ResTP: A Configurable and Adaptable Multipath Transport Protocol for Future Internet Resilience

By

Truc Anh N. Nguyen

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

Date Defended: 3 September 2021

The Dissertation Committee for Truc Anh N. Nguyen
certifies that this is the approved version of the following dissertation:

ResTP: A Configurable and Adaptable Multipath Transport Protocol for Future Internet Resilience

Chairperson: Prof. Victor S. Frost

Date approved: 3 September 2021

Abstract

Motivated by the shortcomings of common transport protocols, e.g., TCP, UDP, and MPTCP, in modern networking and the belief that a general-purpose transport-layer protocol, which can operate efficiently over diverse network environments while being able to provide desired services for various application types, we design a new transport protocol, ResTP. The rapid advancement of networking technology and use paradigms is continually supporting new applications. The configurable and adaptable multipath-capable ResTP is not only distinct from the standard protocols by its flexibility in satisfying the requirements of different traffic classes considering the characteristics of the underlying networks, but by its emphasis on providing resilience. Resilience is an essential property that is unfortunately missing in the current Internet. In this dissertation, we present the design of ResTP, including the services that it supports and the set of algorithms that implement each service. We also discuss our modular implementation of ResTP in the open-source network simulator ns-3. Finally, the protocol is simulated under various network scenarios, and the results are analyzed in comparison with conventional protocols such as TCP, UDP, and MPTCP to demonstrate that ResTP is a promising new transport-layer protocol providing resilience in the Future Internet (FI).

Page left intentionally blank.

Acknowledgments

I would like to sincerely thank my current advisor, Prof. Victor S. Frost for his support and guidance. Without his help, I would not have been able to continue my study and complete my dissertation.

I would like to sincerely thank my deceased advisor, Prof. James P.G. Sterzenz for his support and guidance during the initial years of my Ph.D study.

I would like to thank my committee members, including the former members Prof. Gary Minden, Prof. Michael Welzl, and Prof. Justin Rohrer, and the current members Prof. Bo Luo, Prof. Huynjin Seo, Prof. Taejoon Kim, and Prof. Morteza Hashemi for their valuable feedback to improve this dissertation.

I would like to thank the ResiliNets group members for their suggestions and discussions about research ideas. I especially would like to thank Siddharth Gangadhar for collaborating with me on multiple TCP-related publications and for his work on the TCP post-processing scripts to produce useful statistics from ns-3 simulations. These scripts are used to generate some of the results present in this dissertation

I would like to thank the Information and Telecommunication Technology Center (ITTC) network system administrators and the ITTC administrative staff for their support during the development of this dissertation.

Last but not least, I would like to thank my family for being my biggest supporters. Without their love and encouragement, I would not have gone this far.

Page left intentionally blank.

Contents

List of Figures

Page left intentionally blank.

List of Tables

Chapter 1

Introduction and Motivation

The Internet has been dominated by the two standard transport-layer protocols for decades: the Transmission Control Protocol (TCP) [1–4] and the User Datagram Protocol (UDP) [5]. While UDP is a light-weight protocol, which provides connectionless and unreliable datagram transmissions between communicating hosts, TCP is a connection-oriented and reliable transport protocol, which provides flow control, congestion control, and in-order data delivery service. Both of these have made significant contributions to the success of the Internet, especially in sustaining the global network's scalability, robustness, and stability. However, the emergence of new application classes and dynamic network environments as a result of the rapid growth in networking technologies and use paradigms has challenged the performance of both UDP and TCP since the modern networks are characteristically different from the environments for that these protocols were tailored. As their design assumptions are violated, impairments are inevitable. These issues have resulted in numerous research efforts and studies, yielding scores of proposals, ranging from modifications and extensions on top of the standards to new protocols. Unfortunately, the majority of these approaches tend to address specific problems for particular networks and application types instead of seeking for a general solution that can demonstrate itself as a potential candidate for the Future Internet (FI).

1.1 Thesis Statement

The inflexibility of TCP and UDP in satisfying the requirements of different traffic types and adapting to various network environments is mainly caused by their fixed service sets. An obvious example is the debate among multimedia application developers when making their decisions on whether to select UDP or TCP as the underlying transport protocol. Given that these applications are loss-tolerable, but delay-sensitive, the light-weight UDP is the winner. However, its lack of congestion control poses a high risk to the network stability. Furthermore, the Internet as it is today with UDP and TCP is not a resilient system. Therefore, our thesis statement is:

> *A configurable and adaptable multipath-capable general-purpose transport-layer protocol designed to achieve resilience and survivability is necessary for Future Internet. Composability and cross-layering are essential to allow the protocol to provide resilient services to different application classes operating across various network environments, each with distinct characteristics.*

The goal of this dissertation is fourfold:

1. Introduce a new general-purpose transport-layer protocol for FI resilience that is designed based on an extensive study of the conventional protocols.

2. Present a highly modular implementation of the protocol in the ns-3 open-source network simulator [6] with the focus on reducing the complexity while increasing its extensibility as the protocol evolves over time.

3. Discuss the evaluation of the protocol by simulating it under a variety of network scenarios when carrying different traffic types.

4. Present a comparison of ResTP with the conventional protocols UDP, TCP, and MPTCP

1.2 Proposed Solution

We designed a new transport-layer protocol for FI named Resilient Transport Protocol (ResTP), which was originally proposed [7] as the general-purpose version of the domain-specific Aeronautical Transport Protocol (AeroTP) for a highly-dynamic airborne telemetry network environment [8–10]. To achieve resilience, the protocol development process follows closely the set of design principles derived from the ResiliNets framework, in which *resilience* is defined as *the ability of the network to provide and maintain an acceptable level of service in face of various faults and challenges to normal operation* [11]. With this definition, resilience covers a broad number of relevant disciplines, including those relating to challenge tolerance (fault tolerance, survivability, disruption tolerance), trustworthiness (reliability, performability, security), robustness, and complexity. Similar to the conventional protocols, ResTP supports multiple transport-layer services, including multiplexing/demultiplexing, connection management, error control, transmission control (flow/congestion), and multipath data transfer. However, unlike the current transport protocols, each of ResTP services is implemented by a set of algorithms that have been shown to exhibit their value in different network environments for different types of traffic. The mechanisms make up the ResTP functionalities are *composable*, resulting in multiple ResTP modes (or configurations) that are specifically tailored for particular networks and applications. When ResTP runs on top of the GeoDiverse Routing Protocol (GeoDivRP), a routing protocol that is capable of providing a set of geographically diverse paths developed by other member of the ResiliNets group [12–14], cross-layering is exploited to survive network challenges.

ResTP employs modular design and modular programming that are especially crucial for the *flexible* ResTP to achieve low *complexity* and high *extensibility* as the protocol evolves along with the networking technology evolution.

1.3 ResTP vs. Conventional Protocols

ResTP is not the first transport-layer protocol that employs algorithm composibility, and obviously not the only protocol that allows data transmissions on multiple paths simultaneously. However, to the best of our knowledge, ResTP is the first composible and adaptive multipath transport protocol of which the development is solely motivated by the determination to increase resilience and survivability.

TP++ [15] is another composable transport protocol developed for heterogeneous high-speed networks. Similar to ResTP, TP++ is designed to carry different types of traffic, including transactions, bulk data transfer, and delay-sensitive services. However, TP++ only supports single-path transmissions. Furthermore, the protocol only implements multiple algorithms for its error control service, while utilizing a timer-based connection management scheme and assuming that congestion control is handled by the underlying network.

CTP [16] is a configurable and extensible transport protocol that is implemented using the Cactus microprotocol composition framework [17]. It provides various transport-layer services and functionalities, including reliability, ordering, security, jitter control, congestion control, flow control, data and header compression, MTU discovery, message fragmentation and collation, and connection management with each service implemented by multiple algorithms. However, similar to TP++, CTP is a single-path protocol. Moreover, the goal of CTP is to serve new application classes and networking environments without targeting to achieve resilience and survivability.

Multipath Transmission Control Protocol (MPTCP) [18] is a recent TCP extension that utilizes TCP options to allow communication between a pair of hosts through multiple paths concurrently. MPTCP utilizes TCP 3-way handshake for the master subflow establishment and 4-way handhskae for all subsequent subflows. All subflows (except the master one) can be used as part of the connection immediately or for backup only. The closing of an MPTCP connection requires the terminations at both connection and subflow levels, each through the exchange of FIN messages as in regular TCP. For flow control, MPTCP maintains a single receiving buffer that is shared by all subflows and uses connection-level ACKs to slide the window. In addition, the protocol employs the ARQ with retransmissions error control technique at both connection and subflow levels. For congestion control, there have been multiple coupled congestion control algorithms proposed for MPTCP, including the Link Increase Algorithm (LIA) [19], Opportunistic Linked-Increased Algorithm (OLIA) [20], Balanced Linked Adaptation Algorithm (BALIA) [21], and Weighted Vegas (WVEGAS) [22].

MPTCP inherits the strictly intertwined flow/error/congestion control limitation of TCP and can only provide a fixed set of services. This limits the flexibility of MPTCP in supporting diverse network applications. ResTP, even in its multipath mode, is capable of configuring its service (by mixing and matching its sets of mechanisms) to meet specific application requirements.

Because the functionality of MPTCP is constructed based on TCP options and the imposed limitation on the total option length, extensions or modifications on MPTCP when desired are also constrained.

MPTCP creates subflows by using all possible endpoint addresses, path qualities are not verified before they are utilized. The protocol neither has any knowledge of the characteristics of these paths nor can ensure their disjointness until it starts to use them [23].

Furthermore, MPTCP provides no mechanisms for the application and network layers to participate in the path selection process. On the other hand, the ResTP-GeoDivRP protocol stack with cross layering allows both the upper and lower layers to engage in path selection. The protocol stack also allows geographically diverse paths to be selected for data transmissions, resulting in the achievement of not only fault tolerance, but also survivability.

1.4 Contributions

The main contributions of this dissertation are:

1. Design a new transport-layer protocol, ResTP for FI resilience.
2. Implement a model of the protocol in the ns-3 network simulator.
3. Simulate and compare the performance of ResTP with the conventional protocols using various application and network types to highlight the protocol potentials.

1.5 Relevant Publications

The research presented in this dissertation has resulted in a number of publications, including the following.

Journal articles

Peer-reviewed conference proceedings

5. **Truc Anh N. Nguyen** and James P.G. Sterbenz, "Connection Management in a Resilient Transport Protocol," *13th International Conference on Design of Reliable Communication Networks (DRCN 2017)*, Munich, March 2017.

4. Yufei Cheng, **Truc Anh N. Nguyen**, Md. Moshfequr Rahman, Siddharth Gangadhar, and James P.G. Sterbenz, "Geodiverse Routing Protocol with multipath forwarding compared to MPTCP," *8th International Workshop on Resilient Networks Design and Modeling (RNDM 2016)*, Sweden, September, 2016.

3. Yufei Cheng, **Truc Anh N. Nguyen**, Md. Moshfequr Rahman, Siddharth Gangadhar, Mohammed J.F. Alenazi, and James P.G. Sterbenz, "Cross-Layer Geodiverse Protocol Stack for Resilient Multipath Transport and Routing using OpenFlow," *12th International Conference on Design of Reliable Communication Networks (DRCN 2016)*, Paris, France, March 2016.

2. **Truc Anh N. Nguyen**, Justin P. Rohrer, and James P.G. Sterbenz, "ResTP - A Transport Protocol for FI Resilience," *10th International Conference on Future Internet Technologies (CFI 2015)*, Seoul, Korea, June 2015.

1. Justin P. Rohrer, Kamakshi Sirisha Pathapati, **Truc Anh N. Nguyen**, and James P.G. Sterbenz, "Opportunistic Transport for Disrupted Airborne Networks," *The IEEE Military Communications Conference (MILCOM)*, Orlando, FL, USA, 29 Oct–1 Nov, 2012, pp. 737–745.

1.6 Additional Publications

Other publications have been resulted from my research during my graduate studies.

4. James P.G. Sterbenz, Justin P. Rohrer, Mohammed J.F. Alenazi, **Truc Anh N. Nguyen**, Egemen K. Centinkaya, Hemanth Narra, Kamakshi S. Pathapati, and Kevin Peters, "Distuption-Tolerant Airborned Networks and Protocols," in *UAV Networks and Communications*, Cambridge University Press, 2018, pp.587–95.

3. **Truc Anh N. Nguyen** and James P.G. Sterbenz, "An Implementation and Analysis of SCPS-TP in ns-3," *Workshop on ns-3 (WNS3'17)*, Porto, Portugal, June 13–14, 2017, pp. 1–7.

2. **Truc Anh N. Nguyen**, Siddharth Gangadhar, and James P.G. Seterbenz, "Performance Evaluation of TCP Congestion Control Algorithms in Data Center Networks," *11th International Conference on Future Internet Technologies (CFI 2016)*, Nanhing, China, June 15–17, 2016, pp. 21–28.

1. **Truc Anh N. Nguyen**, Siddharth Gangadhar, and James P.G. Sterbenz, "An Implementation of Scalable, Vegas, Veno, and YeAH Congestion Control Algorithms in ns-3," *Workshop on ns-3 (WNS3'16)*, Washington, USA, June 15–16, 2016, pp. 17–24.

1.7 ns-3 Implementation Models

I have contributed many models to the ns-3 community [1] as the results of my research work during my graduate studies.

1. Scalable TCP (STCP), 2016 (collaborate)
2. TCP Vegas, 2016
3. TCP Veno, 2016
4. Yet Another Highspeed TCP (YeAH), 2016
5. TCP Illinois, 2016 (collaborate)

[1] The community actively encourages submission of new features and models to ns-3. These submissions must follow the ns-3 coding and engineering guidelines and include documentation, tests, and examples before they are reviewed by other members [24].

6. TCP Westwood(+), 2013 (collaborate)

7. Burst error model, 2013

1.8 Organization

The rest of this dissertation is organized as follows: Chapter 2 presents a discussion of transport-layer algorithms and protocols that are adopted by or closely related to ResTP. In Chapter 3, we explain in details the design of ResTP including its header, modules, and the algorithms implemented inside each module. In Chapter 4, we evaluate the performance of ResTP by simulating the protocol with different applications and network types in comparison with conventional transport-layer protocols. Chapter 5 presents our thoughts on ResTP deployability. Finally, Chapter 6 concludes our dissertation with directions for future work. Details on the implementation verification and validation of ResTP in ns-3 is presented in Appendix A.

Page left intentionally blank.

Chapter 2

Background and Related Work

In this chapter, we discuss the different transport-layer services and algorithms that are related to or adopted by ResTP. We also give a summary of the transport protocols that guided the development of ResTP. The discussion highlights the drawbacks of the existing protocols and methods and present ResTP as a solution.

2.1 Connection Establishment

Connection management refers to the mechanism employed by a transport protocol to allocate, synchronize, and deallocate states while allowing the communicating parties to negotiate their operation modes and resources needed for their association. A reliable connection management, which can be achieved by combining different handshaking techniques, timers, and unique connection identifiers ensures a complete data transmission between end hosts with no ambiguity caused by data or acknowledgement duplications from either the current connection or previous ones [25]. Connection management schemes are classified into two main categories: connectionless and connection-oriented.

In the connectionless (CL) transport service, individual datagrams are exchanged between communicating hosts with no initial setup. Unless the protocol provides acknowledgement service for the transmitted data, which requires some state information to be retained at the sender, there is normally no information regarding to the association being maintained. Hence, the CL technique cannot protect the data from loss, missequencing, and duplication [26]. UDP is an example of a connectionless transport protocol.

In the connection-oriented (CO) transport service, user data are exchanged over a connection that can be explicitly or implicitly established and released with state information maintained at both endpoints to ensure that they are in synchronization during the communication lifetime. Based on the approach used to manage state information at end hosts (end-to-end (E2E) state management), the CO mechanism is categorized into three groups: timer-based, handshake-based, and hybrid. In the timer-based or pure soft state (SS) approach, timers are implemented at both ends to determine the state retention intervals, and all installed states will eventually timeout unless the end systems keep refreshing them. Because of the time-out characteristic, no explicit messages are required to remove state information. Delta-t [27], VMTP [28], and TP++ [15] are examples of transport-layer protocols that employ the timer-based technique. In the handshake-based (packet-exchange based) or pure hard state (HS) approach, communicating hosts are required to explicitly exchange control messages (signaling) to initiate and terminate a connection. At the transport layer, signaling may be accomplished in-band or out-of-band. While the former allows control and data messages to be multiplexed on the same connection, the latter transmits them on separate connections [26]. APPN [29–32] and Datakit [33–35] are a couple of examples that implement the pure HS approach. The hybrid approach incorporates both control messages and timers into the connection management. TCP and its variants, NETBLT [36], and HULA [37] are transport-layer protocols that employ the hybrid scheme.

A TCP connection is established using its well-known three-way handshake procedure with the exchange of SYN, SYN-ACK, and ACK messages before the actual data transmission. The use of any optional TCP options such as window scaling [38], timestamps [38], or selective acknowledgement (SACK) [39] is also negotiated through these signaling packets. A TCP connection is terminated using the graceful acknowledged FIN exchange method [40], which allows a full-duplex connection to close only when both ends have no more data to send while ensuring that all data transmitted before the FIN segments are fully received. Before tearing down the connection, the closing initiator (or initiators if both ends close simultaneously) enters a waiting period, which has the length of twice the maximum packet lifetime (MPL) to handle any outstanding data. Hence, TCP connection termination is a hybrid approach that employs both timers and explicit control messages.

Because TCP does not allow any application data to be transmitted during the establishment phase, it costs the protocol one whole round trip time (RTT) just to perform the connection setup. The TCP handshake especially adds a significant latency to short flows that normally terminate within a few RTTs. TCP Fast Open (TFO) [41, 42] addresses this issue by allowing communicating hosts to transmit and process data during the initial handshake after the client (connection initiator) obtains a security cookie from the server with that it wishes to communicate.

ResTP supports three connection establishment schemes: connectionless, three-way handshake, and ResTP opportunistic. The opportunistic approach improves the three-way handshake overhead while being resilient to SYN or SYN-ACK losses. Section 3.5.5 explains the ResTP connection establishment in more details.

2.2 Error Control

A reliable transport protocol that operates on top of an unreliable network layer needs to implement an error control mechanism to detect and correct (or request for a correction of) errors introduced into data packets when they are being transferred over error-prone channels. Error control algorithms are classified into 3 categories: ARQ (automatic repeat request), FEC (forward error correction), and HARQ (hybrid ARQ).

ARQ protocols provide reliable data transfer by requesting a retransmission of a corrupted packet after detecting an error in the received data. Fundamentally, in order to handle bit errors properly, ARQ protocols need to possess three capabilities: error detection, receiver feedback, and retransmission [43]. While the first two are implemented on the receiver side, the last one is employed by the sender of the protocol. There are multiple bit-level error detection techniques, and the simplest is parity checks. Single parity bit is able to detect a single bit error in a packet. When using this method, the sender appends an additional bit into the original d-bit data such that the total number of $(d + 1)$ bits is either even (even parity scheme) or odd (odd parity scheme). On the other end, the receiver counts the number of 1s in the received bits and checks against the scheme in use to detect an error. Two-dimentional parity allows the detection and correction of a single bit error in a packet by diving the original data into multiple rows and columns and appending a parity bit to each row and column as in the single parity scheme. Another error detection technique is checksumming. With this method, d-bit data are treated as k-bit integers, and the sum of these integers is used to detect bit errors. The Internet protocols, UDP, TCP, and IP, implement the checksumming method by treating data as 16-bit integers, calculating their sum, and performing the 1 s complement of the sum. However, while UDP and TCP compute checksum over both header and data fields, IP computes checksum over the IP header only. The receiver's operations involve calculating the sum of the received data together with the checksum and taking the 1 s complement of the total. The result is error-free if it contains all 1s. Finally, another widely-used bit-level error detection technique is CRC (cyclic redundancy check) or polynomial codes. CRC operation involves the use of an $(r + 1)$-bit generator G that is known by both the sender and receiver. The sender then appends an additional r-bit R to any d-bit data D that it wants to transmit so that the $(r + d)$-bit result is divisible by the generator G using modulo-2 arithmetic with no remainder as formulated in Equation 2.1 [43]. The receiver checks for errors by reverting the sender's operations: dividing the received $(r+d)$-bit data by the generator G. An error has occurred if the division gives a nonzero remainder. Comparing with the checksumming method, CRC is more powerful, but with higher overhead.

$$D \times 2^r \text{XOR} R = nG \tag{2.1}$$

The integrity of the data, after being inspected, is informed back to the sender by the receiver through an explicit feedback such as a positive ACK or negative ACK (NAK). An ACK is used if the data is error-free, and a NAK is used if the data is corrupted. Hence, a NAK is also a transmission request.

