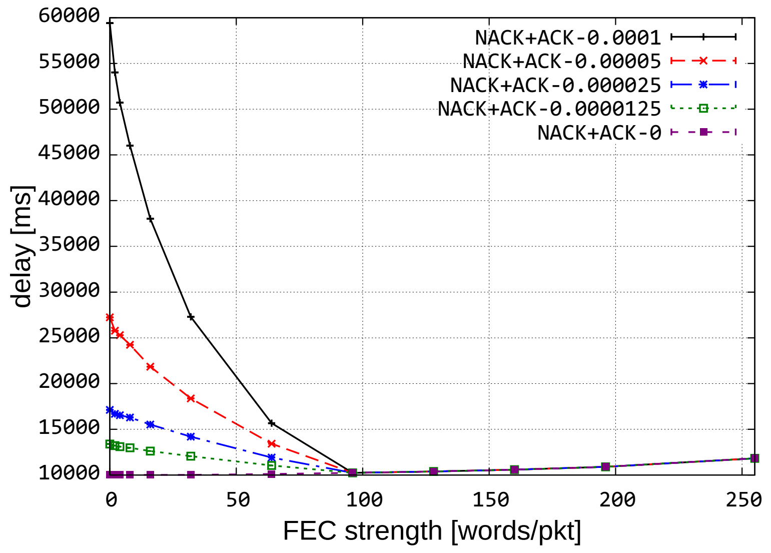


Figure 5.15. ResTP-NACK+MACK: Average delay

For BER > 0 when no FEC bytes are added, the number of corrupted TDPUs increases. This results in an increased number of NACKs sent by the receiver requesting retransmissions. Although some amount of application data is recovered, the delay incurred increases. This can be seen in Figure 5.15 in which a BER of 10^{-4} has the highest delay. However, the delay decreases up to a point as the FEC strength increases. This behavior is very similar to ResTP-NACK except for the difference in delay incurred is significantly higher for ResTP-NACK+MACK. From Figure 5.11 and Figure 5.15 we can see that at BER of 10^{-4}, ResTP-NACK only incurs delay of 18 s while ResTP-NACK+MACK incurs a delay of 60 s. Futhermore, for FEC strengths greater than 96 words/pkt, the delay increases since increased FEC words saturate the link.

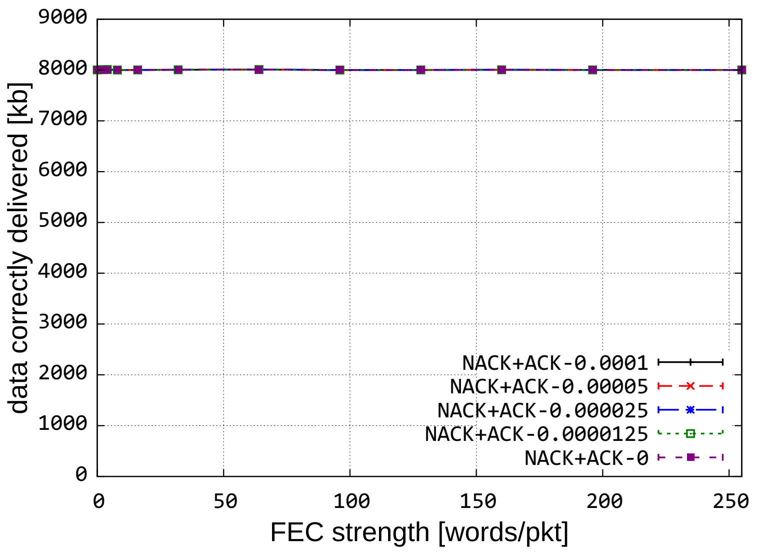


Figure 5.16. ResTP-NACK+MACK: Total data delivered

Figure 5.12 shows that ResTP-NACK+MACK is able to achieve full reliability at all BERs due to feedback from both ACKs and NACKs. Even when no FEC bytes are added for correcting errors, this mode delivers the packet based on the ACK feedback. Hence, this mode can achieve higher reliability with lower FEC strengths.

5.3.3 Effect of End-to-End Delay on ResTP Performance

In this section we present the performance of ResTP over various end-to-end delays. The premise for this simulation is that one of the characteristics than can be indicative of network connectivity is end-to-end delay. For example, low delays can be indicative of a well connected network and higher delays can be indicative of a disconnected network in which packets are ferried. Varying end-to-end delays have an impact on RTO estimation, making it especially challenging to estimate RTO accurately due to higher variance in sample RTTs.

Table 5.6. ResTP performance over varied end-to-end delay

Parameter	Value
Transport protocol	ResTP
ResTP mode	ARQ, FEC, NACK, NACK+MACK
Retransmission strategy	fastRetransmit
NACK retransmission limit	5
Limit retransmissions	0
Delay	1 ms, 10 ms, 100 ms, 1s, 10 s
BER	0.0000125
FEC strength	64

The following set of results compare the performance of ResTP-ARQ, ResTP-FEC, ResTP-NACK, and ResTP-NACK+MACK for increasing delays. Table 5.6 shows the parameters selected for simulating this scenario. We choose a BER of 125×10^{-7} based on our observations with the performance of ResTP-ARQ at low error rates. Figure 5.1 shows that the goodput drops drastically at very low error rates. In the case of previous simulations we used a delay of 10 s and we show how this long delay affects the goodput. We also choose an FEC strength of 64 as a default value since FEC strengths greater than 64 words/pkt are able to correct most of the errors in corrupted packets. Hence, having an FEC strength of 64 words/pkt poses a more challenging condition.

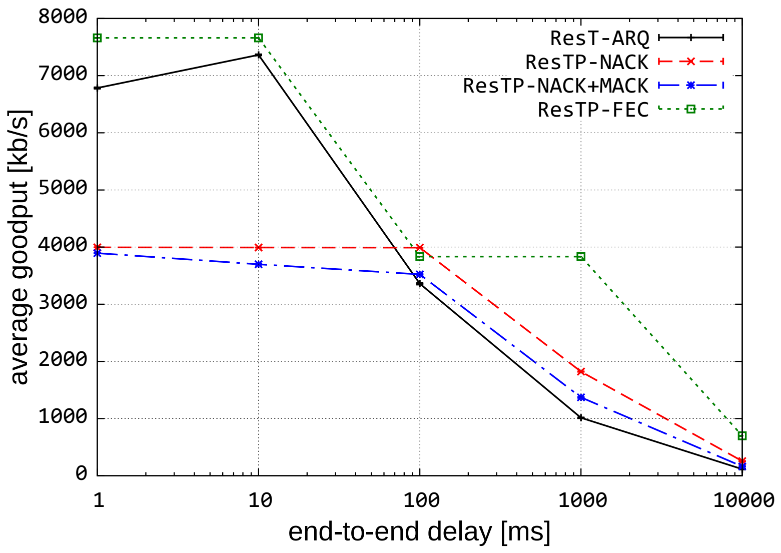


Figure 5.17. ResTP: Average goodput

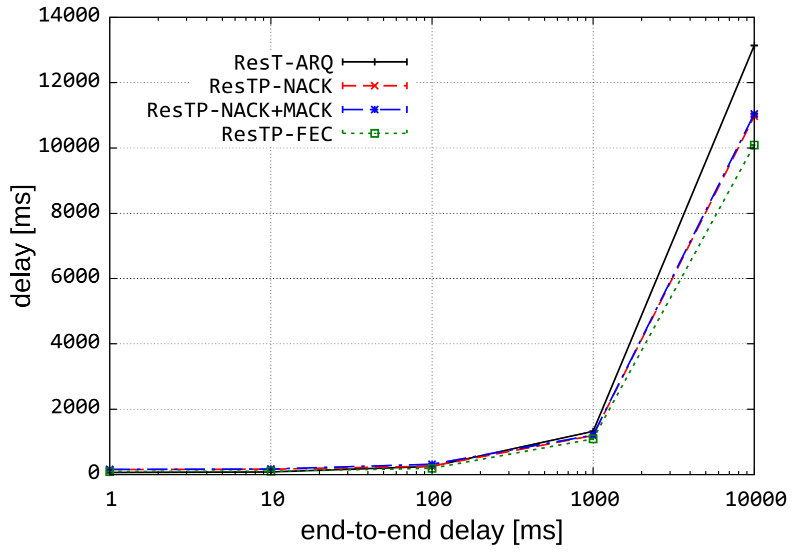


Figure 5.18. ResTP: Average delay

Since the hybrid-ARQ modes employ both extra FEC bytes and retransmissions to recover losses, they achieve lower goodput when end-to-end delays are low as seen in Figure 5.17. All the modes in general are able to achieve higher goodputs because the channel does not go unutilized, unlike when end-to-end de-