There are multiple common ARQ retransmission schemes including Stop-and-Wait, Go-Back-N, Fast Retransmit, and Selective Repeat. In the Stop-and-Wait approach shown in Figure 2.1, individual segment is transmitted and acknowledged (either ACKed or NAKed) before the next packet is sent. A sequence number is assigned to each packet, which allows the receiver to distinguish between an original and a retransmission of a packet in case both versions are correctly arrived at the receiver. One of the reasons that causes the sender to introduce a duplicate packet into the network is its reception of a corrupted ACK or NAK. Furthermore, communication channels can also lose or delay packets in addition to corrupt bits. Hence, a timer is used to trigger a retransmission in case the sender does not receive an ACK for the outstanding data. To prevent premature retransmission, the timer's value is supposed to be greater than the network's RTT (round-trip time). Obviously, this Stop-and-Wait approach introduces significant delay

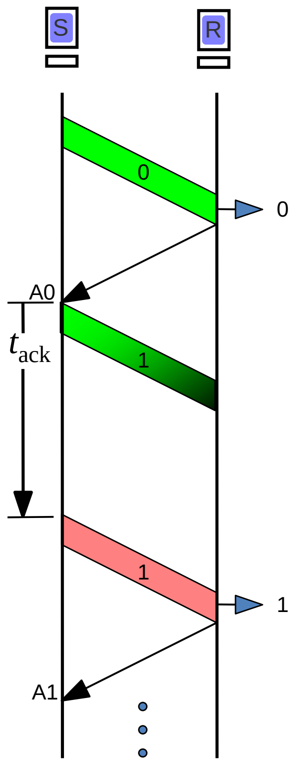

Figure 2.1: Stop-and-Wait Retransmission Scheme

and underutilization, especially in long-delay environments. A calculation shows that it takes 30.008 ms for a Stop-and-Wait sender located on the West Coast of the United States to transmit a single 1000-byte packet to (and receive its ACK from) a receiver located on the East Coast over a cross-country channel with an RTT of 30 ms and a transmission rate of 1 Gb/s. This is equivalent to the sender utilization of only 0.00027 and the effective throughput of only 267 kb/s [43].

The Go-Back-N approach (Figure 2.2) allows multiple packets to be in flight simultaneously. The Go-Back-N protocol is also referred as the sliding-window protocol since the sender operates over a window of N transmittable sequence numbers with the left boundary of this window sitting at the oldest unacknowledged sequence number and moving inward as the sender advances its transmission. The size of the window N places an upper limit on the number of outstanding packets before an ACK arrives. Similar to

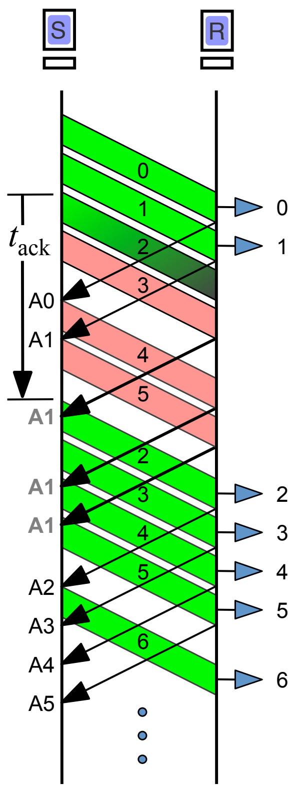

Figure 2.2: Go-Back-N Retransmission Scheme

the Stop-and-Wait scheme, each packet is associated with a retransmission timer that is set by the sender when the packet is placed in the network. The receiver is expected to save and cumulatively acknowledge in-sequence packets while retransmitting the previous ACK and discarding out-of-order packets. The sender views a duplicate ACK as an indication of data loss, but it does not retransmit the missing packet until the retransmission timer expires, upon which the sender goes back and resends all packets starting with the lost one informed by the duplicate ACK. Although the Go-Back-N with its pipeline transmission mechanism is an improvement of the Stop-and-Wait, it still suffers significant loss penalty in high bandwidth-$\times$-delay networks due to its retransmission policy. Given that high speed connections require large number of outstanding bytes at a given time to achieve full utilization, retransmitting all transmitted data that may have successfully reached the other end adds additional unnecessary RTTs without any

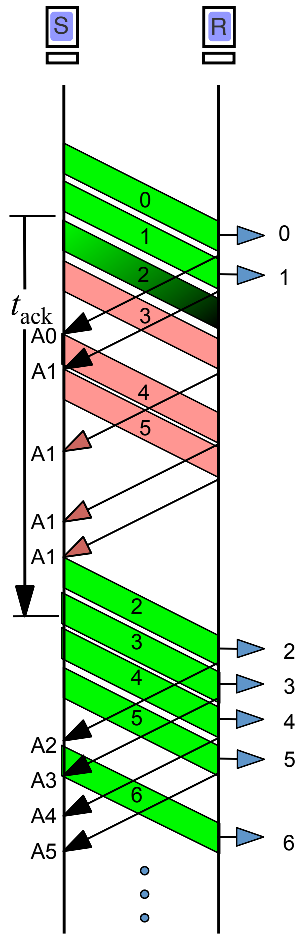

Figure 2.3: Fast Retransmit Retransmission Scheme

benefit gain. Furthermore, waiting for the timer to expire before retransmitting the lost segment delays the loss recovery, which is further intensified if the network links have high RTT.

The Go-Back-N shortcomings motivate the development of a new ARQ retransmission algorithm called Fast Retransmit. As shown in Figure 2.3, instead of waiting for the expiration of the retransmission timer, this algorithm uses the receipt of a certain number of duplicate ACKs (usually 3) as the trigger for the lost segment's resend, allowing more promptly loss recovery. However, Fast Retransmit still suffers from the Go-Back-N inefficiencies caused by the unnecessary retransmissions of all previously transmitted

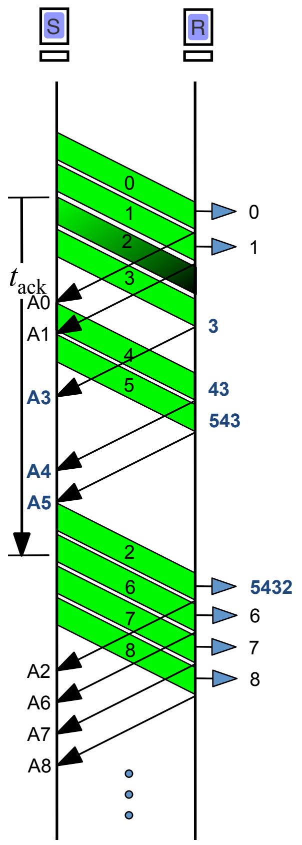

Figure 2.4: Selective Repeat Retransmission Scheme

data following the loss.

Selective Repeat (Figure 2.4) enhances the Go-Back-N retransmission policy by retransmitting only the data received in error (either corrupted or lost). This algorithm implies two requirements: (1) the receiver needs to buffer all received packets, including those that arrive out-of-order, and (2) the sender needs to know which packet is missing among all the currently outstanding ones. The first requirement increases the receiver's complexity and requires larger receiving buffer space when comparing with the Go-Back-N mechanism. In order to satisfy the second requirement, the receiver needs to acknowledge each correctly received packet individually, including the out-of-order packets.

Early TCP implementations [1] use the Go-Back-N approach as the retransmission scheme. After the occurrence of a series of congestion collapses in 1986, Fast Retransmit is introduced into TCP as one of the several new algorithms (slow start, round-trip-time variance estimation, congestion avoidance, and fast retransmit) to handle congested conditions [44]. TCP can perform Selective Repeat when employing the SACK (selective acknowledgement) option [39]. SACK is now the most common implementation.

Adopting the modular design, ResTP's error control is handled by multiple modules, including its reliability module discussed in Section 3.5.8 and ACK module discussed in Section 3.5.9. ResTP can provide different level of reliability depending on the techniques it employs in a specific configuration.

2.3 Congestion Control

Congestion in a packet-switching network happens when network resources such as communication links and memory buffers are saturated, resulting in significant delay of message delivery, data losses, waste of system resources, and possible network collapse [45]. From a user experience, this is a degradation in the quality of service provided. As bandwidth becomes cheaper due to the rapid growth of networking technologies, high-speed communication or high performance networking emerges when Internet service providers (ISPs) add bandwidth to their connections. Hence, modern congestion control is not only about avoiding congestion or keeping the queue occupancy small, but also on how to efficiently utilize all the available capacity to avoid wasting network resources while ensuring fairness among the competing flows [46].

A connection's maximum data delivery rate is determined by the slowest link (the bottleneck) of a path, and this bottleneck is fully utilized if the amount of inflight data (data sent but not acknowledged) matches exactly the bandwidth-$\times$-delay product (BDP), which is determined by Equation 2.2 [47] with b representing the available bottleneck bandwidth, and $\mathrm{RTT}_{\min}$ the minimum round-trip delay, which is controlled by the round-trip propagation time $\mathrm{RT}_{\mathrm{prop}}$. $\mathrm{RT}_{\mathrm{prop}}$ and b are the two physical properties of a path that bound transport performance [48].

$$\mathrm{bdp} = b \times \mathrm{RTT}_{\mathrm{min}} \tag{2.2}$$

If the amount of in-flight data is smaller than BDP, the bottleneck capacity is not fully utilized, and bandwidth is wasted. On the other hand, if the amount of in-flight data is larger than BDP, the bottleneck link is overloaded, and the (inflight − BDP) excess creates a queue. When this queue is exhausted, packets are dropped. Studies have shown that increasing the bottleneck queue's buffer size is not a solution to congestion because it leads to undesirable high latency and delay variation due to exceeded buffering, a phenomenon known as bufferbloat [49]. Furthermore, packet loss can occur despite of the maximum buffer capacity [46]. Hence, the goal of a congestion control mechanism is to determine the amount of data to transmit at a time t to match the inflight with the BDP. Unfortunately, endpoints do not have knowledge of the exact BDP. Instead, they need an algorithm employed in their congestion control to estimate the value.

Congestion control is more challenging when considering shared bottleneck. When multiple flows coexist on a bottleneck, it is not only desirable for the bottleneck to be fully utilized, but fairly shared among all flows. Towards this goal, a congestion control mechanism must be able to converge to a fair share over a shared bottleneck and maximize its throughput over an independent one.

Existing congestion control algorithms can be broadly divided into two main categories: open loop and close loop [45]. Close loop algorithms make their control decisions based on some feedback either from destination to source (global) or from intermediate neighbors (local). The feedback can be either implicit or explicit. Close loop congestion control algorithm with implicit feedback does not require feedbacks to be explicitly sent in any specific messages. Algorithms under this group include the standard TCP congestion control algorithm [50] and its derivatives. Close loop congestion control algorithm with explicit feedback requires feedback to be explicitly transmitted either as separate or piggybacked messages. Explicit feedbacks are further classified into persistent and responsive. While a persistent feedback is available at all times, a responsive feedback is only available in response to certain conditions. In summary, explicit congestion notification (ECN) algorithms are grouped into four main categories: close loop control with persistent and global feedback, close loop control with persistent and local feedback, close loop control with responsive and global feedback, and close loop control with responsive and local feedback. On the other hand, open loop congestion control algorithms make their control decision based only on their knowledge of local node such as local links' capacity and available buffers in the system. Open loop control can either be exhibited at the source or destination.

Congestion control algorithms can also be categorized based on the control method employed by an algorithm, the metric used to infer a congestion event, or the network domain for which the algorithm was designed.

When using the control method as the classification criterion, congestion control approaches can be divided into two categories: window-based and rate-based. Window-based algorithms use a window of maximum in-flight packets to control the transmission rate of the sender. The update of this window is dictated by the feedback from the receiver. Rate-based algorithms allow the sender to transmit at a specific rate until a new rate is informed by the receiver or a network component.

When using the congestion metric as a criterion, congestion control algorithms can be divided into four groups: loss-based, delay-based, hybrid, and congestion-based. Loss-based algorithms treat the loss of a packet as an indication of congestion. However, the assumption that congestion is the only source of data drops behind these loss-based approaches has been challenged by the growing adoption of wireless networks in which corruption, i.e., a bit error is more likely the cause of a loss. Some loss-based protocols are extended with a bandwidth estimation algorithm. Delay-based algorithms interpret the increasing delay due to queue growth as a congestion signal. It is important to note that queuing can only be the single factor that constitutes to packet latency if a burst of packets are assumed to be routed along the same path [46]. TCP Vegas [51], TCP Low Priority (TCP-LP) [52] are a couple examples of delay-based congestion control algorithms. Hybrid algorithms such as Compound TCP (CTCP) [53] and TCP Illinois [54] combine both loss- and delay-based schemes. Recently, Google introduces a new congestion-based congestion control scheme named TCP Bottleneck Bandwidth and Round-trip propagation time (BBR) that responds to actual congestion instead of packet loss or transient queue delay [48].

Transport-layer congestion control algorithms can also be classified based on the domain that they target. Original algorithms such as the standard TCP [50] was designed to prevent congestion collapses in wired networks. These early techniques operate with an assumption that all packet losses during a connection lifetime are caused by congestion. This assumption is challenged with the emergence of wireless environments that are characterized by high random-loss rate. As a result, the wired-based algorithms suffer significant performance degradation when being deployed in wireless networks because they trigger an unnecessary sending rate reduction when data packets are corrupted. This shortcoming is the main motivation behind many new proposals for wireless channels. The coexistence of both wired and wireless links in a network path rises another issue: the congestion control algorithm needs to distinguish the different sources of packet losses in such a heterogeneous environment. Furthermore, as mentioned previously, the emergence of high-speed communication or high performance networking requires congestion control algorithms to go beyond their original design goal of avoiding congestion or consistently keeping queue occupancy small to efficiently utilize the available capacity of transmission facilities. High bandwidth, when coupling with long delay such as in the space or interplanetary communication, places another demand on tranport-layer congestion control design: how to promptly react to congestion taking into account the much longer delay both data and feedback need to take when traversing the network. These requirements begin another generation of congestion control techniques for high BDP environment. When service prioritization among different traffic types is considered, another group of congestion control algorithms specifically designed for low priority data transfer emerges.

TCP Westwood [55] and TCP Veno [56] are a couple of congestion control algorithms designed to enhance TCP performance in wireless networks. TCP Hybla [57] addresses the RTT disparity problem occured in heterogeneous network. HighSpeed TCP (HSTCP) [58], Scalable TCP (STCP) [59], Binary Increase Congestion Control (BIC) [60], Yet Another Highspeed (YeAH) [61], and CUBIC [62] target high BDP environments. Incast congestion Control for TCP (ICTCP) [63] implements a receive-window-based congestion control algorithm for TCP to eliminate incast congestion in data center networks (DCNs).

A taxonomy of TCP congestion control techniques in both wired and wireless networks [64] also categorizes TCP variants based on the device entity that is responsible for handling the congestion control functionality. Using this criterion, TCP congestion control algorithms can be classified into four groups: sender, receiver, sender and receiver, and sender or receiver. In sender-centric protocol (SCP), the data sender performs the congestion control based on the receiver's feedback in acknowledgement (ACK) packets. SCP is the most deployed TCP technique. On the other hand, in receiver-centric protocol (RCP), the receiver fully controls the congestion control functionality. In RCP, the sender only transmits data upon the reception of an explicit request from the receiver. Moreover, the sender's transmission rate is also receiver-dictated. TCP-Real [65] is an

RCP example. Hybrid-centric protocol (HCP) is capable of operating the congestion control functionality at both sender and receiver sides. Detection of Out-of-Order and Response (DOOR) [66] and TCP friendly rate control (TFRC) [67] are hybrid-centric protocols. Finally, some variants such as the mobile-host control protocol (MCP) [68] can utilize either SCP or RCP depending on whether a mobile station is a sender or a receiver.

2.3.1 Multipath Congestion Control

As in single-path protocols, congestion control plays a crucial role in multipath transport-layer protocols. Three desirable properties of a practical multipath congestion control algorithm are identified when the LIA algorithm for MPTCP was developed (more details later) [19]:

1. Property 1 (Improve throughput): A multipath flow should perform at least as well as a single path flow would on the best of its paths.
2. Property 2 (Do no harm): A multipath flow should not take up more capacity from any path or collection of paths than if it was a single path flow using only the best path.
3. Property 3 (Balance congestion): A multipath flow should move as much traffic as possible off its most congested paths.

While properties 1 and 2 ensure fairness at the shared bottleneck, property 3 captures the resource pooling principle to increase robustness and maximize utilization [69].

Given that a large number of traffic flows in the current Internet are mice flows (i.e. short messages), a multipath flow should react promptly to this dynamic environment as these short flows come and leave the network. This requirement defines the fourth desirable property of a practical congestion control algorithm:

4. Property 4 (Responsiveness): A multipath flow should quickly adapt to network dynamics. Responsiveness measures the rate at which the congestion control algorithm is able to converge to its equilibrium [70].

In a study on the relationship between three performance metrics: friendliness, responsiveness, and window fluctuation using a fluid model, it is shown that the design of MPTCP algorithms involves inevitable tradeoffs, and one of them is the tradeoff between responsiveness and TCP-friendliness. Therefore, it is impossible to develop a superior algorithm for all of these metrics [70].

A multipath flow is also required to provide a Pareto-optimal allocation [20] of resources as the regular TCP always does. This leads to the definition of the fifth desirable property of a practical congestion control algorithm:

5. Property 5 (Pareto-optimality): It is impossible for one user to increase the throughput without penalizing another or without increasing the congestion cost [20].

Early versions of MPTCP and SCTP let each subflow control its own congestion window by running the standard AIMD TCP NewReno algorithm [50] independently. While this decoupling approach is simple, it leads to an important problem, TCP-unfriendliness. When sharing the bottleneck with TCP flows, a multipath connection that utilizes N subflows executing the decoupled algorithm is approximately N times as aggressive as each of the TCP flows [71]. As the failure of the decoupling mechanism in achieving TCP-friendliness validated in many studies, various multipath congestion control algorithms that employ sub-window correlation have been proposed. Many algorithms achieve TCP-friendliness by coupling the standard AIMD's increases of the subflows such as EWTCP [71], Coupled [72, 73], Semicoupled [74], or LIA [19, 74].

ResTP allows the integration of multiple congestion control algorithms into the protocol although for our analysis presented in this dissertation, only the NewReno and Coupled algorithms are implemented to be used in single-path and multipath ResTP connection, respectively (see Section 3.5.6 for more details).

2.4 Packet Scheduling

Packet scheduling plays a crucial role in the efficient performance of a multipath transport-layer protocol. A packet scheduler is responsible for distributing individual data packets among multiple available subflows, given the heterogeneous characteristics of the physical paths and the dynamics of the underlying network. For a multipath transport protocol that ensures in-order data delivery at the receiver, a wrong scheduling decision might lead to head-of-line blocking or receive-window limitation [75].