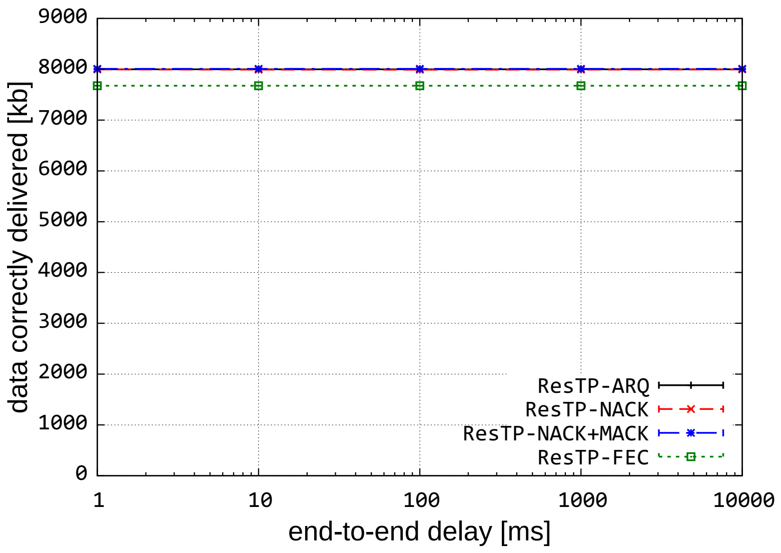


Figure 5.19. ResTP: Total data delivered

lays are longer. When the delays are as high as 10 s, modes involving closed-loop error control scheme take longer to detect and recover losses. Hence, the good-putput drops with increasing delays. This also results in higher delays as seen in Figure 5.18. ResTP-ARQ employs a pure closed-loop error control that can only detect losses based on ACKs. When delays are long, the ResTP-ARQ mode takes longer to deliver all of the application data. The same reasoning applies to ResTP-NACK and ResTP-NACK+MACK. However, they able to achieve better delay characteristics than ResTP-ARQ because they employ FEC codes to correct corrupted TPDUs whenever possible, resulting in fewer retransmissions. ResTP-FEC incurs the least delay since it does not employ any retransmission mechanism for error recovery. Although delays within the network affect the goodput and overall delay of the protocol, it does not affect the overhead or the total amount of data delivered. This is because these metrics are not dependent on time. This behavior is shown in Figure 5.20 and Figure 5.19.

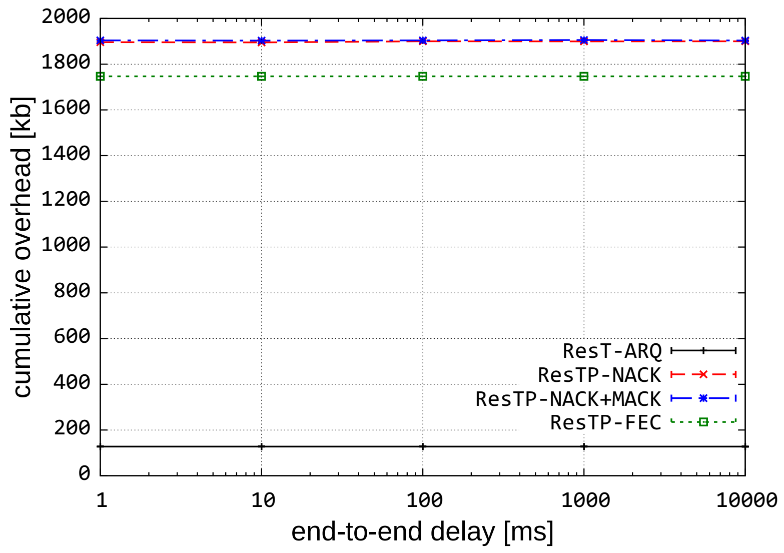


Figure 5.20. ResTP: Cumulative overhead

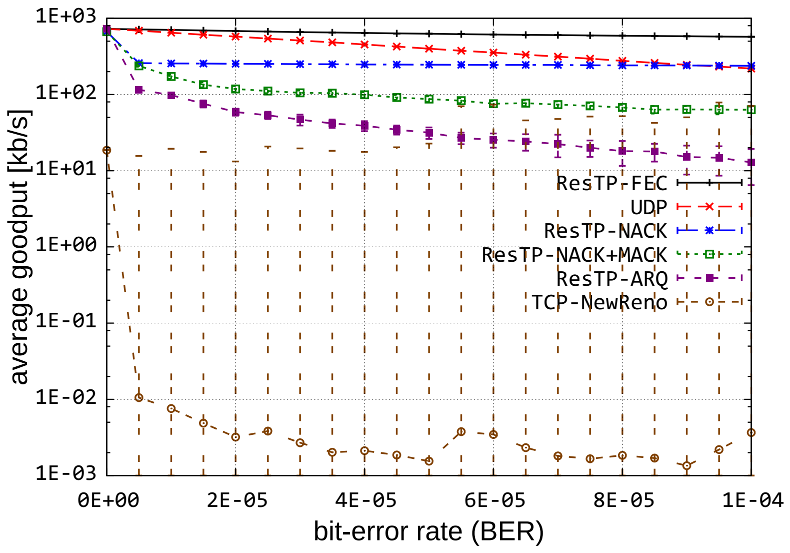


Figure 5.21. Mode comparison: Average goodput

5.3.4 Mode Comparison over Lossy Links

In this section, we discuss the effect of increasing BER on transport layer performance metrics using different ResTP modes, on UDP, and on TCP NewReno. The results show tradeoffs between the modes as a result of deteriorating chan-