Many scheduling methods have been proposed and studied, including the round robin technique that selects a path in a round-robin fashion, the RTT-aware method that selects a path with minim RTT, or a scheduler that selects a subflow based on its current congestion situation [76].

ResTP currently supports two basic scheduling algorithms: round robin and best RTT. Section 3.5.10 explains ResTP packet schedulers in more details.

2.5 SCPS-TP and Space Communications

The advancements in networking technologies has introduced many network environments with distinct characteristics from the wired network for which the conventional

Internet transport protocols were tailored. One of those environments is the space communication environment. In this section, we discuss the characteristics of this network, the limitations of TCP, and one of the protocol design for space communications, SCPS-TP, from which we learn many techniques and apply to our ResTP design.

2.5.1 Space Communication Environment Characteristics and TCP Shortcomings

In space communications or in wireless communications in general, bit errors are common, and their rate can be very high (e.g., on the order of 10^{-1} for deep space missions [77]). Unfortunately, standard TCP always assume that a loss is caused by a congestion event. As the result, the protocol invokes its congestion control algorithm and unnecessarily reduces its transmission rate (by reducing its congestion window), causing a drop in its throughput.

Space communication channels are also known to be bandwidth-asymmetric with the forward link (from the spacecraft to the ground) capacity substantially larger than the return link (from the ground to the spacecraft) capacity. This assymmetry is typically on the order of at least 1000:1 in spacecraft missions [77]. TCP depends heavily on its ACK packets to determine its sending rate and requires an ACK to be generated for every other data packet received when no losses happen. Otherwise, the receiver has to acknowledge every packet following a loss. Due to the limited bandwidth available to the reverse channel, these ACK packets may congest the channel, resulting in delayed or lost ACK packets, and hence limiting the TCP throughput.

In addition, space communication channels are characterized by long propagation delay. The range from user to geostationary Earth orbit (GEO) satellite is a minimum of $36,000$ km, introducing a propagation delay of about 250 ms for a single hop between a user pair [78]. Deep-space (interplanetary) communication links even have much longer propagation delay. For example, the propagation delay from Earth to Mars is approximately 4 to more than 20 minutes, to Jupiter is approximately 30 to 45 minutes, to Saturn is between 70 to 90 minutes, depending on their orbital locations [77]. This long delay means several things to TCP. First, the one-RTT overhead introduced by its 3-way handshake now becomes a more prominent problem. TCP also needs to stay longer in its slow start phase, during which the available link capacity is not fully utilized. For GEO satellites, the time required by the slow start phase to reach a bit rate of 10 Mb/s when using average packet length of 1000 bits is around 5.73 seconds even when the receiver is required to acknowledge every received segment. For short flows, it is possible that the entire data transfer only happens in the slow start phase [79]. Furthermore, long propagation delay causes ACK packets longer time to get back to the data transmitter, delaying the transmitter's response to loss events when they happen.

In space communications, there is no assurance that a connectivity is episodic. Within the Interplanetary Internet, a significant source of periodic link outages is orbital obscurations, in which communicating systems lose line-of-sight because of moving planetary bodies [77]. TCP does not have any mechanism to properly handle a temporary link outage. Without a steady flow of acknowledgments, the TCP sender will assume that a congestion event has happened, invoke its congestion control algorithm, reduce its sending rate, retransmit data packets, and back-off its retransmission timer. The consequence of this behavior is a significant degradation in its throughput, if not a connection abortion when TCP reaches its retransmission threshold before the link is restored.

These characteristics of space communication channels and the limitations of TCP are discussed in many research work [77, 79, 80].

2.5.2 SCPS-TP

Satellite Communication Transmission Protocol (SCPS-TP) [80] comprises of a set of extensions to TCP to improve its performance in the space communication environment. Each extension is a solution to the shortcoming of TCP caused by a specific characteristic of the space network. Instead of assuming that every loss is congestion-based and always invoking its congestion control algorithm, SCPS-TP implements a distinct approach to respond to the three possible sources of loss in space communication: congestion, corruption, and link outage. To cope with congestion-induced losses, SCPS-TP utilizes the TCP Vegas congestion control algorithm [51]. However, if a loss is caused by corruption, the protocol does not invoke TCP Vegas. Hence, the sending rate is not reduced, and the retransmission timer is not backed-off. When a loss is caused by a link outage, SCPS-TP enters persist mode, suspends all timers and ceases all transmissions. Instead, the protocol only sends periodic probe packets. Determining cause of a loss is difficult. SCPS-TP implements two approaches to determine the source of packet loss. With the first approach, it uses a parameter that can be set by a network manager. This parameter has its default value set to 'corruption' because bit errors happen more often in space communication than congestion. With the second approach, SCPS-TP learn about the source of loss through explicit signaling from intermediate nodes.

SCPS-TP addresses the shortcoming of TCP when transmitting over asymmetric channels by modifying the acknowledging frequency. Instead of sending an ACK every other receive segment when no losses happen and immediately sending an ACK for every segment received when losses happen, SCPS-TP sends an ACK every configurable period of time. This ACK-delay duration is related to the RTT estimation. In addition, header compression is implemented in SCPS-TP as a second technique to enhance the protocol throughput on asymmetric channels. Header compression is also a mechanism used by

SCPS-TP to improve performance on bandwidth-constrained channels.

Furthermore, SCPS-TP employs a new acknowledgment method, selective negative acknowledgment (SNACK). The format of SNACK allows the receiver to inform multiple losses to the sender in one segment (similar to SACK [39]), which is beneficial in high bit-error rate environment, but is more bit-efficient than SACK.

2.6 Transport Protocols for Real Time Multimedia Applications

One of the most known protocols for real-time conversation applications is the real-time transport protocol (RTP) [81]. RTP provides end-to-end delivery services for audio and video data, including payload type identification, sequence numbering, timestamping, and delivery monitoring. However, the protocol does not provide any mechanism to ensure timely data delivery or other quality-of-service (QoS) guarantees. RTP typically runs on top of UDP to make use of UDP's multiplexing and checksum services. Every media chunk is encapsulated inside an RTP packet, which is in turn encapsulated in a UDP segment before being passed down to IP.

Using the Internet for the delivery of real time multimedia applications prompts the developers to select between UDP and TCP. Those who choose UDP prefer its light-weight design with no latency introduced by the protocol when transmitting data. Packets are immediately delivered to the application without worrying about its ordering. Lost packets are simply ignored without the need for retransmissions. These properties make UDP more attractive to delay-sensitive real-time application than TCP, which favors reliability over timeliness.

While UDP has been more favorable for real-time multimedia applications, researchers have learned that packet losses can have a significant impact on the performance of these applications. Real-time video and audio streaming is actually more sensitive to network loss than jitter for two reasons: First, these applications tend to use a large playback buffer (at least in the order of hundreds of milliseconds), which can insulate the playback from the network jitter. Second, the contents of these applications are typically highly compressed to reduce the required data rate, a single loss of an important packet may cause a noticeable disruption in the content reception [82]. Many real-time applications rather receive damaged packets with a few single-bit errors than losing a full packet [83]. As the result, many researchers have proposed some modifications to UDP to address some level of packet losses while others promote the use of TCP (with proper extensions and modifications) to take full advantage of the protocol's error recovery and congestion control.

UDP Lite [83] promotes the use of partial checksum in UDP to prevent packets with a few acceptable bit errors from being dropped. The protocol replaces the existing UDP header field Length with a new Coverage field, which specifies the amount of sensitive data the sending application wants UDP Lite to include in the checksum calculation. Any errors occur outside of the sensitive part of the data are ignored, and the whole packet is passed to the receiving application as a normal packet. For this mechanism to work, the link layer checksum or CRC needs to be disabled.

UDP-Liter [84] takes advantage of the concept proposed in UDP Lite but addresses the two issues associated with the protocol: its backward incompatibility with traditional UDP and its lack of flexibility for the receiving application to choose how it wants to handle corrupted packets. To avoid modifying the UDP header, UDP-Liter allows the sending application to convey its wish to discard a keep a packet when the checksum fails through a new parameter in its BSD socket() function call. Similarly, UDP-Liter uses a new parameter passed in the BSD recvfrom() function call to differentiate a damaged packet from a normal one when it reaches the receiving side. This is useful when the receiver wants to handle the two types of packets differently.

Research work on TCP for real-time multimedia applications are the various solutions for the common problem: How to modify or extend TCP so that the protocol can provide both reliable and timely data delivery while keeping its congestion control benefit intact?

TCP-RTM [82] extends TCP with a real-time mode (RTM) to support real-time traffic. The extension allows the receiver to read beyond a hole in its receiving buffer and accept the out-of-order data following that hole. With this modification, the receiver is never blocked from waiting for the hole to be filled by a retransmission. On the sending side, TCP-RTM allows the sender to discard the oldest data segment queuing in the send buffer waiting to be acknowledged or transmitted if the buffer is full. In order for the application to determine which frames are lost and skipped over by the receiver, TCP-RTM employs multiple framing techniques to ensure that the application-level frame boundaries always align with TCP segment boundaries for applications with fixed- and variable-sized frames. Moreover, the authors propose two application-level techniques to maximize the benefits of TCP-RTM: a TCP-sized playback buffer that allows TCP to recover from a packet loss using its fast retransmit within the playback delay, and a heart-beat packet transmission when the receiver has not received any new data from the sender for some period of time. This heart-beat message serves as an affirmation from the receiver that all data have successfully received so the sender can continue transmitting new data. The message is significantly useful when an ACK loss suspends the sender's transmission due to its congestion window constraint, especially during slow start.

ResTP implements multiple algorithms that can be employed for real-time multimedia traffic including its partially reliable and partially ordering techniques. The performance of these algorithms and how they are configured in ResTP are discussed in Section 4.5.

Page left intentionally blank.

Chapter 3

ResTP Design

In this chapter, we present the design of ResTP, starting with its header and header's extensions, the services that it provides as a transport-layer protocol in both unipath and multipath modes, and the algorithms that implement each of those services. As highlighted in Chapter 1, our design of ResTP follows the set of design principles established in the ResiliNets framework [11], and is guided by our studies and analysis of well-known transport protocols, especially TCP, UDP, TP++, SCPS-TP, and MPTCP to achieve resilience and survivability.

3.1 ResTPDU

Given that ResTP is a composite protocol that can operate in multiple modes with different configurations, it is essential for its header to be extensible, and the header fields to be added only when necessary to minimize the overhead. We start with the basic (the minimal) ResTP transport protocol data unit (ResTPDU) format shown in Figure 3.1 [1], which is utilized when ResTP operates as a lightweight and unreliable protocol similar to UDP. We then define multiple header extensions that allow the protocol to provide more complicated services to the upper application layer.

[1]Packet headers are shown using the IETF standard method.

3.1.1 Basic ResTPDU

The header fields are constructed with the most significant byte first, which means that the version field is located at the four most significant bits of the first byte in our header.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  HL   |     Flags     |Ord| Flow  | P |     Error     |
|       |       |-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       |       |C|E|U|A|P|R|S|F|P|F|C|C|O|C|M|M| |E|A|C|F|N|S|D|
|       |       |W|C|R|C|S|S|Y|I|A|U|K|O|P|X|P|P| |N|R|R|E|A|A|E|
|       |       |R|E|G|K|H|T|N|N|R|L|I|N|T|F|?|M| |C|Q|C|C|K|K|L|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Cong  |  FC   |       |       | Sched |                       |
|-+-+-+-+-+-+-+-|   k   |       |-+-+-+-|                       |
|       |     |W| (# of | Resv. |   |F|R|                       |
|BitVect|     |I| paths)|       |   |R|R|                       |
|       |     |N|       |       |   |R|S|                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Source Port Number       |   Destination Port Number     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                Payload (variable length)                      /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.1: Basic ResTPDU

- **Version**: 4 bits (mandatory, unsigned integer)

  The ResTP version number. For the current implementation, this field has a value of 0.

- **HL**: 4 bits (mandatory, unsigned integer)

  The ResTP header length (the number of 32-bit words in the header).

- **Flags**: 8 bits (mandatory, unsigned integer)

  The ResTP flags. This field includes 2 Explicit Congestion Notification (ECN) bits

(`CWR` and `ECE`) [3] and 6 TCP flags [1]. In ResTP, they have the same meanings as in TCP, with `ACK`, `RST`, `SYN`, and `FIN` used for connection establishment and termination. These flags are also used for TCP splicing and TCP segment translating when ResTP is deployed at the gateways in the future implementation of ResTP.

- **Ord**: 2 bits (unsigned integer)

  The ordering service in operation.

  `PAR`: partially-ordered data delivery

  `FUL`: ordered data delivery

- **Flow**: 6 bits, 4 in use (unsigned integer)

  The flow management paradigm in operation.

  `CKI`: cookie flag used in cookie request/response to set up opportunistic connection establishment

  `CON`: connection-oriented connection establishment paradigm

  `OPT`: opportunistic connection establishment paradigm

  `CXF`: custody transfer at realm gateways

  Note that some of these bits, when set, indicate the use of associated extensions.

- **P**: 2 bits (mandatory, unsigned integer)

  The path mode (uni-path or multi-path) in operation.

  `MP?`: ResTP is in single-path (multi-path) mode if this bit is 0 (1).

  `MPM`: ResTP is in hot-standby multi-path mode if this bit is 0, and the protocol is in spreading multi-path mode if this bit is 1. Note that the value in `MPM` is valid only when `MP?` is set to “1”.

- **Error**: 8 bits, 6 in use (mandatory, unsigned integer)

  The error correction mode and the corresponding acknowledgment packet format in operation.

`ENC`: FEC-encoded

`ARQ`: automatic repeat request

`CRC`: CRC-32 cyclic redundancy code

`FEC`: forward error correction

`NAK`: negative acknowledgment

`SAK`: selective acknowledgment

`DEL`: delayed ACK

Note that some of these bits, when set, indicate the use of associated extensions. When both `SAK` and `NAK` are set to "1", it signifies the Selective Negative Acknowledgment (SNACK) mechanism originally introduced in SCPS-TP [85, 86].

- **Congestion (Cong)**: 4 bits (unsigned integer)

  The bit vector specifying the congestion control algorithm in operation. This 4-bit field can specify up to (2^4 - 1) different single-path congestion control algorithms and (2^4 - 1) different multipath congestion control algorithms depending on the `MP?` mode. We use bit vector to accommodate the plethora of proposed algorithms should ResTP be used to study congestion control techniques. In the current implementation, ResTP supports NewReno and BIC.

  0001: NewReno (single path) / NewReno (multipath)

  0010: BIC (single path)

- **FlowControl (FC)**: 4 bits (mandatory, unsigned integer)

  The flow control technique in operation.

- **k**: 4 bits (mandatory, unsigned integer)

  The number of paths that are used for data transfer. When `MP?` in **P** is 0, **k** has a value of 1. When `MP?` is 1, the value in **k** is greater than 1.

- **Reserved**: 4 bits (mandatory, unsigned integer)

  Reserved for future use; must be 0.

- **Source port number**: 16 bits (mandatory, unsigned integer)

  The source port number. The 4-tuple (source port number, source IP address, destination port number, destination IP address) identifies the connection (association) to which the ResTPDU belongs.

- **Destination port number**: 16 bits (mandatory, unsigned integer)

  The ResTP destination port number. The receiver uses this port number to demultiplex the ResTPDU to the correct receiving application.

- **Payload**: variable length

  The application data transmitted in the ResTPDU.

3.1.2 CRC Extension

```
0                   1                   2                   3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       ...                     |         HEC CRC-16            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Payload CRC-32                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.2: CRC-extension

The CRC extension depicted in Figure 3.2 is appended to the basic ResTPDU when cyclic redundancy check is enabled (`CRC` bit in **Error** is set to 1) to detect bit errors on both the header and the payload.

- **HEC CRC-16**: 16 bits (unsigned integer)

  The integrity check for the header using cyclic redundancy check with a 16-bit generator.

- **Payload CRC-32**: 32 bits (unsigned integer)

  The integrity check for the data using cyclic redundancy check with a 32-bit generator. This field only presents if the ResTPDU carries some payload.

CRC is used for ResTP instead of checksum because unlike TCP that was tailored for wired networks, ResTP is also designed for wireless environments that are known to have higher error rates. The generator polynomials G(x) used to compute the CRCs are implementation-specific.

3.1.3 Flow Control Extension

```
0                   1                   2                   3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       ...                     |          Window               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.3: flow-control-extension

The flow control extension (Figure 3.3) is appended to the header whenever flow control is enabled.

- **Window**: 16 bits (unsigned integer)

  The number of segments beginning with the one indicated in the acknowledgment field which the sender of this ResTPDU is willing to accept.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                             Cookie                            /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.4: Cookie response extension

3.1.4 Cookie Response Extension

The cookie response extension is used in the client-authentication process of an opportunistic connection (see Section 3.5.5 for more details).

- **Cookie**: 32 to 128 bits (4 to 16 bytes) (unsigned integer)

  The **Cookie** has a non-zero value and is sent by a server (this ResTPDU's source) in response to a previous client's request (this ResTPDU's destination).

3.1.5 FEC Extension

To support the FEC (forward error correction) error control mechanism, the basic header is extended with the **FEC Type** and **FEC Range** as depicted in Figure 3.5. FEC usage is negotiated using the `FEC` bit in the **Error** header field.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   FEC Type    |               FEC Range                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.5: FEC extension

- **FEC Type**: 8 bits (unsigned integer)

  The FEC encoding scheme used for this connection. Table 3.1 describes the schemes

Value	**Type**	**Description**
0	Undefined	
1	Basic XOR	Payload of encoded packet is the XOR of payloads of every packet, up to 16 packets in 1 encoded packet.
2	Interleaved XOR	Payload of encoded packet is the XOR of payloads of every other packets, up to 8 packets in 1 encoded packet.

Table 3.1: FEC encoding types

currently supported by ResTP and their corresponding values in this field. The basic XOR and interleaved XOR are the two FEC types designed for TCP-IR (TCP Instant Recovery) [87, 88]. ResTP allows the employment of any other FEC schemes.

- **FEC Range**: 24 bits (unsigned integer)

  The number of packets encoded in the payload. When FEC is used, this field is only valid if the `ENC` bit is set. The field has an additional meaning when HARQ is used.

3.1.6 Sequence Number Extension

```
0                   1                   2                   3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Sequence Number                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.6: Sequence number extension

- **Sequence Number**: 32 bits (unsigned integer)

  The sequence number assigned to this ResTPDU.

```
0                   1                   2                   3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Sequence Number                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Acknowledgement Number                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.7: ARQ extension

3.1.7 ARQ Extension

To support the ARQ (automatic repeat request with retransmissions) error control mechanism using positive ACK (acknowledgement), the basic header needs to be extended with the **Sequence Number** and **Acknowledgement Number** fields as depicted in Figure 3.7. ARQ usage is negotiated using the `ARQ` bit in the **Error** header field.

- **Sequence Number**: 32 bits (unsigned integer)

  The sequence number assigned to this ResTPDU.

- **Acknowledgement Number**: 32 bits (unsigned integer)

  The acknowledgement sequence number (RCV.NXT) assigned to this ResTPDU. If this ResTPDU is not an ACK, it carries the sequence number 0 in this field. If this ResTPDU is an ACK with the `ACK` bit in the **Flags** field set, it carries the next sequence number the sender of this ResTPDU is expecting to receive.

Note that this extension is mandatory when the ARQ error control is employed, regardless of the ACKing mechanism. In addition, it must precede the NACK, SACK, and SNACK extensions when they are attached to the same ACK packet.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Left Edge of First Block                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Right Edge of First Block                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                            ...                                /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Left Edge of nth Block                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Right Edge of nth Block                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.8: ARQ with SACK extension

3.1.8 SACK Extension

Following the TCP SACK (selective acknowledgement) option definition [39], our SACK extension for ResTP has the format shown in Figure 3.8. Although ResTP does not impose any limitations on the maximum length of its extension, given the format of SACK that requires 64 bits to inform a single block of received data, a ResTP implementation may limit the maximum number of SACK blocks that can be appended to a packet. SACK usage is negotiated using the `SAK` bits in the **Error** header field.

- **Left Edge of ith Block**: 32 bits (unsigned integer)

  The sequence number of the first segment in the i^{th} block being reported in this ResTPDU.

- **Right Edge of ith Block**: 32 bits (unsigned integer)

  The sequence number of the last segment in the i^{th} block being reported in this ResTPDU.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Hole1 Offset            |          Hole1 Size           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               \
/                   Bit Vector (variable length)                /
\                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.9: ARQ with SNACK extension

3.1.9 SNACK Extension

The ResTP's SNACK (selective negative acknowledgment) extension (Figure 3.9) has the fields presented in the SCPS-TP SNACK option [86], including **Hole1 Offset**, **Hole1 Size**, and the optional **Bit Vector**.

- **Hole1 Offset**: 16 bits (unsigned integer)

  The offset (in packet units) from the current acknowledgment number of the first hole being reported in this ResTPDU. The offset is calculated by subtracting the current acknowledgment number from the sequence number of the first packet in the hole using integer arithmetic.

- **Hole1 Size**: 16 bits (unsigned integer)

  The size (in segment units) of the first hole that is reported in this ResTPDU.

- **Bit Vector**: variable length

  Information about additional segments following the first hole (Hole1) being reported in this ResTPDU. A "0" represents a missing segment while a "1" represents a received segment. Zeros are padded to the right of the last "1" when needed to ensure the **Bit Vector** ending on an octet boundary, and these zeros shall be ignored when processing the **Bit Vector** [86].

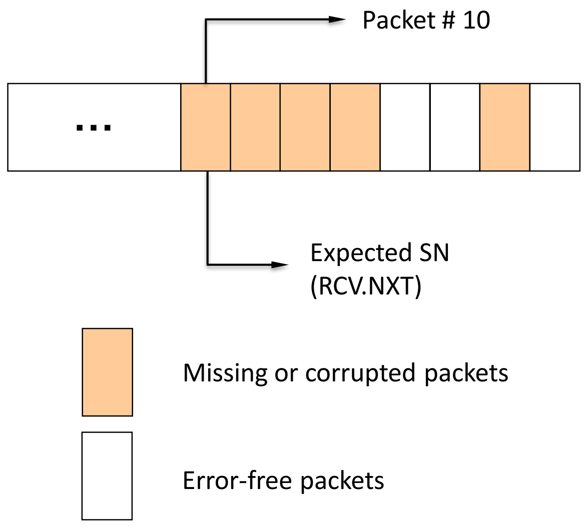

Figure 3.10: Example of a reordering queue

To demonstrate the difference between SACK and SNACK in ResTP, we provide an example to illustrate the use of each given the out-of-sequence receiving buffer shown in Figure 3.10.

We assume that all segments from 1 to 9 have been received correctly, and after that the sender transmits a burst of 8 segments. In this burst, segments 10, 11, 12, 13, and 16 are lost, and segments 14, 15, and 17 successfully arrive at the receiver. We assume that the ResTP receiver will send an ACK for every out-of-order packet received.

Upon receiving segment 14, the receiver generates the following NACK, SACK, and SNACK depending on which ACK mechanism is negotiated at the beginning at the connection. The **Acknowledgment Number** field carries a value of 10 because the receiver is waiting for the 10$^{\text{th}}$ segment after having received all of the first nine segments correctly.

Figure 3.11 shows how the receiver reports the out-of-order block of received data in a SACK format. The equal values in the left and right edge indicate that this block has only one segment.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Left edge of 1st block = 14                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Right edge of 1st block = 14                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.11: SACK extension upon the arrival of packet 14

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           0                   |             4                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    10000000   |                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.12: SNACK extension upon the arrival of packet 14

Figure 3.12 shows the values inside SNACK fields to report the missing segments when 14 arrives.

Upon the arrival of 15, another out-of-packet, the receiver generates another duplicate ACK with the **Acknowledgment Number** field again having a value of 10.

This time, the right edge of the first block, which is the only block of the SACK extension has a value of 15, instead of 14 (Figure 3.13).

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Left edge of 1st block = 14                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Right edge of 1st block = 15                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.13: SACK extension upon the arrival of packet 15

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           0                   |             4                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    11000000   |                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.14: SNACK extension upon the arrival of packet 15

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Left edge of 1st block = 17                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Right edge of 1st block = 17                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Left edge of 2nd block = 14                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Right edge of 2nd block = 15                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.15: SACK extension upon the arrival of packet 17

When 15 arrives, the only change in the SNACK extension is the **Bit Vector** field (Figure 3.14).

Upon the arrival of 17 (16 is lost as in our assumption), the receiver transmits another duplicate ACK with **Acknowledgment Number** having a value of 10. Attached to this ACK is a SACK, or SNACK extension.

According to the SACK policy [39], the receiver must arrange the block that is most recently formed by the newly coming packet first. That is why we have the value 17 in both the left and right edge of the first block as shown in Figure 3.15.

For SNACK, the only field that has a different value is **Bit Vector** as shown in Figure 3.16.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           0                   |             4                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    11010000   |                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.16: SNACK extension upon the arrival of packet 17

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   FEC Type    |                 FEC Range                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Sequence Number                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgement Number                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.17: HARQ extension

3.1.10 HARQ Extension

To support the HARQ (hybrid ARQ) error control mechanism, the basic header is extended with both the ARQ and FEC extensions (Figure 3.17. HARQ usage is negotiated using the `ARQ` and `FEC` bits in the **Error** header field.

- **FEC Range**: 24 bits (unsigned integer)

  If the `ENC` bit is set, this field specifies the number of segments encoded in the payload of this ResTPDU. If the `ENC` bit is not set, but the `ACK` bit is set, this field specifies the number of segments that cannot be recovered by the encoded packet that triggers this RESTPDU transmission.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  ...                          |A|B|C|D|E|F|G|H|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                        Sender's Key                           |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                         Receiver's Key                        |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.18: MP_CAPABLE extension

3.1.11 MP_CAPABLE Extension

The MP_CAPABLE extension is used to initiate a multipath ResTP connection. MP_CAPABLE is present in the SYN, SYN-ACK, and ACK of the three-way handshake to establish the first (master) subflow. The format of this extension shown in Figure 3.18 is similar to the MPTCP's MP_CAPABLE option's format in RFC 6824 [18].

- **Flags**: 8 bits (mandatory, unsigned integer)
  The MP_CAPABLE extension's flags. These flags have the same meanings as those in the corresponding MPTCP option.
  `A`: This bit, when set to "1", indicates that the integrity checks for both header and payload are required.
  `B`: This bit is an extensibility bit for future use. For the current ResTP implementation, `B` must have a value of 0.
  `C` through `H`: These bits are used to negotiate the crypto algorithm to be used. One of these bits must be set to "1", which means that a crypto algorithm must be specified. Otherwise, the extension is ignored. As in MPTCP, the current ResTP

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  ...                |     |B|  Address ID   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Receiver's Token                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Sender's Random Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.19: MP_JOIN extension for initial SYN

implementation only supports the HMAC-SHA1 algorithm, which is indicated by setting the `H` bit to 1.

- **Sender's key**: 64 bits (mandatory for the three-way handshake's `SYN` and `ACK` packets, unsigned integer)
  A key generated by the connection's initiator that is used to authenticate subsequent subflows.

- **Receiver's key**: 64 bits (mandatory for the three-way handshake's `SYN-ACK` and `ACK` packets, unsigned integer)
  A key generated by the connection's responder that is used to authenticate subsequent subflows.

3.1.12 MP_JOIN Extension

The MP_JOIN extension is used to add a new subflow to an existing multipath ResTP connection, and is only present in the three-way handshake's SYN, SYN-ACK, and ACK. The format of MP_JOIN is similar to the MPTCP MP_JOIN option's format [18].

The MP_JOIN format for the SYN packet is illustrated in Figure 3.19.

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  ...                  |     |B|  Address ID   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                 Sender's Truncated HMAC                       |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Sender's Random Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.20: MP_JOIN extension for SYN-ACK

- **Flags**: 4 bits, 1 in use (mandatory, unsigned integer)

  The MP_JOIN flags.

  B: backup bit indicating the subflow is used as a backup only when it is set to "1"

- **Address ID**: 8 bits (mandatory, unsigned integer)

  The source address of this MP_JOIN packet.

- **Receiver's Token**: 32 bits (mandatory, unsigned integer)

  The token used to identify the connection to which the subflow wants to join. This is a cryptographic hash of the MP_JOIN receiver's key exchanged in the initial MP_CAPABLE handshake. For the current ResTP specification, the token is the most significant 32 bits of the result generated using the SHA-1 algorithm.

- **Sender's random number**: 32 bits (mandatory, unsigned integer)

  Random numbers (nonces) generated by the sender of this MP_JOIN to prevent relay attacks.

The MP_JOIN format for the SYN-ACK packet is illustrated in Figure 3.20.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
|                         Sender's HMAC                         |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.21: MP_JOIN extension for ACK

- **Sender's truncated HMAC**: 64 bits (mandatory, unsigned integer)

  The truncated (leftmost 64 bits) HMAC generated by the sender of this MP_JOIN packet.

- **Sender's random number**: 32 bits (mandatory, unsigned integer)

  A random number generated by the sender of this MP_JOIN packet.

The MP_JOIN format for the ACK packet is illustrated in Figure 3.21.

- **Sender's HMAC**: 160 bits (mandatory, unsigned integer)