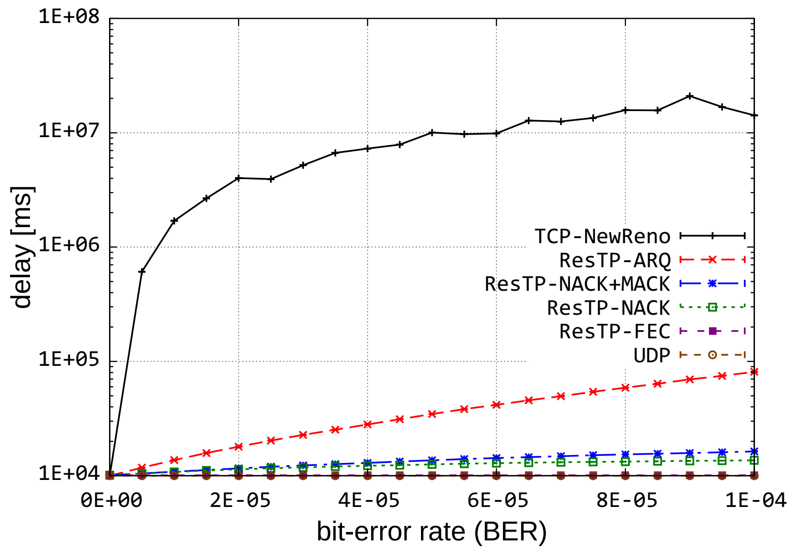


Figure 5.22. Mode comparison: Average delay

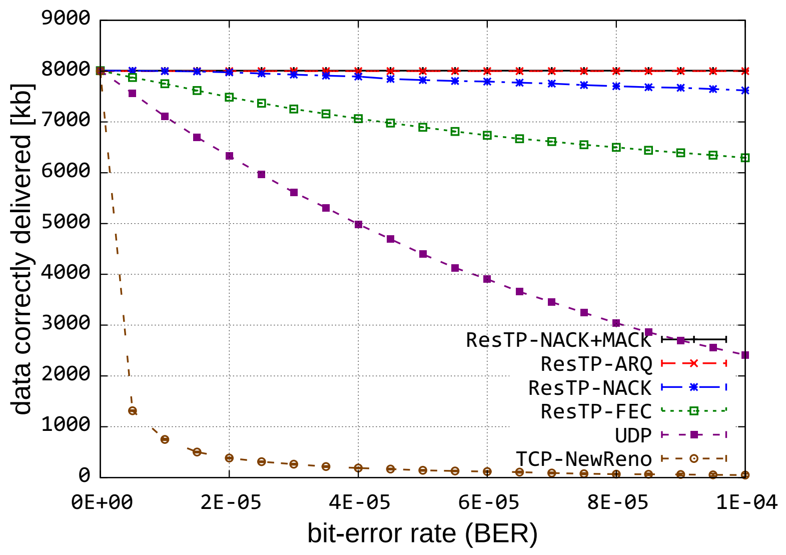


Figure 5.23. Mode comparison: Total data delivered

nel conditions. Figure 5.21 shows that the ResTP quasi-reliable mode is able to achieve better performance than ResTP reliable mode, which drops off significantly as the BER increases. Both the ResTP hybrid-ARQ modes perform better than ResTP reliable mode as expected. In comparison to the ResTP protocol,