  The HMAC generated by the sender of this MP_JOIN packet

3.1.13 Data Sequence Signal (DSS) Extension

The Data Sequence Signal (DSS) extension has two purposes. First, it is used to define the mapping between subflow-level sequence space to the connection level to ensure in-order data delivery to the recipient application. Second, it is used as a connection-level ACK, which is called Data ACK in MPTCP terminology. The format of DSS is similar to the MPTCP DSS option's format [18].

```
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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         |F|M|A|      Data-Level Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Data ACK                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Data Sequence Number (DSN)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Subflow Sequence Number (SSN)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Payload CRC-32                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3.22: DSS extension

- **F**: 1 bit (mandatory, unsigned integer)

  The flag used to indicate "DATA_FIN", which is the connection-level FIN. The enabling of this flag implies that the mapping covers the last data sent from the sender.

- **M**: 1 bit (mandatory, unsigned integer)

  The flag used to indicate that the packet contains a data sequence mapping, which in turn implies the presence of Data Sequence Number (DSN), Subflow Sequence Number (SSN), Data-Level Length, and Payload CRC-32 fields.

- **A**: 1 bit (mandatory, unsigned integer)

  The flag used to indicate the presence of Data ACK.

- **Data-level length**: 16 bits (mandatory, unsigned integer)

  The length (in packets) for which this mapping is valid.

- **Data ACK**: 32 bits (mandatory, unsigned integer)

  Data-level acknowledgment number.

- **Data sequence number (DSN)**: 32 bits (mandatory, unsigned integer)
  The starting data-level sequence number for this mapping.

- **Subflow sequence number (SSN)**: 32 bits (mandatory, unsigned integer)
  The starting subflow-level sequence number for this mapping.

- **Payload CRC-32**: 32 bits (mandatory, unsigned integer)
  The integrity check for the payload covered in this mapping if the use of payload integrity check has been negotiated at the establishment of the first subflow of this multipath ResTP connection.

3.2 ResTP Extension Precedence

Unlike TCP options, ResTP extensions do not have the **Kind** field to specify the type of each extension as well as marking the beginning and the end of each extension when they are appended to the basic header. Hence, in ResTP, the flags in the basic header plays a very important role in letting the endpoints know what extensions they should expect in the header. Moreover, ResTP requires that once a service is negotiated at the beginning of a connection, all subsequent packets must carry the corresponding flags. For example, the `OPT` bit signifying the use of the opportunistic connection management scheme has to be ON in the initial setup messages (SYN, SYN-ACK, and ACK) and in all of the following packets. If the hosts have agreed to use FEC as the error control technique, all packets exchanged during the communication must have the `FEC` bit in **Error** set to "1".

Furthermore, a ResTP segment may contain multiple extensions, it is necessary for ResTP to establish an extension precedence rule as shown in Table 3.2 so that the hosts know how to extract and deserialize a packet's header.

Precedence	**Extension**
1	basic header (Figure 3.1)
2	cookie response (Figure 3.4)
3	MP_CAPABLE (Figure 3.18, MP_JOIN (Figures 3.19, 3.20, 3.21))
4	DSS (Figure 3.22)
5	FEC (Figure 3.5, ARQ (Figure 3.7), HARQ (Figure 3.17)
6	SACK (Figure 3.8), SNACK (Figure 3.9)

Table 3.2: ResTP extension precedence

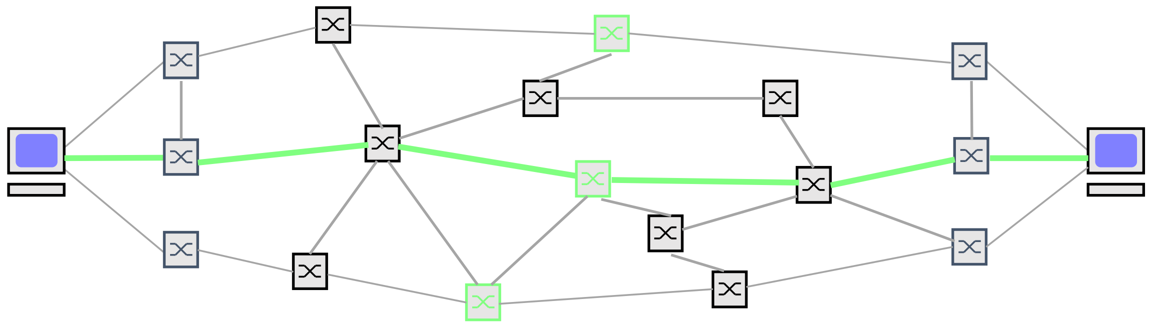

Figure 3.23: ResTP single-path communication

3.3 Single- and Multi-Path Communication Modes

ResTP is both a uni- and multi-path transport-layer protocol. The end-systems negotiate the communication mode that they want to operate in using the main header's **P** field.

3.3.1 Single-Path Communication Mode

ResTP is in its unipath E2E communication (Figure 3.23) when both MP? and MPM are unset and the total number of paths used for data transfer specified in **k** has a value of 1 throughout the whole connection lifetime.

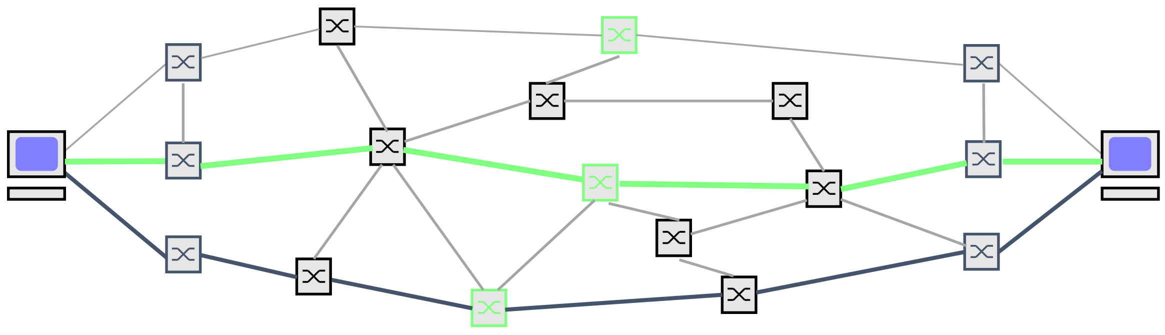

Figure 3.24: ResTP multi-path hot-standby communication

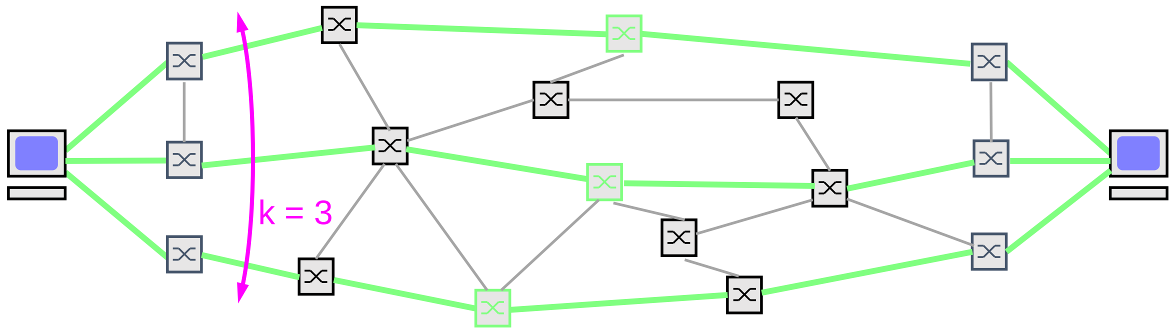

Figure 3.25: ResTP multi-path spreading communication

3.3.2 Multi-path Communication Mode

ResTP is in its multipath E2E communication when `MP?` is set to "1", and the value in **k** is greater than 1. Furthermore, ResTP supports two multi-path sub-modes (or subflow policies): hot-standby (`MPM` = 0) shown in Figure 3.24 and spreading (`MPM` = 1) shown in Figure 3.25

When operating in hot-standby, the protocol transmits data on one primary path and only utilizes a different path in the set of secondary paths that are in hot-standby state when the main one fails as illustrated in Figure 3.26. Comparing to the traditional single-path communication, this mode allows faster restoration when the primary path fails, yet with the possibility of some data losses. The degree of losses depends on how fast

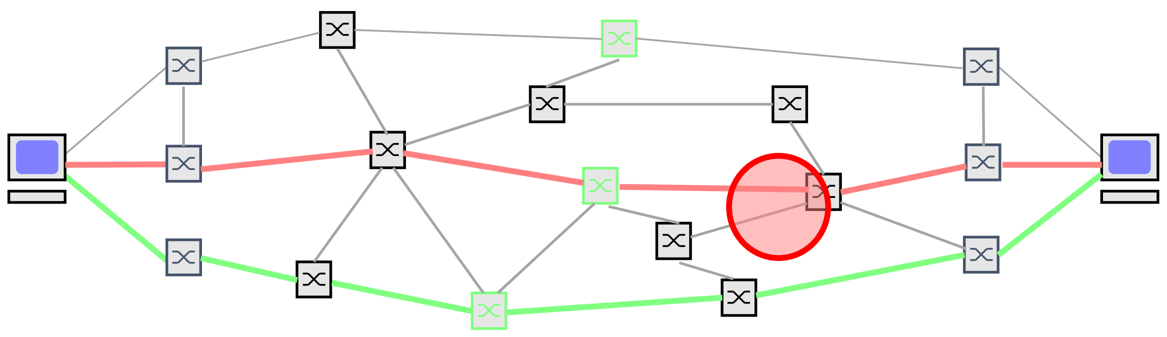

Figure 3.26: ResTP multi-path hot-standby communication with path failure

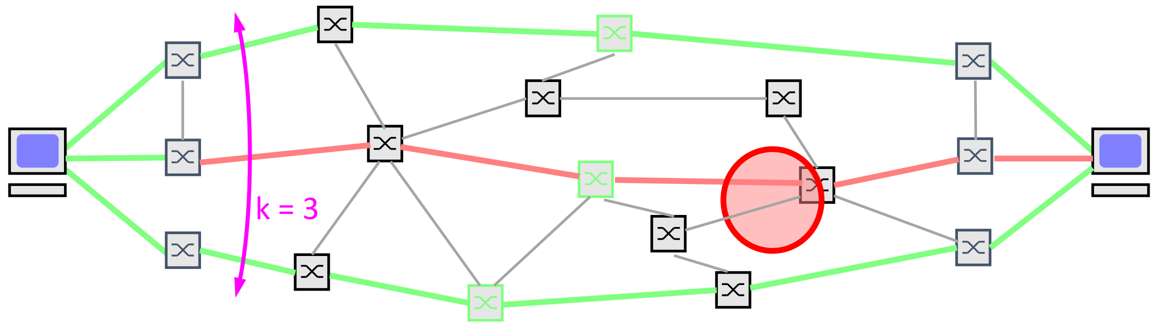

Figure 3.27: ResTP multi-path spreading communication with path failure

ResTP is able to learn about the failure and make the transition. With cross-layering, ResTP can be explicitly notified about the disruption of the main path by the lower layer, allowing a more promptly switching to an alternate path.

Using one path for data transfer and only replacing it with a second path in the event of its failure is also referred as the 'all-or-nothing' approach. Another instance of the hot-standby policy is to completely saturate one path before utilizing an additional path, which is referred as the 'overflow' approach [18].

When operating in its spreading mode, ResTP actively distributes data among all established paths. This is where all ResTP multi-path components (packet scheduler, path manager, ...) get involved. A spreading communication is able to survive the failure of individual paths with no loss to the application (Figure 3.27).

The selection between the hot-standby and spreading subflow policies is based on many factors including path characteristics and application requirements. When considering path characteristics alone, with high-bandwidth and low-delay links, the need to simultaneously spread data across multiple paths is small, so the hot-standby subflow policy may be more suited to reduce the overall complexity as many components (coupled congestion control, packet scheduler) are not required. With low-bandwidth and high-delay links, the spreading mode could be more beneficial as ResTP pools multiple paths to increase the throughput.

Bandwidth-intensive applications typically benefit from the spreading mode to maximize their throughput. However, in an environment with high-bandwidth and low-delay channels, developers may choose to run their applications over ResTP in its overflow hot-standby mode if system complexity needs to be taken into considerations.

A study [89] on the suitability of multi-path transports (CMT-SCTP and MPTCP) for latency-sensitive traffic including live video communication (Skype), gaming (Massive Multiplayer Online or MMO), and Web traffic shows that multi-path communication can reduce latency significantly when paths have similar delay and loss rate. However, in asymmetric scenarios, the latency reduction is not achieved. The study also emphasizes that the scheduling mechanism employed plays an important role on the performance of multi-path transports as a scheduler designed for throughput maximization may lead to latency increase in some scenarios with heterogeneous paths. When discussing the impact that MPTCP may have on applications in RFC 6897 [90], the authors also state that the jitter perceivable to an application may appear higher when spreading data across subflows with different delays, and the delay and jitter of data transport over MPTCP highly depend on the congestion control and scheduling algorithms that are in operation. So, when using ResTP, the users may want to exploit the flexibility in its design to experiment with different congestion and scheduling algorithms to find the best fit.

For loss-intolerant applications, utilizing multiple links simultaneously could minimize data losses upon link failures. In a more stable network environment, unless the applications are also bandwidth-intensive, spreading data over multiple paths may not be necessary. Choosing the most reliable link with low error rate and operating in the hot-standby mode with the ARQ or HARQ error control may be sufficient.

3.4 Cross-Layering Framework

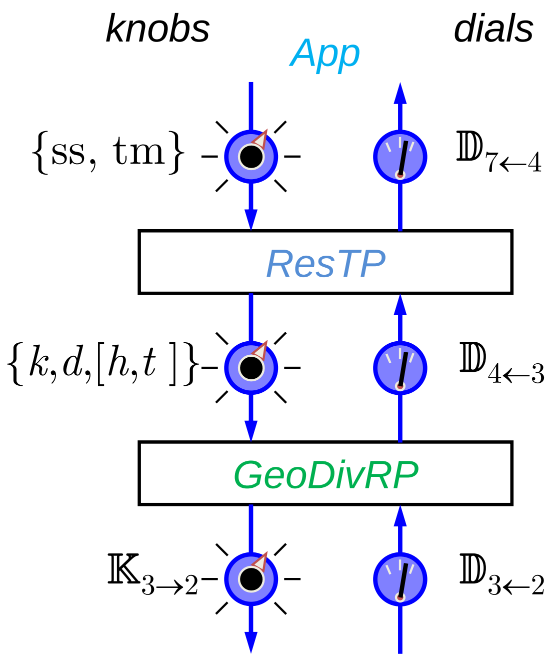
knobs
App
dials
{ss, tm}
$\mathbb{D}_{7\leftarrow 4}$
ResTP
$\{k,d,[h,t\,]\}$
$\mathbb{D}_{4\leftarrow 3}$
GeoDivRP
$\mathbb{K}_{3\rightarrow 2}$
$\mathbb{D}_{3\leftarrow 2}$

Figure 3.28: ResTP in the ResiliNets protocol stack

The ResTP-GeoDivRP protocol stack supports cross-layering as illustrated in Figure 3.28. In the model, the downward *knobs* $\mathbb{K}$ allow service tuning from the application layer while the upward *dials* $\mathbb{D}$ allow feedback updating from the network layer. Specifically, ResTP receives the service specification (ss) and threat model (tm) from the application. Based on the information, ResTP then determines and adapts its mechanisms and resilient services to satisfy the application requirements. When being deployed together with the GeoDivRP routing protocol [12–14] and operating in its multipath mode, ResTP requests the GeoDivRP to calculate k number of d-geodiverse paths meeting $[h,t]$ stretch (number of additional hops for diverse paths) and skew (delay difference across paths) criteria, where d is the distance between the paths. These paths, when being passed up to ResTP will be used for actual data transfer. This ResiliNets framework therefore achieves survivability.

3.5 ResTP Modules

In this section, we explain all the modules that made up our ResTP transport-layer protocol, starting with the heart of the protocol, the main module.

3.5.1 Main Module

The main module must be present in every ResTP configuration. It performs multiple tasks and is critical to the operation of the protocol:

- The module is the connection point between ResTP and the upper and lower layer.
- The module performs multiplexing and demultiplexing.
- The module holds variables shared among the other modules and coordinates their operations when they are present in a configuration.

3.5.2 Helper Module

This module performs functions and calculations that are necessary for other modules to operate such as the round trip time estimation.

3.5.3 Framing Module

The framing module converts application-layer frames into ResTP segments. There are two framing modes handled by the framing module. The fixed-size framing mode ensures that each application-layer frame is written as a separate ResTP segment on the sending end and read as a separate ResTP segment on the receiving end. With this mode, packet boundaries are reserved even on retransmissions (through ARQ or HARQ with fully reliable service enabled). The coalesce framing mode allows multiple application-layer frames to be combined in a single ResTP segment. The framing module participates in every ResTP configuration, and one of the two framing modes needs to be enabled.

3.5.4 Flow Control Module

Flow control in ResTP is handled by the flow control module, which limits a ResTP sender's transmission rate to avoid overwhelming the receiver. ResTP currently implements only the TCP-like windowed flow control although, which can be enabled through the WIN bit in **FC**.

3.5.5 Connection Management Module

The connection management module handles the establishment and termination of a ResTP connection (both single-path and multipath).

Single-Path Connection Management

ResTP supports three connection establishment schemes: UDP's connectionless, TCP's three-way handshake, and ResTP's opportunistic that can be specified using the corresponding bits in the **Flow** header field. ResTP employs the TCP's four-way FIN exchange termination technique in all of its reliable connections.

Connectionless Connection Establishment

Operated in the same way as UDP, in a connectionless connection, ResTP communicating hosts do not exchange any signaling messages. Application data are transmitted as soon as the connection is opened, and none of the bits in **Flow** are set in the header of these data packets to signify that the connectionless scheme is currently in operation.

Three-Way Handshake

Following the TCP's connection establishment scheme, ResTP communication hosts exchange SYN, SYN-ACK and ACK messages to initiate a three-way handshake connection before any data are allowed to be transmitted. ResTP requires all packets sent during this type of connection, including signaling, data, and ACK packets to have the CON bit in **Flow** set to "1".

We assume that Host A is the connection initiator. Host A begins by sending a SYN packet to host B. This signaling message is a non-payload packet that carries the basic ReaTP header illustrated in Figure 3.1 with the `SYN` bit in **Flags** and the `CON` bit in **Flow** set to "1". Host B, upon receiving Host A's SYN, replies with a SYN-ACK if it accepts the communication request. This SYN-ACK has the same format as the SYN with the `ACK` bit in **Flags** also set to "1". The connection is established when Host A sends an ACK back to Host B acknowledging the receipt of the SYN-ACK. This ACK's header has `ACK` in **Flags** and `CON` in **Flow** set to "1". All data packets following the final ACK signaling message must have the `CON` bit enabled.

Opportunistic

As mentioned in Section 2.1, TCP 3-way handshake adds an RTT to the overall network delay, which is significant for short flows such as short Web transfers, especially over a high-latency environment. To address this shortcoming, ResTP proposes an enhanced handshake approach called opportunistic (ResTP OPT). Introduced in our previous publication [91], ResTP's opportunistic allows data to be exchanged and delivered to the application during the establishment process. TCP Fast Open (TFO) also allows data transmission during handshake, but takes a slightly different approach than our opportunistic. First, TFO transmits the initial application data inside the SYN segment. So the amount of data transmitted is limited to the amount that fits within the SYN. ResTP's opportunistic transmits application data during the handshake in separate packets. Hence, the client can send as much data as permitted by the congestion and flow control that are in use (note that QUIC [92] also employs this approach but QUIC is not a pure transport-layer protocol as TCP and ResTP).

Second, TFO does not consider control (signaling) segment losses in its design. Measurements show that control packets do get lost at a relatively high rate of 0.5% [93]. Specifically, a measurement at the University of Innsbruck employee's web mail server shows that out of a total of 737,188 recorded connection requests, 730,523 (99.1%) are successful, and 5162 (0.7%) of the successful connections require more than one SYN packet. A second measurement performed at the university's Internet proxy shows that out of a total of 1,721,382 logged connection attempts, 1,708,442 (99.2%) are successful, and 2,148 of those successful connections require multiple SYN-ACK and SYN control packets. TCP sets its retransmission timeout (RTO) to be 3 seconds during the handshake (the first RTT). Hence, a SYN or SYN-ACK loss costs the connection at least an additional 3-second delay. ResTP's opportunistic mitigates this issue by turning on the `SYN` bit for the first data packet sent after initial SYN control message and the `SYN` and `ACK` bits for the first data packet sent after initial SYN-ACK control messages. These flags serve as the second SYN and SYN-ACK to survive any initial losses.

As highlighted in RFC 7413 [42], allowing data to be transmitted and passed up to the application layer before the 3-way handshake is completed opens up many serious vulnerabilities. To address these security issues, ResTP OPT adopts the client authentication process proposed in TFO. A client that wants to use the opportunistic mechanism should first transmit a cookie request to the server. The cookie request is a SYN packet that contains the basic header shown in Figure 3.1 with no payload, and with the `SYN`, `OPT`, and `CKI` flags set to 1. The server then generates a cookie and sends it back in a cookie response, which has the same format as the cookie request, except with the addition of the **Cookie** extension (Figure 3.4). The response is a SYN-ACK packet that has both the `SYN` and `ACK` flags set to 1 in addition to the `OPT`, and `CKI` bits. The client then caches the cookie for future ResTP opportunistic connections.

To illustrate our opportunistic connection establishment scheme, we assume that a client wants to request a file from a server, and the client has been fully authenticated by the process described above. It means that the client has obtained a cookie and is ready to initiate its web page request through the following steps:

1. The client sends a SYN packet to the server. This SYN has no payload and carries the ResTP basic header (Figure 3.1) extended by the **Cookie** extension containing the cookie the client obtained previously. The header has the `OPT` bit in **Flow**, and `SYN` in **Flags** set to "1".

2. The client sends its request for the file. This request packet serves two purposes: file request and duplicated SYN. Hence, the request needs to carry the cookie and has both `SYN` and `OPT` enabled. In case the initial SYN is lost, the connection setup can still proceed when the server receives this request from the client.

3. When the server receives the client's SYN, the server checks the client's cookie to ensure that the request is from an authenticated client. If the server decides to accept the communication request, the server responds with a SYN-ACK message as normal. This message's header has the bits `OPT`, `SYN`, and `ACK` enabled.

4. When the server receives the client's request, there are two possibilities. If the server has not received the initial SYN packet, the server treats this request as a SYN and performs the cookie verification before acknowledging with a SYN-ACK. If the server already received the initial SYN, the server ignores the cookie and the `SYN` flag and only processes the actual request.

5. The server sends the requested file back to the client. This response serves two purposes: file response and duplicated SYN-ACK. The response's header has the **Flags**:`SYN`, **Flags**:`ACK`, and **Flow**:`OPT` bits enabled.

6. If the client receives the normal SYN-ACK packet, it acknowledges the receipt with an ACK packet as normal. If the client receives the response (but have not received any SYN-ACK), the client treats this response as a SYN-ACK to move the connection to the established state before processing the data file carried in the response.

We note that while the use of the Cookie option in TCP for TFO is limited by the maximum 40-byte option length allowed in a TCP segment and the presence of other options, ResTP does not have that concern because our protocol does not impose any limits on the number of extensions carried in a ResTP packet.

Multipath Connection Management

ResTP supports the standard three-way handshake and opportunistic connection establishment techniques when operating in its multipath mode.

Three-Way Handshake

With three-way handshake, ResTP follows the MPTCP's connection initiation described in Section 3.1 of RFC 6824 [18] with some minor modifications. All the initial signaling messages must have the `MP?` and `MPM` bits properly set (in addition to those bits described in the single-path three-way handshake section above). The value of **k**, which specifies the total number of paths the two hosts use for this communication session must be initialized to 1 for the master subflow and incremented every time a new subflow is added. If the multipath bits (`MP?` and `MPM`) are not correctly enabled during the handshaking, ResTP will fall back to its single-path mode (with three-way handshake enabled).

Opportunistic

ResTP OPT in multipath mode operates as in unipath mode. As with the three-way handshake technique, all initial signaling messages must have `MP?` and `MPM` in the **P** header field properly set in order to differentiate between single-path and multipath operations.

3.5.6 Congestion Module

Congestion module handles congestion control for both single and multi-path ResTP connections. To accommodate the different congestion control algorithms proposed, ResTP uses a bit vector to identify a specific algorithm. If `MP?` is set to 0, the **Cong** field specifies a single-path algorithm, and if `MP?` is set to 1, this field specifies a multipath algorithm.

The current ResTP implementation only supports three congestion control algorithms: no congestion control, TCP NewReno, and TCP BIC.

3.5.7 Ordering Module

The ordering module controls the order in which data are delivered to the receiving application with respect to the order in which they were transmitted. ResTP provides three types of ordering service to the application: unordered, partially ordered, and ordered delivery.

Unordered Data Delivery

With the unordered service, a ResTP receiver delivers data to the application as soon as they arrive without worrying about their orders. When this service is in operation, none of the bits in the **Ord** header field are enabled.

Ordered Data Delivery

With the in-order data delivery service, the receiver keeps out-of-order packets (those that have sequence numbers greater than the next expected sequence number) buffered until all of their predecessors have been received and delivered. This is the exact mechanism employed by TCP and MPTCP. To negotiate the in-order data delivery, the communicating hosts use the `FUL` bit in the **Ord** header field. All packets transmitted during the lifetime of an in-order data delivery connection need to enable this `FUL` bit.

Partially-Ordered Data Delivery

With partially-ordered data delivery, the receiver allows out-of-order packets to be delivered to the receiving application when the application is waiting for more packets, but the next expected in-order packets have not yet arrived. With this service, some data can make it on time for their delivery while some have to miss the deadline. The amount of out-of-ordered data passed to the application depends on many factors, including the underlying network delay and error rate. This service is especially useful for real-time multimedia applications. The partially-ordered data delivery is enabled using the `PAR` bit in the **Ord** header field.

3.5.8 Reliability Module

The reliability module handles the error control functionality of ResTP. Basically, it performs two main tasks: error detection and error recovery.

Error detection is achieved using CRC (cyclic redundancy check) once enabled through the `CRC` bit in the **Error** header field. When `CRC` bit is set to “1”, ResTP performs error check on both the header and the payload (if present).

We use the more powerful cyclic redundancy check (CRC) instead of checksum for bit error detection in ResTP because unlike TCP that was designed for wired networks, ResTP is also tailored for wireless environments, in which bit errors are more common, and the error rates are much higher.

Error recovery is accomplished using one of the three mechanisms supported by ResTP: FEC (forward error correction), ARQ (automatic repeat request with retransmission), and HARQ (hybrid ARQ).

Unreliable

ResTP can be configured to provide no reliability service at all, in which neither an error check is performed nor a lost packet is recovered. In this case, the reliability module is unplugged, and ResTP is said to operate in its unreliable mode.

Quasi Reliable

In its quasi reliable mode, ResTP provides some level of statistical reliability by relying on open-loop error recovery mechanisms such as FEC.

The current ResTP implementation supports two XOR-based FEC encoding types proposed by TCP-IR [87, 88]: the basic XOR technique that encodes every data packet up to 16 packets in a single encoded one, and the interleaved XOR technique that encodes every other ResTP packet up to 8 packets in a single encoded one. Basic XOR can recover a single packet loss in the encoding range while interleaved XOR can recover two consecutive losses.

The use of FEC is negotiated using the `FEC` bit in **Error** and the FEC extension (Figure 3.5) during the initial handshake. In these signaling messages, the **FEC Range** field in the FEC extension has a value of 0.

After the communicating hosts agree to use one of the supported encoding type for their communication, all packets exchanged afterward must have the `FEC` bit set, and those that carry an encoded payload also need to have the `ENC` bit set with an appended FEC extension. The **FEC Range** field in an encoded packet must be non-zero to specify the number of packets encoded in the payload. Furthermore, the quasi reliable mode requires all data packets (regular and encoded) to have a sequence number carried in the Sequence Number extension (Figure 3.6).

The behavior of the ResTP sender when operating in its quasi reliable mode follows the TCP-IR sender's. Before a regular data packet is transmitted, an FEC timer is set. The duration of this timer is currently set to RTT/4, but this value can be adjusted if desired. When the timer fires, an encoded packet is created and transmitted. The payload of this packet is the XOR of the payloads of every (basic XOR) or every other (interleaved

XOR) packets that were not encoded before, and the sequence number of this packet is the sequence number of the first packet it encodes.

On the receiver's side, when FEC is enabled, the receiver keeps a copy of the last 15 packets in its buffer even if they have been consumed by the application layer. When the receiver receives an FEC encoded packet, the receiver checks the **FEC Range** field to determine the range of packets encoded. If all packets in the encoding range have been received successfully, the receiver discards the encoded packet. If there is a lost packet in the encoding range, the receiver will try to reconstruct the packet.

Fully Reliable

The fully reliable mode guarantees correct data delivery by preserving the E2E (end-to-end) ACK semantics with ARQ or HARQ error correction mechanism. When ResTP operates in its fully reliable mode, the ACK module (described in the following section) must be plugged in. Furthermore, fully reliable can be configured in combination with either unordered or ordered data delivery service discussed in the previous section

Fully Reliable with ARQ

ARQ usage is negotiated using the `ARQ` bit in **Error**. All packets transmitted during the connection must carried an ARQ extension with a **Sequence Number** field and an **Acknowledgment** as illustrated in Figure 3.7.

The behavior of the ResTP sender and receiver in the fully reliable mode with ARQ follows the behavior of TCP entities, except that the sender's retransmission behavior depends on the ACK technique employed:

1. Positive ACK and fast retransmit: If positive ACK is enabled, the sender employs fast retransmit, and a retransmission event is triggered when the sender receives

three duplicate ACKs or when the retransmission timer (RTO) expires depending on which event happens first.

2. Positive ACK and periodic probe: This retransmission behavior is especially useful when handling link outages. When the RTO expires, ResTP ceases its new data transmission, suspends its RTO and transmits periodic probe packets (see Section 3.6 for more details on ResTP loss handling).

3. Delayed ACK: If delayed ACK is enabled, fast retransmit is disabled (the reason is explained in the ACK module section). One way to speed up the loss recovery in this case is through the help of a SNACK, which gives the sender more information about the state of the receiving buffer than a single ACK header field used in delayed ACK.

4. Selective ACK: If selective ACK is enabled, ResTP employs selective repeat and retransmits only lost packets identified in the SACK extension. A retransmission is also triggered when the RTO expires before the sender receives any SACK, but this behavior is not desirable.

5. Selective negative ACK: If SNACK is enabled, ResTP also employs selective repeat and retransmits only lost packets identified in the SNACK extension. RTO is set, but a retransmission through RTO is not desirable.

Fully Reliable with HARQ

The use of HARQ is negotiated using the `FEC` and `ARQ` bits in **Error**. All signaling messages (SYN, SYN-ACK, and ACK) exchanged during the initial handshake must carry the HARQ extension (Figure 3.17) consisting of both the FEC and ARQ extensions (Figure 3.5 and 3.7, respectively). All packets exchanged after the handshake must carry the ARQ extension if they are regular data packets, and both FEC and ARQ extensions if they are encoded or ACK packets.

On one end, the sender encodes packets the exact same way as when it is in quasi reliable mode with FEC. On the other end, the receiver also processes encoded packets the same way it does in quasi reliable mode. However, in addition to an attempt to recover lost packets based on the encoded data, the receiver also sends an ACK back to the sender. If a loss can be recovered successfully, the receiver sends a regular positive ACK back to the sender. This ACK carries both the FEC and ARQ extensions, but the **FEC Range** field has a value of 0. If none of the packets in the encoding range can be recovered, the receiver sends a positive ACK, but with the **FEC Range** specifying the loss range. The loss range is the offset between the sequence number of the next expected packet (rcv_nxt) and the sequence number of the last packet lost.

When receiving an ACK with a zero **FEC Range** field, the sender can adjust its congestion window according to the algorithm currently in operation if the congestion module is plugged in and congestion control is enabled. When receiving an ACK with non-zero **FEC Range** field, the sender extracts the loss range and tries to recover lost packets through Fast Retransmit.

Partially Reliable

ResTP operates in partially reliable mode when both ARQ and partially ordered data delivery are enabled (both `ARQ` and `PAR` bits are set to 1). The sender operates as a normal ARQ without having to retransmit some lost packets if the receiver has skipped them.

3.5.9 ACK Module

The ACK module handles the different acknowledgment techniques provided by ResTP, including positive ACK, delayed ACK, negative ACK (NACK), selective ACK (SACK), and selective negative ACK (SNACK).

Positive ACK

The positive ACK technique is enabled through the `ACK` bit in **Flags** with the sequence number of acknowledgment number extension (Figure 3.7) appended to every ACK packet. There are a couple of differences between the ResTP's positive ACK and the traditional cumulative ACK employed by TCP. First, the receipt of a positive ACK does not necessarily indicate that the receiver (the ACK generator) has received all data packets whose sequence numbers less than the acknowledgment number specified in the ACK. It only means that the receiver has advanced its receive pointer (rcv_next_ptr) and no longer needs the data prior to the acknowledgment number specified in this ACK. ResTP relaxes the traditional meaning of cumulative ACK because in addition to supporting fully reliable and ordered data delivery services, ResTP also supports partial reliable and partially-ordered services. With these services, the receiver is allowed to generate an ACK even some data are missing, and the sender is allowed to discard those data without worrying about retransmitting them.

Second, ResTP's positive ACK does not have to come with the fully reliable and ordered data delivery services as in TCP. For example, ResTP can be configured as an unreliable, unordered protocol that provides only flow control and congestion control. In this case, a regular flow of ACKs would be needed for the proper function of the selected congestion and flow control algorithms, so the ACK module with the positive ACK scheme enabled must be plugged while the reliability module and ordering module can be unplugged.

With the positive ACK technique, the receiver transmits an ACK for every other packets received. When an out-of-order queue forms, the receiver acknowledges every packet received. When positive ACK is used together with the partially-ordered technique, the receiver also transmits an ACK whenever the application reads pass the hole and advances its receive pointer.

Delayed ACK

The delayed ACK technique is similar to the positive ACK except that the ACK transmission frequency is delayed for a configurable period of time (delay period). Delayed ACK is especially useful in network environments with asymmetric channels such as the space communication in which the capacity of the forward link (from the spacecraft to the ground) is substantially larger than the capacity of the reverse link (from the ground to the spacecraft). If the receiver generates an ACK for every other packet received as in the positive ACK technique, the reverse link may be overrun.