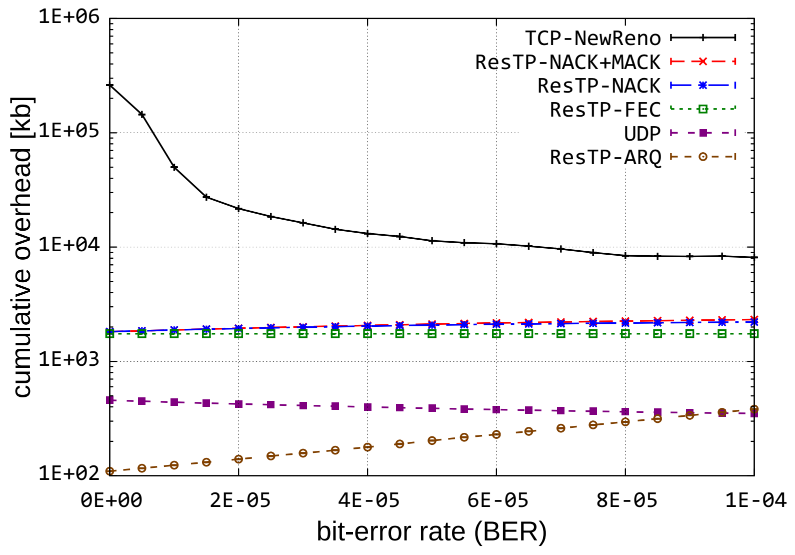


Figure 5.24. Mode comparison: Cumulative overhead

TCP backs off significantly as the BER increases due to its congestion control mechanism. The UDP protocol is able to achieve better goodput compared to ResTP reliable mode and hybrid-ARQ modes but it drops off as BER increases due to corrupted data. The ResTP reliable mode end-to-end delay increases significantly in comparison with ResTP hybrid-ARQ modes with a BER of 10^{-4} as shown in Figure 5.22. This can be attributed to the decrease in number of retransmissions due to the use of FEC codes. On the other hand, ResTP-FEC and UDP incur no delay as they do not retransmit any data. The TCP's end-to-end delay increases to nearly double than that of the ResTP fully-reliable mode at BER of 10^{-4} due to a greater number of retransmissions. Over the course of the simulation, the ResTP fully reliable mode and ResTP hybrid mode with both NACKs and MACKs enabled are able to deliver the full 1 MB of data transmitted for all BERs, but the ResTP hybrid mode with only NACKs enabled starts losing data for BER $> 2.5 \times 10^{-5}$, as shown in Figure 5.23. The plot also shows that FEC looses a higher percentage of the data due to corruption as the BER increases compared to ResTP NACK, because the hybrid mode requests retransmissions and recovers some data. The UDP protocol does not employ any error detection or recovery, hence loses a fraction of data at higher BERs. Furthermore, the plot shows that the TCP is unable to deliver the full 1 MB of data due to the decrease in the congestion window each time a data segment is lost, after which eventually the TCP connection times out. Figure 5.24 shows that both ResTP hybrid modes have larger overhead compared to a purely FEC-based quasi-reliable mode and purely-ARQ based fully-reliable mode because they have both additional FEC bits sent and data retransmitted. All the ResTP modes incur much less overhead compared to that of TCP NewReno.

5.4 Summary

In this chapter we presented the performance of the ResTP modes and compared it against different TCP and UDP protocols. We also presented the characteristics of each ResTP mode as a function of BER. We presented the tradeoffs between using a pure open-loop error control (FEC) and a pure closed-loop error control (ARQ) against a hybrid error control.

We found that hybrid error control performs better than pure FEC and pure ARQ at moderate error conditions by combining the advantages of both. We also found that higher end-to-end reliability is achieved at the cost of increased overhead. We also presented the effect of delay on the performance of the ResTP protocol and determined that with large delays the goodput drops significantly due to inefficient utilization of the channel. Furthermore, to show the efficiency of ResTP protocol, we compared it against the TCP and the UDP protocols. We showed that ResTP clearly outperforms TCP and UDP both in terms of reliability and in terms of achieving higher goodput.

Chapter 6

Conclusions and Future Work

This chapter summarizes the contributions of this thesis and its achievements. It highlights the advantages of using end-to-end alternative error control schemes as a part of the ResTP protocol. We also discuss the future work needed to improve the design of the protocol.

6.1 Contributions

The contributions of this thesis are:

- modified and implemented ResTP segment structure by adding additional MACK flag bit and additional MACK length field to the header
- implemented ResTP connection management scheme by modifying and implementing the ResTP state machine by defining event vectors, state-action pairs, and event lookup
- implemented ResTP reliable mode, ResTP-ARQ in ns-3

  - implemented selective-repeat ARQ mechanism with retransmissions based purely on timeouts and based on fast recovery
  - optimized the selective-repeat ARQ algorithm by employing aggregated MACKs to achieve improvement in performance over low error rates

- implemented type-I hybrid-ARQ modes, ResTP-NACK and ResTP-NACK+MACK, based on only NACKs and both ACKs and NACKs respectively in ns-3

- analyzed the performance of end-to-end ARQ (ResTP-ARQ), hybrid-ARQ (ResTP-NACK and ResTP-NACK+MACK), end-to-end FEC (ResTP-FEC) and compared them against each other and with traditional TCP and UDP protocols

- examined the tradeoffs between using open-loop and closed-loop error control with hybrid filling in the spectrum

6.2 Conclusions

To address the end-to-end issues posed by challenged networks we employed adaptive error control schemes in the design of ResTP protocol. We provided reasons as to why traditional protocols fail to perform in such environments by learning their drawbacks. We also discussed various optimizations made to TCP protocol for wireless networks. We presented various error control schemes that can be used to detect and recover lost or corrupted packets. Furthermore, in this thesis we presented the ns-3 implementation of the ResTP protocol and described in detail the implementation of ResTP-ARQ, ResTP-NACK+MACK, and ResTP-NACK modes.