There are some limitations of delayed ACK that we want to emphasize. Delayed ACK impairs the function of TCP congestion control algorithms that are currently employed by ResTP because these algorithms require a steady flow of ACKs to correctly determine the data sending rate. Hence, delayed ACK should not be enabled when ResTP is configured with those algoritms. In the future, ResTP can be extended with an open-loop rate based congestion control algorithm (similar to the one suggested by SCPS-TP [80]) that works better with delayed ACK. Furthermore, delayed ACK should not be combined with the partially-ordered data delivery mechanism. When the application skips a hole in the receiver buffer, the receiver should send an ACK immediately so that the sender can continue to send out new data. Lastly, delayed ACK does not work well with Fast Retransmit because the sender may never receive enough duplicate ACKs (normally 3 duplicate ACKs) to trigger the fast retransmit due to the reduced ACK frequency.

Future work on ResTP requires more simulations and analysis to determine an optimal value (or range of value) for the delay period considering different network conditions including their latency characteristic.

Delayed ACK is enabled through the `DEL` bit in **Error**, and every ACK packet's header carries the extension illustrated in Figure 3.7.

Selective ACK

Selective ACK is used in combination with positive ACK or delayed ACK to inform multiple holes in the receiving buffer and is negotiated using the `SAK` bit in **Error**. When enabled, a SACK extension (Figure 3.8) is appended to an ACK packet whenever a hole exists. ResTP follows the SACK generation and processing policy outlined in RFC 2018 [39]. Because ResTP does not limit the size of its extension, ResTP implements a configurable parameter to limit the number of SACK blocks that can be appended to an ACK due to the large number of bytes requires to signify a contiguous data block in SACK.

Selective negative ACK

Similar to SACK, selective negative ACK (SNACK) can be combined with either positive ACK or delayed ACK to inform multiple holes in the receiving buffer. For the signaling SYN, SYN-ACK, and ACK messages, the use of SNACK is negotiated using both the `SAK` and `NAK` bits or only the `NAK` bit in **Error**. If a SNACK does not carry the **Bit Vector** field, it is basically a NACK, which signifies a single hole in the receiving buffer. In this case, only the `NAK` bit is enabled. On the other hand, if a SNACK signifies multiple holes in **Bit Vector**, both `SAK` and `NAK` bits are enabled. Based on these flags, a SNACK recipient can determine the presence of the optional **Bit Vector** field.

Both SACK and SNACK have the ability to convey multiple missing holes in the receiving buffer, which helps the data sender recover from multiple losses faster than when only positive ACK is employed. The benefit of being able to recover from multiple losses per sending window is even intensified when dealing with long-delay channels. However, because the format of SNACK is more bit-efficient than SACK, SNACK is more suitable for lossy and bandwidth-constrained network environments.

SCPS-TP treats SNACK as a request for retransmission. It means that when the data sender receives a SNACK, it needs to retransmit all the missing data signified by the SNACK immediately. In ResTP, the decision to retransmit all missing data does not depend on whether the sender receives a SNACK or a SACK, but on the reliability and ordering techniques that are in operation. If ResTP is configured with partially reliable and partially ordered data delivery services, the sender will retransmit all missing segments upon a SNACK receipt. This is to increase the chance that these missing data can arrive on time to fill up the holes in the receiving buffer before the receiving application reads the data.

Duplicate ACK

ResTP provides a parameter to enable duplicate ACK transmissions. When enabled, the data receiver retransmits its last sent ACK packet if it has not received any new data from the sender for a configurable period of time. ACK duplication is useful when the ACK channel is noisy, and an ACK loss may block the sender from transmitting new data because it has drained its congestion window. Without receiving an ACK on time, the sender has to timeout and resets its window.

3.5.10 Scheduling Module

The scheduling module handles the distribution of data packets among available paths when ResTP operates in its multipath spreading mode (MPM and MP?) set to 1. ResTP currently supports two packet scheduling algorithms: round robin enabled through the `RRS` bit and fastest RTT enabled through the `FRS` bit in **Sched**.

3.6 ResTP Data Loss Handling

Because ResTP is designed as a general-purpose transport-layer protocol that can be employed in different network environments, including wired and wireless, the protocol needs to be able to cope with different sources of data loss, including one caused by congestion, corruption, and link outage.

The first step in handling data loss is to identify the source of loss. The current ResTP implementation only supports a static loss identification: the protocol uses a parameter that can be set by a network manager to determine the type of packet losses. In the future, dynamic loss identification will be studied and incorporated into ResTP so that the protocol can operate more efficiently, especially over mixed wired and wireless network. One way of implementing dynamic loss identification for ResTP is to exploit the cross-layering framework employed by the ResiliNets protocol stack (ResTP working together with GeoDivRP), which allowing the lower layer to communicate directly with ResTP about the cause of a drop.

When the loss parameter is set to congestion, ResTP plugs in its congestion control module and invoke its NewReno algorithm. The protocol also selects ARQ with positive ACK as its error control mechanism with ordered data delivery service, three-way handshake as its connection management method, and the windowed flow control technique.

This configuration of ResTP matches the algorithms implemented by the standard TCP, which has been known for handling network congestion efficiently. We note that this default configuration can be changed if desired. For example, instead of NewReno, a different congestion control algorithm can be used.

When the loss parameter is set to corruption, ResTP automatically disables its congestion control module while enabling its reliability module by default. ARQ with positive ACK is selected to retransmit lost packets. The three-way handshake is employed as the connection establishment technique, and the windowed flow control is enabled. Instead of disabling congestion control, the network manager can plug in the module and use an algorithm that handles corruption-based losses better than the standard NewReno.

When the loss parameter is set to link outage, ResTP is configured with three-way handshake connection establishment, ARQ with positive ACK and periodic probe error control, NewReno congestion control, and windowed flow control. In this default configuration, when a loss happens, ResTP will cease its transmission, suspend its retransmission timer and only transmits periodic probe packets. This mechanism allows ResTP to sustain its throughput and prevents the protocol from exceeding its retransmission threshold and aborting the connection in the worst case. ResTP uses the last transmitted data packet as probe packet. We do not use dummy segments as probe packets because we want every chance to transmit useful data, especially when a connection is failure-prone. In addition, because the current version of ResTP does not implement a link failure detection mechanism (it uses a parameter to pre-set different sources of loss), ResTP only starts sending probe packet when an RTO happens. For simplicity, ResTP also uses this RTO to time its probe transmission.

The idea having different modes for handling different causes of losses is inspired by SCPS-TP although SCPS-TP does not provide the flexibility to configure the desired algorithms like ResTP.

3.7 ResTP for Real-Time Multimedia Applications

ResTP provides multiple configurations for real-time multimedia applications. If timeliness is the only concern, the unreliable ResTP configuration should be able to quickly transport the application-layer data across the network. This configuration is as simple and light-weight as UDP and performs as well as the standard Internet protocol. In case timeliness is the requirement, but congestion and corruption should also been taken care of, ResTP provides an all-in-one configuration consisting of congestion control, error, and flow control that is specifically tailored for real-time multimedia applications to make sure the data are delivered on time.

Based on our simulations and analysis, the most suitable ResTP configuration that can achieve reliability without sacrificing timeliness is ResTP-3WH-PAR-ARQ. In this section, we explain the techniques that this configuration consists of. In Section 4.5, we simulates and analyzes its performance in comparison with TCP.

ResTP-3WH-PAR-ARQ consists of the three-way handshake technique, partially-ordered data delivery, and ARQ with SACK or SNACK error control. With this configuration, the fixed framing mode inside the framing module (Section 3.5.3) must be enabled.

The implementation of the partially-ordered data delivery technique in ResTP includes the enabling of either SACK or SNACK extension. SNACK is a better choice if the network is bandwidth-constrained or bandwidth-asymmetric. When a hole exists in its reordering buffer, the ResTP receiver immediately transmits an ACK with a SACK or SNACK letting the sender know the exact data that are missing. While waiting for missing segments, if the receiving application requests data, the receiver passes all data currently in its buffer (including out-of-order packets) up the stack. This is the skip-over hole technique implemented in TCP-RTM. Whenever the receiver has to skip over a hole, it immediately informs the sender through an ACK so that the sender can discard the segments instead of trying to retransmit them. Before this ACK, the sender will try to retransmit missing data upon a SACK or SNACK reception.

ResTP for real-time multimedia applications is different from TCP-RTM in multiple ways. First, ResTP does not require an application-level framing technique. ResTP's framing module with the fixed-size mode enabled will ensure that packet boundaries are maintained between application-level frames and transport-level segments. Second, with ResTP, the application does not have to worry about sending periodic heartbeat packets to keep the data stream flowing in case ACKs are lost. If the reverse channel is lossy, ResTP can be configured with duplicate ACK enabled to wake the sender up occasionally. ResTP can efficiently handle real-time multimedia traffic without any changes from the applications. Third, in addition to minimizing the delay caused by the ARQ technique, ResTP can be configured with its opportunistic handshaking to further reduce overall latency.

3.8 ResTP Solutions for UDP, TCP, and MPTCP Shortcomings

In Table 3.3, we summarize the shortcomings of the conventional transport protocol (UDP, TCP, and MPTCP) and how ResTP addresses those issues using the ResiliNets framework that it employs and the algorithms (discussed above) that it implements. In Chapter 4, we perform various simulations to demonstrate the ResTP benefits highlighted in this table.

Limitations of UDP, TCP, and MPTCP	**ResTP solutions**
TCP (and MPTCP) 3-way handshake adds latency, especially to short flows	Opportunistic allows data to be processed during setup, hence saving an RTT
TCP Fast Open (TFO) is unable to survive SYN, SYN-AC, or both loss	Opportunistic can survive these losses without retransmissions or connection re-start
UDP & TCP do not perform well in different network environments	Composibility allow ResTP to be environment-specifically customized
UDP & TCP are unable to serve different application classes due to their fixed sets of services	Composibility allows ResTP to be application-specifically customized
UDP & TCP do not have mechanism to survive link outage	Link outage handling technique helps ResTP save data from being lost when facing link breakage
MPTCP cannot survive network challenge	ResTP-GeoDivRP can survive network challenge

Table 3.3: ResTP solutions to UDP, TCP, and MPTCP shortcomings

Chapter 4

ResTP Performance Evaluation and Analysis

The implementation of our ResTP in ns-3 along with the verification and validation of our model are described in Apendix A. In this Chapter, we present our performance evaluation and analysis of ResTP through multiple sets of simulations to show:

1. The advantages of our opportunistic approach in improving the completion time of short flows when comparing with TCP (and MPTCP)'s three-way handshake, and in surviving SYN or SYN-ACK or both SYN and SYN-ACK loss without having to wait for their retransmissions or re-start the connection when comparing with TFO (TCP Fast Open).

2. The benefits of ResTP's composability and custom configurations in serving different application classes and adapting to different network environments.

3. The ability of ResTP in surviving link outages in networks with intermittent connectivity

4. The benefits of cross-layering and the ResiliNets protocol stack in surviving network failures when comparing ResTP-GeoDivRP with MPTCP.

4.1 Single-Path ResTP with File Transfer Application

In this set of simulations, we compare the performance of ResTP with TCP when transporting data files across network. File transfer applications require a fully reliable and in-order data transfer service, which is the exact service that TCP provides. We expect ResTP to perform as well as TCP in this scenario.

4.1.1 Simulation Setup and Topology

Figure 4.1 illustrates the topology that we use for this set of simulations. A single traffic generator (source) communicates with a receiver (sink) through a router. The link between the source and the router has a bandwidth of 10 Mb/s and a delay of 0.01 ms. The link between router and the sink has a bandwidth varied from 2 Mb/s to 10 Mb/s, a delay 0.01 ms, and a bit error rate varied from 10^{-7} to 10^{-4}. The router implements a drop-tail queue that has a size of the bandwidth-$\times$-delay product (BDP). TCP employs the NewReno congestion control algorithm. The file size varies from 250 kB, 500 kB, and 1000kB.

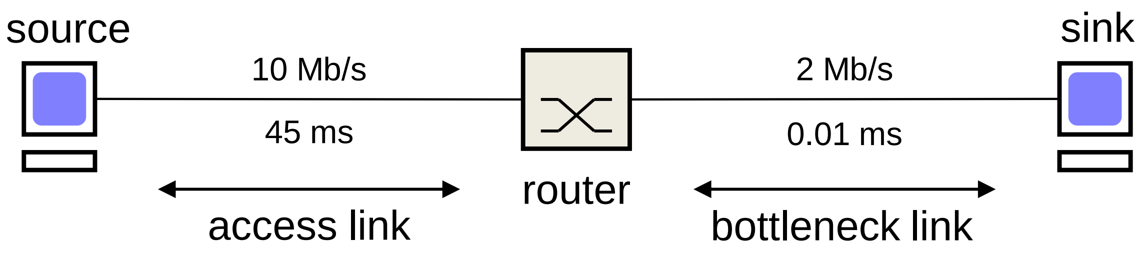

Figure 4.1: Single router simulation topology

4.1.2 ResTP Configuration

ResTP has two configurations that can be used for reliable file transfer: ResTP-3WH-ARQ (Table 4.1) and ResTP-3WH-HARQ (Table 4.2).

CM	**FC**	**CC**	**Ordering**	**Reliability**	**ACK**
3WH	windowed	NewReno	ordered	fully reliable with ARQ	positive ACK

Table 4.1: ResTP-3WH-ARQ configuration for evaluating ResTP with file transfer application in comparison with TCP

CM	**FC**	**CC**	**Ordering**	**Reliability**	**ACK**
3WH	windowed	NewReno	ordered	fully reliable with HARQ	positive ACK

Table 4.2: ResTP-3WH-HARQ configuration for evaluating ResTP with file transfer application in comparison with TCP

4.1.3 Simulation Results and Analysis

We compare the performance of ResTP with TCP when they transfer a small data file of size 250 kB, 500 kB, and 1000 kB across an error-prone link.

Figure 4.2 illustrates the average throughput of ResTP and TCP when the file size is 1000 kB for different PER values. ResTP achieves slightly higher throughput than TCP for PER values smaller than 0.2.

File size	**PER**	**TCP**	**ResTP**	**% Latency Reduction**
250 kB	0.01	2.55 s	1.39 s	45.5%
500 kB	0.01	4.93 s	2.30 s	53.3%
1 MB	0.01	10.2 s	6.30 s	38.2%
250 kB	0.1	40.3 s	32.5 s	19.4%
500 kB	0.1	78.8 s	73.0 s	7.4%
1 MB	0.1	118 s	107 s	9.3%

Table 4.3: Flow Completion Time of ResTP vs. TCP

When we consider the flow completion time (FCT), a more important performance metric than throughput from a user perspective, especially for short flows, ResTP with HARQ

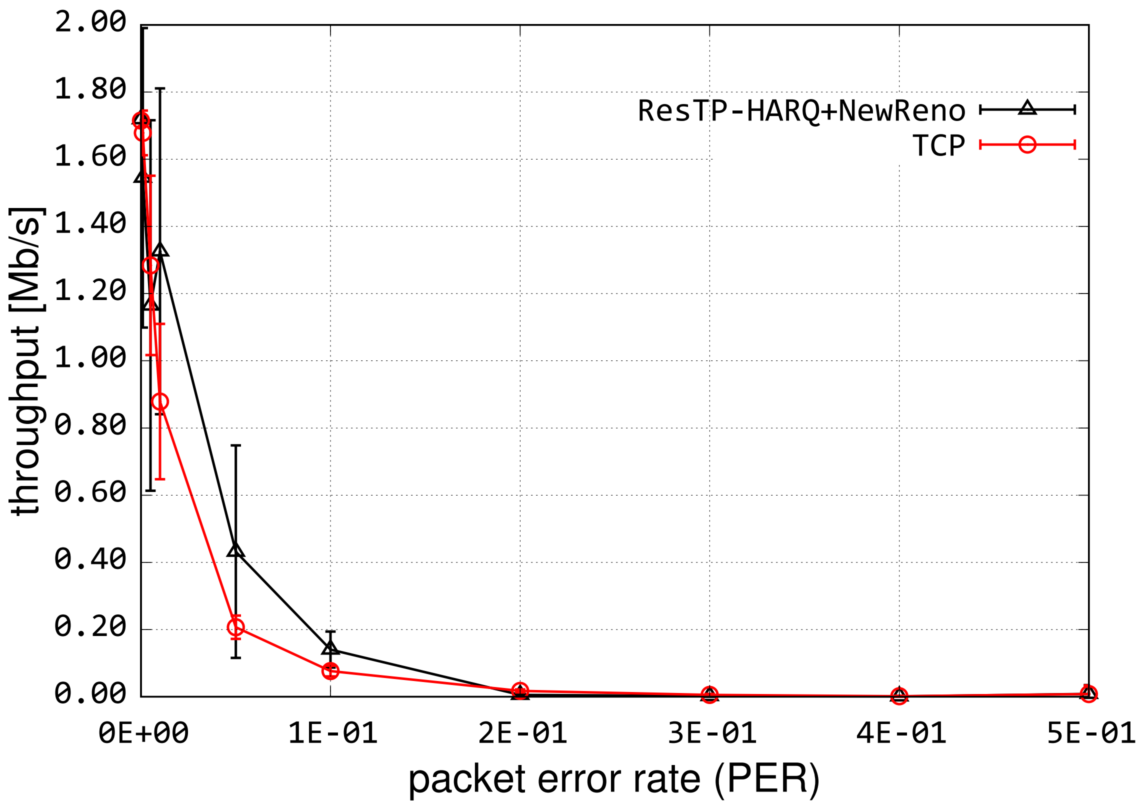

Figure 4.2: Throughput of ResTP vs. TCP using short file transfer over lossy link

is able to reduce the time it takes to complete the file transfer. Table 4.3 presents the average FCT after 20 replications for each PER value of 0.01 and 0.1. ResTP is able to reduce up to 53.3% of the TCP latency. With the 1000-kB file size, the percentage of latency reduction achieved by ResTP is 38.2% at 0.01 PER and 9.3% at 0.1 PER. The higher the PER, the smaller the latency improvement is because more packets are dropped, and some of the dropped packets are the FEC encoded packets that help the receiver recovering losses without waiting for retransmissions.

4.2 Single-Path ResTP with Web Services

In this section, we evaluate the performance of single-path ResTP, specifically its opportunistic connection management scheme when handling web browsing traffic using the Hypertext Transfer Protocol (HTTP) in comparison with TCP's three-way handshake and TFO (TCP Fast Open).

4.2.1 Simulation Setup and Topology