Simulations performed on these protocols show the effect of BER on the performance of these modes. We showed the tradeoffs between using ResTP reliable mode, quasi-reliable, and the hybrid modes in terms of achieving reliability, incurring overhead, and compromizing delay and goodput. The results of ResTP-ARQ, ResTP-FEC, ResTP-NACK, and ResTP-NACK+MACK modes showed that while ResTP-ARQ and ResTP-NACK+MACK are able to achieve high reliability they do incur larger delays due to retransmission. However, ResTP-NACK+MACK mode is able to achieve better goodput and delay than ResTP-ARQ since it has more accurate feedback from the receiver with the use of NACKs. The number of retransmissions in the ResTP-NACK+MACK is more controlled compared to the ResTP-ARQ mode. ResTP-FEC and ResTP-NACK incur lower delay since they do not employ any retransmission strategies. This is also why they do not achieve full reliability. However, ResTP-NACK is able to achieve higher system reliability than ResTP-FEC since it employs a feedback mechanism through which it is able to recover losses. These observations show that full reliability can be achieved only by employing positive ACKs.

Another important aspect of these simulations was the effect of BER on TCP and UDP protocols. In Chapter 5 we established that the performance of TCP-NewReno degrades with increasing BERs and ResTP clearly outperforms TCP-NewReno. TCP fails to distinguish between congestion based losses and corruption based losses and triggers the congestion control scheme that reduces TCP window size each time a loss occurs. At higher BER TCP backs off until the connection times out thus achieving lower data reliability. UDP, however, incurs no delays since it does not perform any retransmissions. UDP also achieves higher goodput than ResTP at lower BER but loses a significant amount of data at higher

BER thus decreasing the goodput.

6.3 Publications

Following are the publications as a result of my research with the ResiliNets group:

1. Justin P. Rohrer, **Kamakshi Sirisha Pathapati**, Truc Anh N. Nguyen, and James P.G. Sterbenz. Opportunistic Transport for Disruption Airborne Networks. In *Proceedings of IEEE Military Communications Conference (MILCOM 2012)*, San Diego, CA, November 2012 (to appear).

2. **Kamakshi Sirisha Pathapati**, Justin P. Rohrer, and James P.G. Sterbenz. Comparision 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 (to appear).

3. **Kamakshi Sirisha Pathapati**, Anh Nguyen, Justin P. Rohrer, and James P.G. 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, **awarded best graduate-student paper**.

4. **Kamakshi Sirisha Pathapati**, Justin P. Rohrer, and James 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.

6.4 Future Work

This thesis presents the design and implementation of ResTP, a resilient and disruption tolerant transport protocol. There is scope for improving the design of the ResTP protocol by employing type-II hybrid-ARQ schemes, by investigating a mechanisms to detect lost packets in ResTP-NACK mode, and by employing FEC to protect the header information. Employing type-II hybrid error control schemes means modeling and implementing convolution codes instead of linear block codes such as Reed-Solomon. Type-II hybrid error control schemes offer various retransmission strategies, for example, while some retransmit only the error correcting codeword some retransmit the original TPDU. This requires an investigation into various strategies and their performance in varying error conditions. ResTP-NACK mode is able to recover only corrupted TPDUs but fails to recover lost TPDUs since it does not look for missing seqeunce numbers.

Most of the performance analysis is based on one challenge scenario, an example of a bandwidth constrained network with varying BER. This shows that there is a need to model more challenge scenarios to study the effects of parameters other than channel BER. For example, in wireless scenarios, the effects of mobility and node velocity can be analyzed on the performance of ResTP and attacks can be modelled to disrupt the network [51]. Other possible scenarios include running ResTP over various mobile ad hoc routing protocols such as DSDV, AODV, OLSR, DSR and performance analysis of ResTP in the face of link failures and node failures [52,53]. Another important area to evaluate performance is to study the effect of high delay variance on the performance of ResTP. Since ResTP's RTO is calculated using current RTT, when the delay variation is very high, estimating RTO becomes challenging. The future work also includes comparing the performance of ResTP against various TCP variants such as TCP Vegas, TCP Westwood, TCP Westwood+, and SCPS-TP. These variants are currently being implemented in ns-3 by the members of the ResiliNets research group [54–57].

References

[1] James P. G. Sterbenz, David Hutchison, Egemen K. Çetinkaya, Abdul Jabbar, Justin P. Rohrer, Marcus Schöller, and Paul Smith. Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines. *Computer Networks*, 54(8):1245–1265, 2010.

[2] Justin P. Rohrer, Abdul Jabbar, Erik Perrins, and James P. G. Sterbenz. Cross-layer architectural framework for highly-mobile multihop airborne telemetry networks. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, pages 1–9, San Diego, CA, November 2008.

[3] C. A. Sunshine and Y. K. Dalal. Innovations in internetworking. chapter Connection management in transport protocols, pages 245–264. Artech House, Inc., Norwood, MA, USA, 1988.

[4] R. W. Watson and S. A. Mamrak. Gaining efficiency in transport services by appropriate design and implementation choices. *ACM Transactions on Computer Systems*, 5(2):97–120, May 1987.

[5] David Feldmeier. An overview of the TP++ transport protocol project. In Ahmed N. Tantawy, editor, *High Performance Networks: Frontiers and Experience*, volume 238 of *Kluwer International Series in Engineering and*

Computer Science, chapter 8. Kluwer Academic Publishers, Boston, MA, USA, 1993.

[6] Justin P. Rohrer, Abdul Jabbar, Egemen K. Çetinkaya, Erik Perrins, and James P.G. Sterbenz. Highly-dynamic cross-layered aeronautical network architecture. *IEEE Transactions on Aerospace and Electronic Systems*, 47(4):2742–2765, October 2011.

[7] Justin P. Rohrer, Egemen K. Çetinkaya, Hemmanth Narra, Dan Broyles, Kevin Peters, and James P. G. Sterbenz. AeroRP Performance in Highly-Dynamic Airborne Networks using 3D Gauss-Markov Mobility Model. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, pages 834–841, Baltimore, MD, November 2011.

[8] Justin P. Rohrer, Erik Perrins, and James 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.

[9] Justin P. Rohrer. *End-to-End Resilience Mechanisms for Network Transport Protocols*. PhD thesis, The University of Kansas, Lawrence, KS, November 11 2011.

[10] Kamakshi Sirisha Pathapati, Justin P. Rohrer, and James 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.

[11] Kamakshi Sirisha Pathapati, Truc Anh N. Nguyen, Justin P. Rohrer, and James P.G. 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.

[12] Kamakshi Sirisha Pathapati, Justin P. Rohrer, and James P.G. 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.

[13] Hemanth Narra, Egemen K. Çetinkaya, and James P.G. Sterbenz. Performance analysis of aerorp with ground station updates in highly-dynamic airborne telemetry networks. In *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2011.

[14] Hemanth Narra, Egemen K. Çetinkaya, and James P.G. Sterbenz. Performance Analysis of AeroRP with Ground Station Advertisements. In *Proceedings of the ACM MobiHoc Workshop on Airborne Networks and Communications*, pages 43–47, Hilton Head Island, SC, June 2012.

[15] Mohammed J.F. Alenazi, Egemen K. Çetinkaya, Justin P. Rohrer, and James 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.

[16] Santosh Ajith Gogi, Dongsheng Zhang, Egemen K. Çetinkaya, Justin P. Rohrer, and James P. G. Sterbenz. Implementation of the AeroTP Transport Protocol in Python. In *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2012.

[17] Justin P. Rohrer and James P. G. Sterbenz. Performance and disruption tolerance of transport protocols for airborne telemetry networks. In *Proceedings of the International Telemetering Conference (ITC)*, Las Vegas, NV, October 2009.

[18] The ns-3 network simulator. `http://www.nsnam.org`, July 2009.

[19] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[20] Van Jacobson. Congestion avoidance and control. *SIGCOMM Comput. Commun. Rev.*, 18(4):314–329, 1988.

[21] J. Postel. User Datagram Protocol. RFC 768 (Standard), August 1980.

[22] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz. A comparison of mechanisms for improving TCP performance over wireless links. *IEEE/ACM Transactions on Networking (TON)*, 5(6):756–769, 1997.

[23] Van Jacobson. Modified TCP congestion avoidance algorithm, April 1990.

[24] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *SIGCOMM Comput. Commun. Rev.*, 24(4):24–35, 1994.

[25] S Liu, T Başar, and R Srikant. TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks. *ScienceDirect - Performance Evaluation*, 65:417–440, 2008.

[26] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H Katz. Improving TCP/IP performance over wireless networks. In *Proceedings of the ACM Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 2–11. ACM Press, 1995.

[27] I.F. Akyildiz, G. Morabito, and S. Palazzo. TCP-Peach: a new congestion control scheme for satellite ip networks. *Networking, IEEE/ACM Transactions on*, 9(3):307–321, Jun 2001.

[28] Claudio Casetti, Mario Gerla, Saverio Mascolo, M. Y. Sanadidi, and Ren Wang. TCP Westwood: End-to-end congestion control for wired/wireless networks. *Wireless Networks*, 8(5):467–479, 2002.

[29] T. V. Lakshman and Upamanyu Madhow. The performance of tcp/ip for networks with high bandwidth-delay products and random loss. *IEEE/ACM Transactions on Networking*, 5(3):336–350, 1997.

[30] Robert C. Durst, Gregory J. Miller, and Eric J. Travis. TCP extensions for space communications. In *MobiCom '96: Proceedings of the 2nd annual international conference on Mobile computing and networking*, pages 15–26, New York, NY, USA, November 1996. ACM Press.

[31] R. W. Watson. The Delta-T transport protocol: Features and experience. *Local Computer Networks*, 1989.

[32] Shu Lin, D. Costello, and M. Miller. Automatic-repeat-request error-control schemes. *IEEE Communication Magazine*, 22(12):5–17, December 1984.

[33] M.D. Rice and S.B. Wicker. Adaptive error control based on type-I hybrid-ARQ protocols. In *SBT/IEEE International Telecommunications Symposium, 1990*, pages 176–180, September 1990.

[34] C. Leung and A. Lam. Forward error correction for an ARQ scheme. *IEEE Transactions on Commmunications*, 29(10):1514–1519, oct 1981.

[35] Hang Liu, Hairuo Ma, Magda El Zarki, and Sanjay Gupta. Error control schemes for networks: an overview. *Mobile Network Applications*, 2(2):167–182, October 1997.

[36] Shu Lin and P. Yu. A hybrid ARQ scheme with parity retransmission for error control of satellite channels. *IEEE Transactions on Communications*, 30(7):1701–1719, July 1982.

[37] E. Y. Rocher and R. L. Pickholtz. An analysis of the effectiveness of hybrid transmission schemes. *IBM J. Res. Dev.*, 14(4):426–433, July 1970.

[38] Shu Lin, Daniel J. Costello. Jr., and Michael J. Miller. Automatic-Repeat-Request Error-Control Schemes. *Communications Magazine, IEEE*, 22(12):5–17, December 1984.

[39] P. Turney. An improved stop-and-wait ARQ logic for data transmission in mobile radio systems. *IEEE Transactions on Communications*, 29(1):68 – 71, jan 1981.

[40] R.A. Comroe and Jr. Costello, D.J. ARQ schemes for data transmission in mobile radio systems. *IEEE Transactions on Vehicular Technology*, 33(3):88–97, August 1984.

[41] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Obsoleted by RFC 5681, updated by RFC 3390.

[42] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), November 2000.

[43] C. Q. Yang, E. Hossain, and V. K. Bhargava. On adaptive hybrid error control in wireless networks using reed-solomon codes. *Transaction on Wireless Commmunications*, 4(3):835–840, may 2005.

[44] Sastry A. Improving automatic repeat-request ARQ performance on satellite channels under high error rate conditions. *IEEE Transactions on Communications*, 23(4):436–439, April 1975.

[45] D. Mandelbaum. An adaptive-feedback coding scheme using incremental redundancy (corresp.). *IEEE Transactions on Information Theory*, 20(3):388–389, may 1974.

[46] R. W. Watson. Timer-based mechanisms in reliable transport protocol connection management. *Computer Networks*, 5(1):47–56, February 1981.

[47] Gonca Gursun, Ibrahim Matta, and Karim Mattar. On the performance and robustness of managing reliable transport connections. Technical Report BUCS-TR-2009-014, Boston Univeristy, Boston, MA, April 2009.

[48] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018 (Proposed Standard), October 1996.

[49] R. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122 (Standard), October 1989. Updated by RFCs 1349, 4379.

[50] James P.G. Sterbenz. Communication networks, the university of kansas eecs 780. http://www.ittc.ku.edu/~jpgs/courses/nets/.

[51] Dongsheng Zhang, Santosh Ajith Gogi, Dan S. Broyles, Egemen K. Çetinkaya, and James P.G. Sterbenz. Modelling malicious attack in MANETs. In *Great Plains Graduate Student Network Research Summit*, Kansas City, MO, May 2012. Extended Abstract.

[52] Yufei Cheng, Egemen K. Çetinkaya, and James P.G. Sterbenz. Dynamic source routing (DSR) protocol implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*, pages 367–374, Sirmione, March 2012.

[53] Hemanth Narra, Yufei Cheng, Egemen K. Çetinkaya, Justin P. Rohrer, and James P.G. Sterbenz. Destination-sequenced distance vector (DSDV) routing protocol implementation in ns-3. In *Proceedings of the ICST SIMUTools Workshop on ns-3 (WNS3)*, pages 439–446, Barcelona, Spain, March 2011.

[54] Luigi A. Grieco and Saverio Mascolo. Performance evaluation and comparison of westwood+, new reno, and vegas TCP congestion control. *SIGCOMM SIGCOMM Computer Communication Review*, 34(2):25–38, apr 2004.

[55] S. Mascolo, L. A. Grieco, R. Ferorelli, P. Camarda, and G. Piscitelli. Performance evaluation of westwood+ TCP congestion control. *Performance Evaluation*, 55(1–2):93–111, jan 2004.

[56] Siddharth Gangadhar, Truc Anh N. Nguyen, Greeshma Umapathi, Kamakshi Sirisha Pathapati, and James P.G. Sterbenz. A Comparative Simula-

tion Study of Transport Protocols. `http://www.ittc.ku.edu/resilinets/posters/ITTC-IAB-poster-2012-tcpComparision.pdf`, June 2012.

[57] Truc Anh N. Nguyen, Siddharth Gangadhar, Greeshma Umapathi, and James P.G. Sterbenz. Performance Evaluation of the AeroTP Protocol in Comparison to TCP NewReno, TCP Westwood, and SCPS-TP. In *Proceedings of the International Telemetering Conference (ITC)*, San Diego, CA, October 2012.