We start with the HTTP application that is part of the standard ns-3 release [94]. This implementation models a persistent connection, which is kept open until the client disconnects. At the beginning, the client opens a connection to the server. After the connection is established, the client sends a request packet asking for a main object. By default, each client's request has a constant size of 350 bytes. The size of a main object is determined by a truncated log-normal random distribution with a mean of 10710 bytes, a standard deviation of 25032, a lower bound of 100 bytes and an upper bound of 2 MB. After receiving the main object, the client parses the object to determine if there are any embedded objects. If there are, the client sends requests for them sequentially. A request for a subsequent embedded object is sent only after the previous object has been completely received. The number of embedded objects in a page is determined by a truncated Pareto distribution with a scale parameter of 2, a shape parameter of 1.1, and an upper bound of 55. These default values result in the number of embedded objects falling within [0, 53)] interval with an actual mean of approximately 3.95. The size of an embedded object is determined by a truncated log-normal distribution with a mean of 7758 bytes, standard deviation of 126168 bytes, a lower bound of 50 bytes, and an upper bound of 2 MB. After all embedded objects are received, the client enters its reading time for the downloaded web page before requesting another page. The parsing and reading time are determined by an exponential distribution with a mean of 130 ms, and 30 s, respectively.

On the server side, main objects and embedded objects are generated without any delay. The maximum transmission unit (MTU) size is randomly generated with a 24% chance for a low MTU size of 536 bytes, and a 76% chance for a high MTU of 1400 bytes. This means that if generated main object and embedded object size is greater than the MTU size, the server transmits its response in multiple packets.

The client and the server communicate through a 5 Mb/s point-to-point link. The channel delay ranges from 20 ms to 500 ms (RTT from 40 to 1000 ms). We modify the HTTP application to use ResTP protocol instead of TCP. For a fair comparison, we implement TFO inside ResTP. As previously mentioned, ResTP design and implementation allow the protocol to be extended with minimal effort when we want to demonstrate that capability.

The metric that we study in these simulations is the page load time (PLT). PLT is the amount of time for a client to see the whole web page (including main and embedded objects) loaded on the client's web browser after he/she initiates a search (or enters a URL). The ns-3's `ThreeGppHttpClient` requests multiple pages during a simulation duration. We only keep track of the PLT for the first page that the client requests, which consists of a 79453-byte main object and a 226-byte embedded object. We vary the simulation duration depending on the channel delay to ensure that at least the first page is received by the client successfully.

We compare the performance of TCP's 3-way handshake, TFO, and ResTP's opportunistic under four scenarios: no loss, one SYN loss, one SYN-ACK loss, and one SYN and SYN-ACK loss.

4.2.2 ResTP Configuration

In these simulations, ResTP is configured with the same mechanisms employed by the traditional TCP, except is connection establishment technique. Table 4.4 shows the modules that are plugged in for this configuration and the corresponding algorithms employed. We note that the main module is present in every ResTP configuration so it is not included in the table. Background on single-path connection establishment at the transport layer is provided in Section 2.1, and design details on ResTP's opportunistic approach is discussed in Section 3.5.5.

CM	**FC**	**CC**	**Ordering**	**Reliability**	**ACK**
opportunistic	windowed	NewReno	ordered	fully reliable with ARQ	positive ACK

Table 4.4: ResTP configuration for evaluating ResTP opportunistic handshaking with web services in comparison with TCP's 3WH and TFO

4.2.3 Simulation Results and Analysis

Figure 4.3 shows the page load time with increasing RTT when no losses occur, and Table 4.5 presents the improvement percentage of ResTP OPT over the conventional 3WH (because ResTP OPT performs as well as TFO in this scenario, a table for ResTP

OPT and TFO is omitted). The time it takes to download a page is longer when the channel latency increases. However, because both ResTP OPT and TFO allow data to be transmitted and delivered to the application during the handshake, they achieve lower page load time than 3WH. We also note that our ResTP's connection establishment technique OPT performs close to TFO as expected when no signaling messages are lost. The approach of transmitting data during the initial RTT in separate packets that ResTP OPT employs does not have any negative impacts on its performance when comparing with TFO.

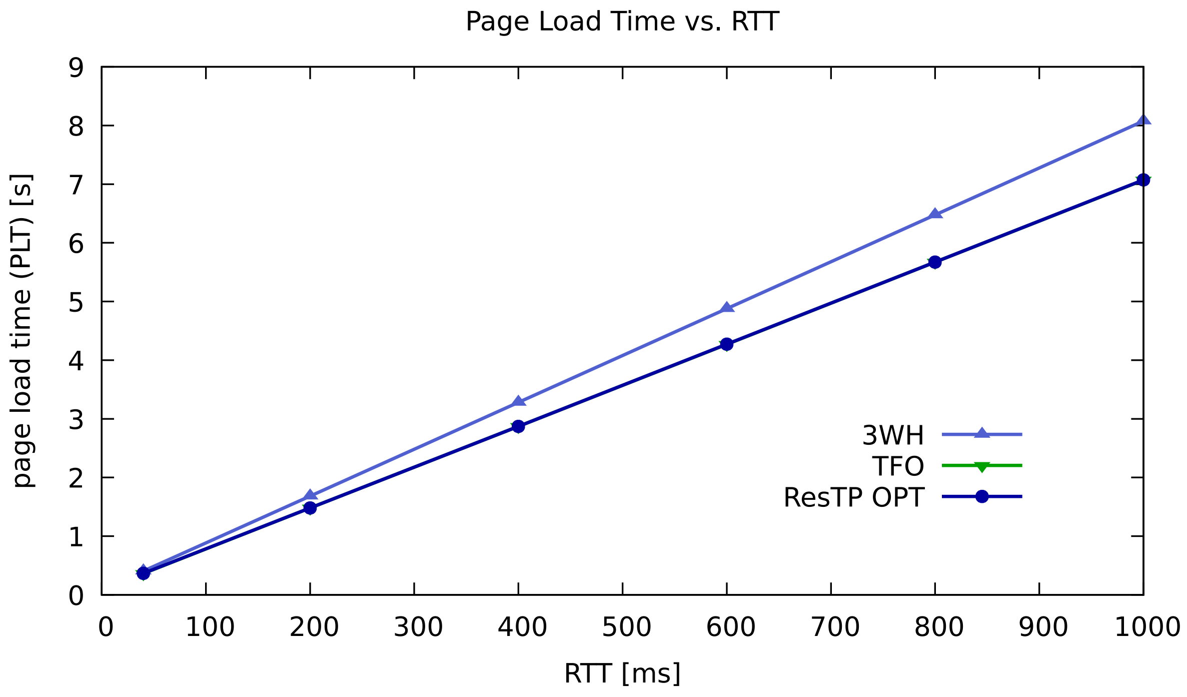

Figure 4.3: PLT comparison under no-loss scenario

Figure 4.4 plots the page load time with increasing RTT when the initial SYN is dropped, and Tables 4.6 and 4.7 calculate the improvement percentage of ResTP OPT over 3WH and TFO, respectively. In this scenario, the opportunistic approach (OPT) has the best performance. Both 3WH and TFO require a SYN retransmission, which only happens when the SYN retransmission timer expires after 3 seconds (the 3-second RTO value is

RTT (ms)	**PLT: 3WH (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	0.407	0.366	10.32%
200	1.683	1.482	11.94%
400	3.279	2.871	12.44%
600	4.877	4.271	12.43%
800	6.476	5.671	12.43%
1000	8.076	7.071	12.44%

Table 4.5: PLT comparison under no-loss scenario showing percentage of ResTP OPT improvement when comparing with 3WH

used by TCP and kept intact in ResTP for the handshake messages when information needed to calculate RTO is still missing). We can verify this 3-second delay by comparing a PLT value of 3WH and TFO in this scenario with the previous no-loss case. The difference between the two PLTs is exactly 3 seconds.

ResTP OPT, on the other hand, does not need to wait for the second SYN. As soon as the server receives the initial page request (the first data packet) from the client, which is sent right after transmitting the SYN, the server can accept the connection and start sending out its responses because the request carries both SYN flag and the cookie.

RTT (ms)	**PLT: 3WH (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	3.407	0.366	89.26%
200	4.683	1.482	68.35%
400	6.279	2.871	54.28%
600	7.877	4.271	45.78%
800	9.476	5.671	40.15%
1000	11.076	7.071	36.16%

Table 4.6: PLT comparison under SYN-loss scenario showing percentage of ResTP OPT improvement when comparing with 3WH

Figure 4.5, Table 4.8, and Table 4.9 compare the performance of the three handshake technique when the initial SYN-ACK is dropped. While 3WH requires the SYN-ACK to be retransmitted after 3 seconds, TFO and ResTP OPT can proceed as soon as the client receives the first in-sequence data packet from the server. We emphasize that while

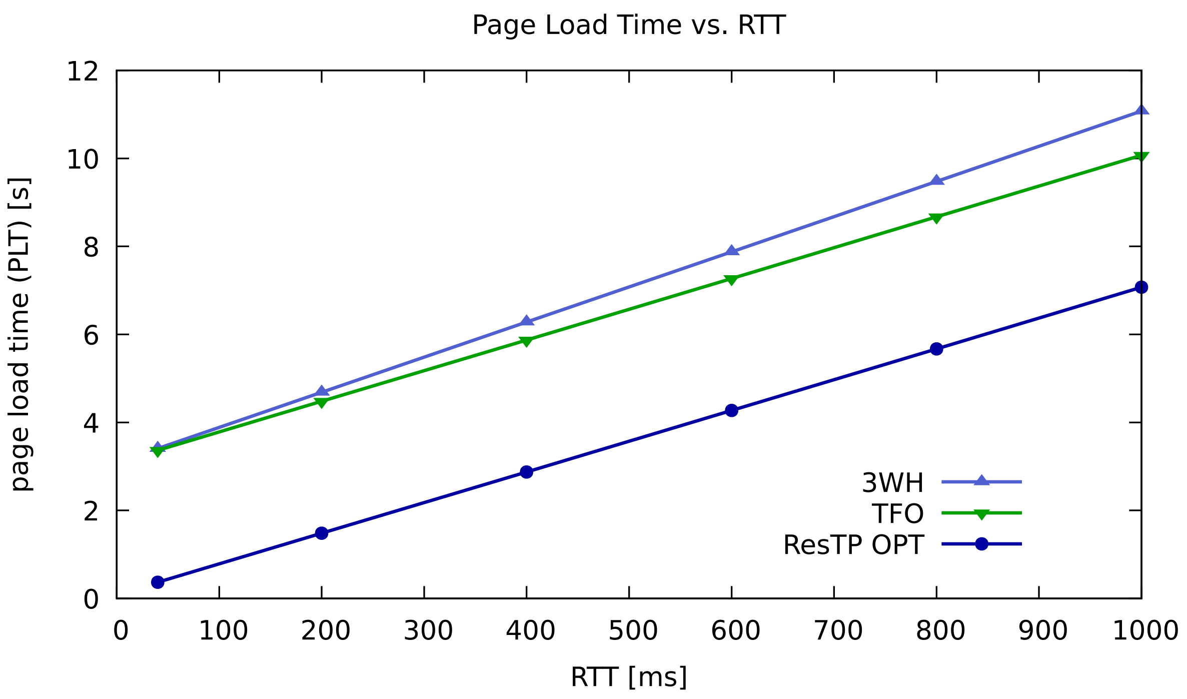

Figure 4.4: PLT comparison under SYN-loss scenario

RTT (ms)	PLT: TFO (s)	PLT: ResTP OPT (s)	Improv.
40	3.365	0.366	89.12%
200	4.480	1.482	66.92%
400	5.871	2.871	51.10%
600	7.271	4.271	41.26%
800	8.671	5.671	34.60%
1000	10.071	7.071	29.79%

Table 4.7: PLT comparison under SYN-loss scenario showing percentage of ResTP OPT improvement when comparing with TFO

ResTP OPT explicitly turns on the SYN-ACK flag for this data packet to allow the data packet act as a regular SYN-ACK in case the SYN-ACK is lost, TFO does not have this feature. RFC 7413 [42] does not discuss how the client would handle a data packet, instead of a SYN-ACK after sending SYN. In our implementation of TFO, we allow the client to accept an in-order data packet in place of a SYN-ACK during its SYN-SENT state and proceed with the final ACK transmission to end the establishment process. Because of this implementation, TFO is able to achieve the same performance as our

ResTP OPT when the SYN-ACK is dropped. ResTP OPT does not cause such confusion in implementation.

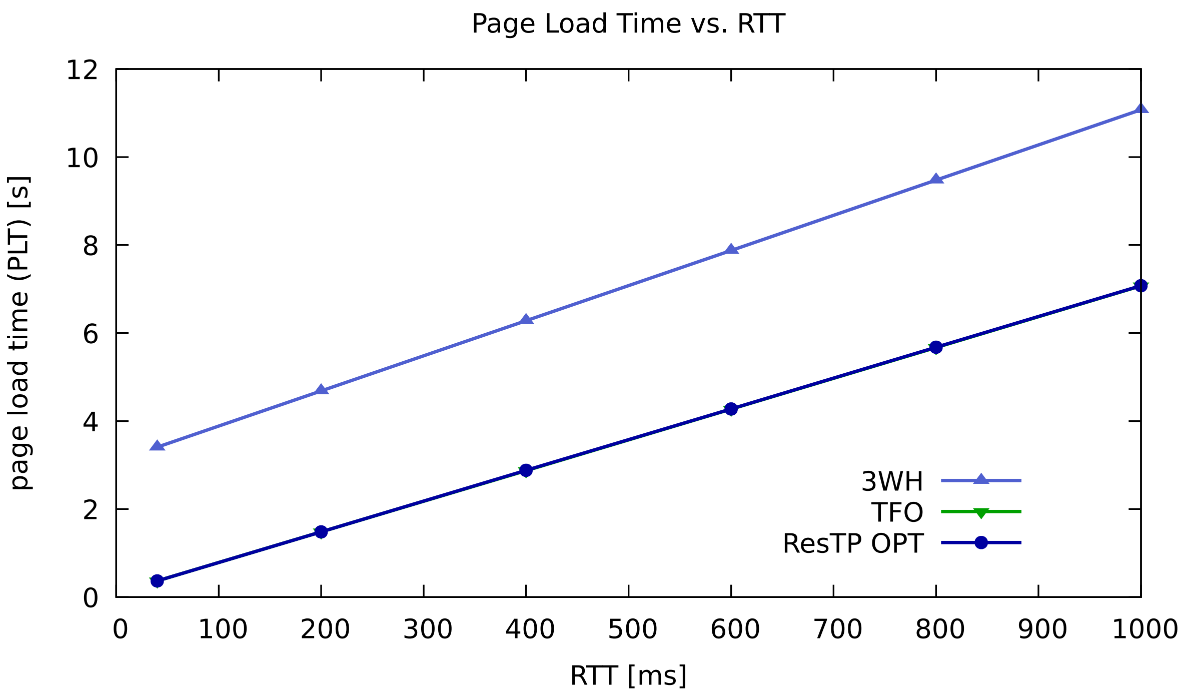

Figure 4.5: PLT comparison underSYN/ACK-loss scenario

RTT (ms)	**PLT: 3WH (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	3.407	0.367	89.23%
200	4.683	1.483	68.33%
400	6.279	2.879	54.15%
600	7.877	4.277	45.70%
800	9.476	5.676	40.10%
1000	11.076	7.076	36.11%

Table 4.8: PLT comparison under SYN/ACK-loss scenario showing percentage of ResTP OPT improvement when comparing with 3WH

Figure 4.6, Table 4.10, and Table 4.11 show the results of the three handshake techniques when both SYN and SYN-ACK are dropped. 3WH suffers the most since it has no mechanism to survive handshake message losses, and it only allows application data to be exchanged after the handshake is completed.

RTT (ms)	**PLT: TFO (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	0.366	0.367	0%
200	1.482	1.483	0%
400	2.872	2.879	0%
600	4.272	4.277	0%
800	5.672	5.676	0%
1000	7.072	7.076	0%

Table 4.9: PLT comparison under SYN/ACK-loss scenario showing percentage of ResTP OPT improvement when comparing with TFO

RTT (ms)	**PLT: 3WH (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	6.407	0.367	94.27%
200	7.683	1.483	80.70%
400	9.279	2.879	68.97%
600	10.877	4.277	60.68%
800	12.477	5.676	54.51%
1000	14.077	7.076	49.73%

Table 4.10: PLT comparison under SYN-and-SYN/ACK-loss scenario showing percentage of ResTP OPT improvement when comparing with 3WH

RTT (ms)	**PLT: TFO (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	3.366	0.367	89.10%
200	4.482	1.483	66.91%
400	5.872	2.879	50.97%
600	7.272	4.277	41.19%
800	8.672	5.676	34.55%
1000	10.072	7.076	29.75%

Table 4.11: PLT comparison under SYN-and-SYN/ACK-loss scenario showing percentage of ResTP OPT improvement when comparing with TFO

In this section, we show that ResTP and its opportunistic connection establishment technique (OPT) addresses TCP 3WH's shortcomings when servicing Web traffic. By allowing application data to be exchanged during the first RTT when the handshake is happening, ResTP OPT saves a traffic flow (especially short flow) a whole RTT. As a resilient transport protocol, ResTP takes a step further to not only reduce latency, but also survive signaling message losses, a capability that is missing in TCP Fast Open

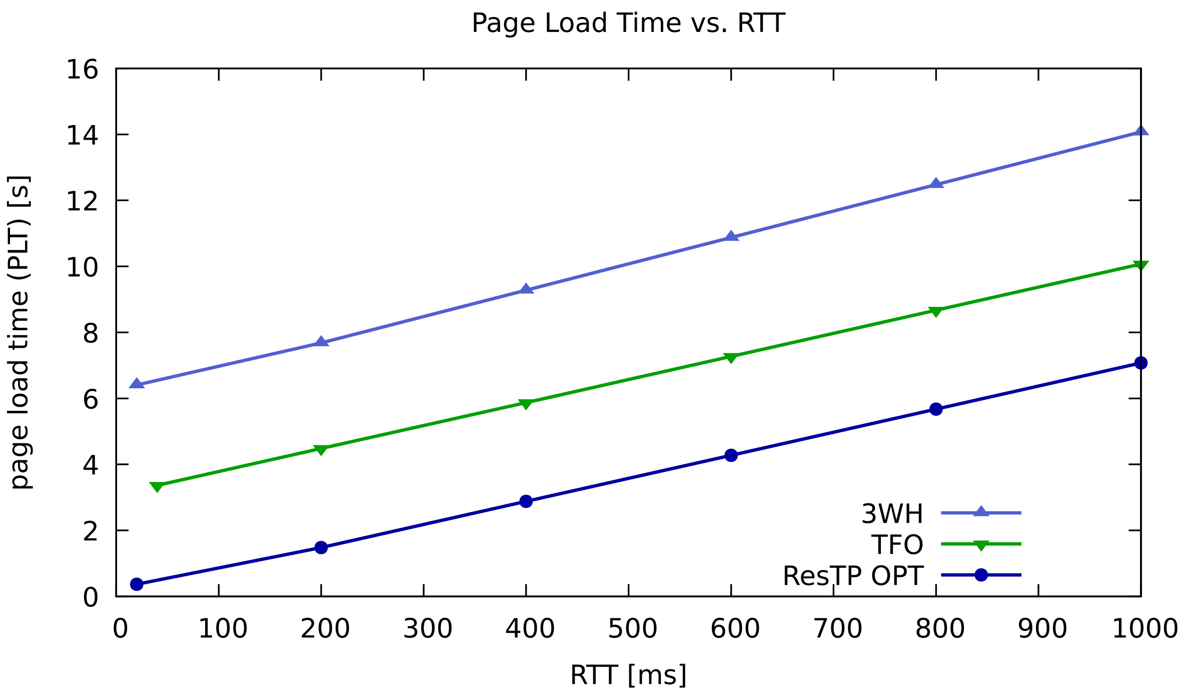

Figure 4.6: PLT comparison under SYN-and-SYN/ACK-loss scenario

(TFO).

4.3 Multipath ResTP with Web Services

In this section, we evaluate the performance of ResTP opportunistic handshake when ResTP operates in its multipath mode. We compare the performance of ResTP OPT with the standard three-way handshake employed by MPTCP using the same HTTP model used in the previous single-path study.

4.3.1 Simulation Setup and Topology

Figure 4.7 illustrates the topology we use in this set of simulations. All links have the same data rate of 2 Mb/s. The access link between the hosts and the routers have a negligible delay of 0.1 ms while the links connecting the routers have varying delays.

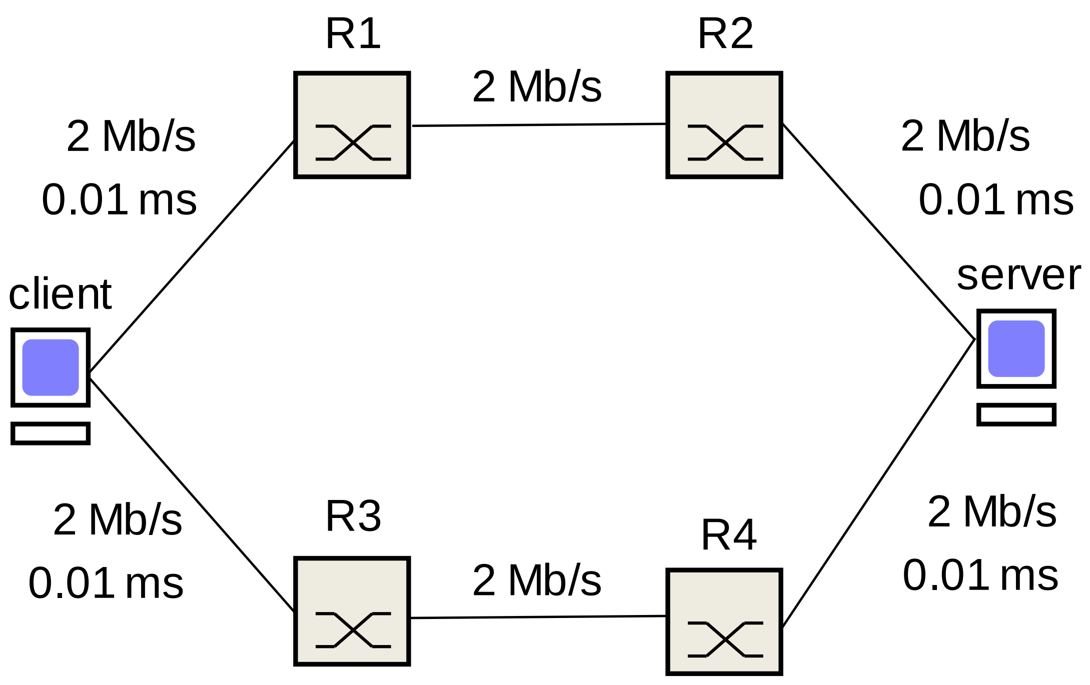

Figure 4.7: Multipath topology for comparing ResTP in its multipath mode and MPTCP

ResTP is configured with the same set of algorithms employed by MPTCP, except the handshaking technique. Every main object generated by the HTTP server has a minimum size of 100 bytes and a maximum size of 2 KB.

4.3.2 Simulation Results and Analysis

Tables 4.12 and 4.13 show the page load time (PLT) for ResTP OPT and MPTCP three-way handshake (3WH) when there are no losses and when the initial SYN is dropped. In both scenarios, the opportunistic handshaking technique helps reduce the PLT. The improvement percentage is proportional with the network latency. Especially when SYN is dropped, ResTP OPT performs much better than 3WH because it does not require a SYN retransmission for completing the handshaking.

RTT (ms)	**PLT: 3WH (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	1. 771	1.726	2.54%
60	1.831	1.766	3.55%
80	1.891	1.806	4.49%
100	1.951	1.846	5.38%
200	2.251	2.046	9.11 %

Table 4.12: PLT comparison under no-loss scenario showing percentage of ResTP OPT (MPResTP) improvement when comparing with 3WH (MPTCP)

RTT (ms)	**PLT: 3WH (s)**	**PLT: ResTP OPT (s)**	**Improv.**
40	4. 771	1.726	63.82%
60	4.831	1.766	63.44%
80	4.891	1.806	63.08%
100	4.951	1.846	62.71%
200	5.251	2.046	61.04%

Table 4.13: PLT comparison under SYN-loss scenario showing percentage of ResTP OPT (multipath mode) improvement when comparing with 3WH (MPTCP)

4.4 Single-Path ResTP over Satellite Communications

In this set of simulations, we compare the performance of ResTP with TCP in a satellite network that has no assurance of an episodic connectivity. Our simulations demonstrate that as a resilient transport protocol, ResTP has a mechanism to handle link outages and achieve better performance than TCP in this scenario.

4.4.1 Simulation Setup and Topology

In our simulation, we use Satellite Network Simulator 3 (SNS3), a satellite network extension to the ns-3 platform [95,96]. This module implements a full interactive multi-spot beam satellite network with a GEO (geostationary satellite) and transparent star (bent-pipe) payload and adopts the Digital Video Broadcast - Return Channel via Satellite - 2nd generation (DVB-RCS2) and Digital Video Broadcasting - Satellite - 2nd generation (DVB-S2) communication standards to model the return and forward links.

Our simulation topology consists of a single user terminal (UT) and a single gateway (GW) communicating through a GEO satellite located at a latitude of 0.0, longitude of 33.0, and an altitude of 35786000.0. Unlike UT, GW has multiple net devices (`ns3::SatNetDevice`) to serve multiple spot-beams. In our simulation, we enable only one spot-beam to provide service for the only UT we create. When we simulate a link outage, we bring down both the forward and return satellite links. To emphasize the impact of link outage handling at the transport layer, we disable link-layer ARQ on both up and down directions.

A data generator (source) is connected with the UT using a CSMA channel. This source transmits traffic as fast as possible to a server located on the GW side using `ns3::BulkSendApplication`, until the application is stopped at the end of our 1000-second simulation duration.

We simulate three link failure events: a 10-second outage from second 20 to second 30, a 100-second outage from second 200 to second 300, and a 50-second outage from second 600 to second 650. A tracker is installed at the server to compute the instantaneous throughput every 50 seconds.

4.4.2 ResTP Configuration

ResTP is configured to use the same NewReno congestion control algorithm, 3-way handshake connection establishment, and ARQ error control technique as in TCP (Table 4.14). However, ResTP's 'source-of-loss' parameter is set to link outage (instead of congestion or corruption). This setting allows ResTP to enable its link outage handling technique when a loss happens as described in Section 3.6.

CM	**FC**	**CC**	**Ordering**	**Reliability**	**ACK**
3WH	windowed	NewReno	ordered	fully reliable with ARQ & periodic probe	positive ACK

Table 4.14: ResTP configuration for handling link outages in satellite communications

4.4.3 Simulation Results and Analysis

Figure 4.8 plots the instantaneous throughput of TCP and ResTP. ResTP achieves much higher throughput than TCP because when in its link-failure mode, ResTP does not reset its sending rate. Instead, ResTP only suspends its normal data transmission and starts sending probe packet. Once the link is recovered, ResTP resumes its transmission at the same rate as it did before the failure.

Comparing Figure 4.9 to Figure 4.10 in that we trace all the packets received at the sink, we can see that while ResTP is able to survive all three outages and continues to transmit

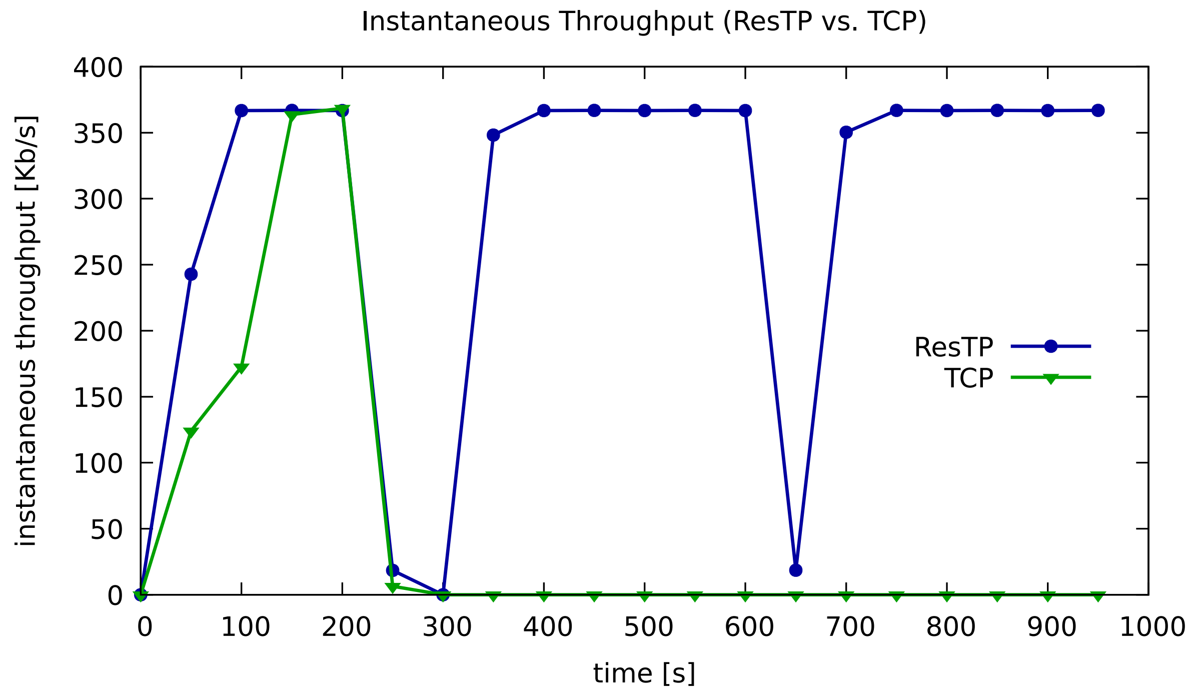

Figure 4.8: Instantaneous throughput of ResTP and TCP with multiple outages of the satellite link

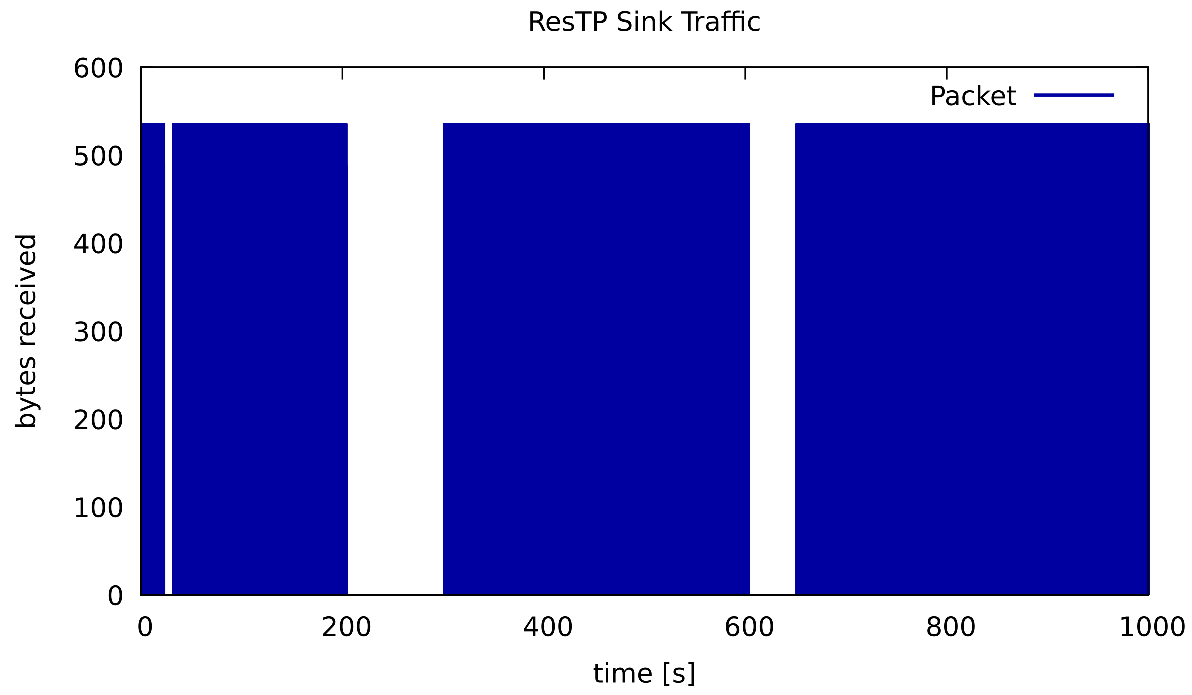

Figure 4.9: ResTP sink traffic trace with multiple outages of the satellite link

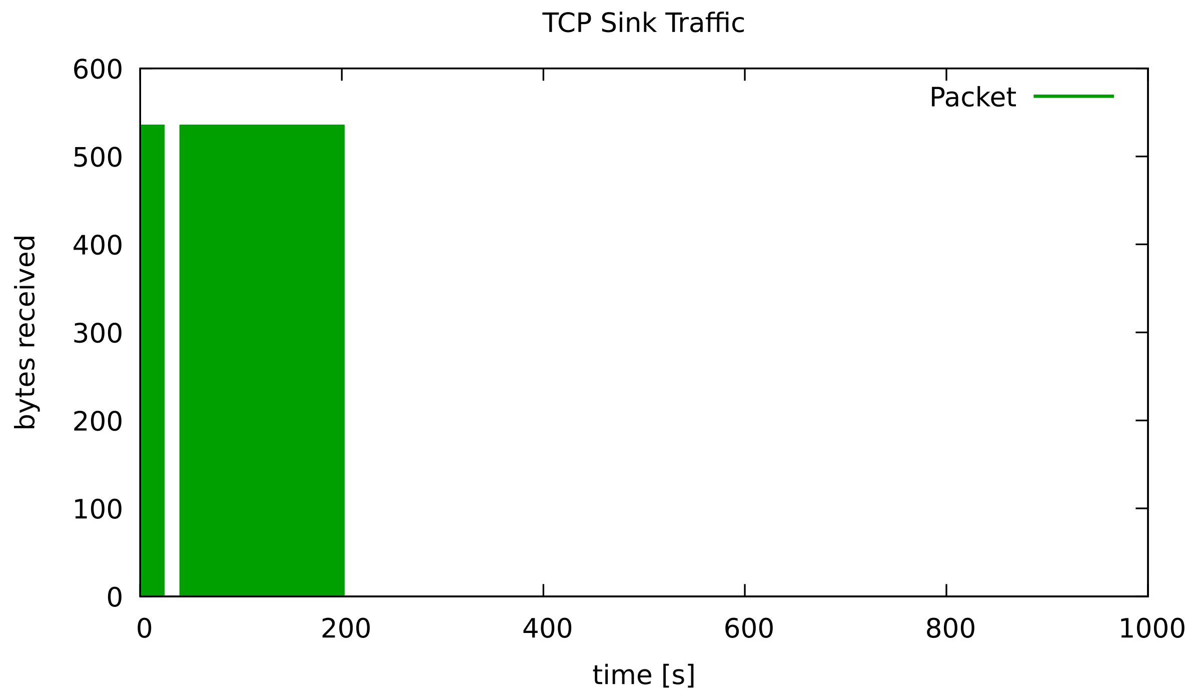

Figure 4.10: TCP sink traffic trace with multiple outages of the satellite link

traffic until the end of the simulation, TCP is only able to survive the first outage. At the second link failure happening at second 200, due to the long outage duration, TCP finally runs out of its data retransmission attempts and has to abort the whole connection. This explains the 0-throughput of TCP after the second 300 shown in Figure 4.8.

Figure 4.11 plots all the RTO events during the 1000-simulation duration, the time at which each event happens and the RTO value when the timer is expired. Every time an RTO occurs, TCP doubles its value for the next retransmission. On the other hand, ResTP does not back off its timer. ResTP also uses RTO to time its probe packet transmission, which explains the many RTO events happening during the three link outage periods in the ResTP plot.

In this section, we show that ResTP, as a transport protocol designed for different network environments, works better than the traditional TCP in satellite networks when link

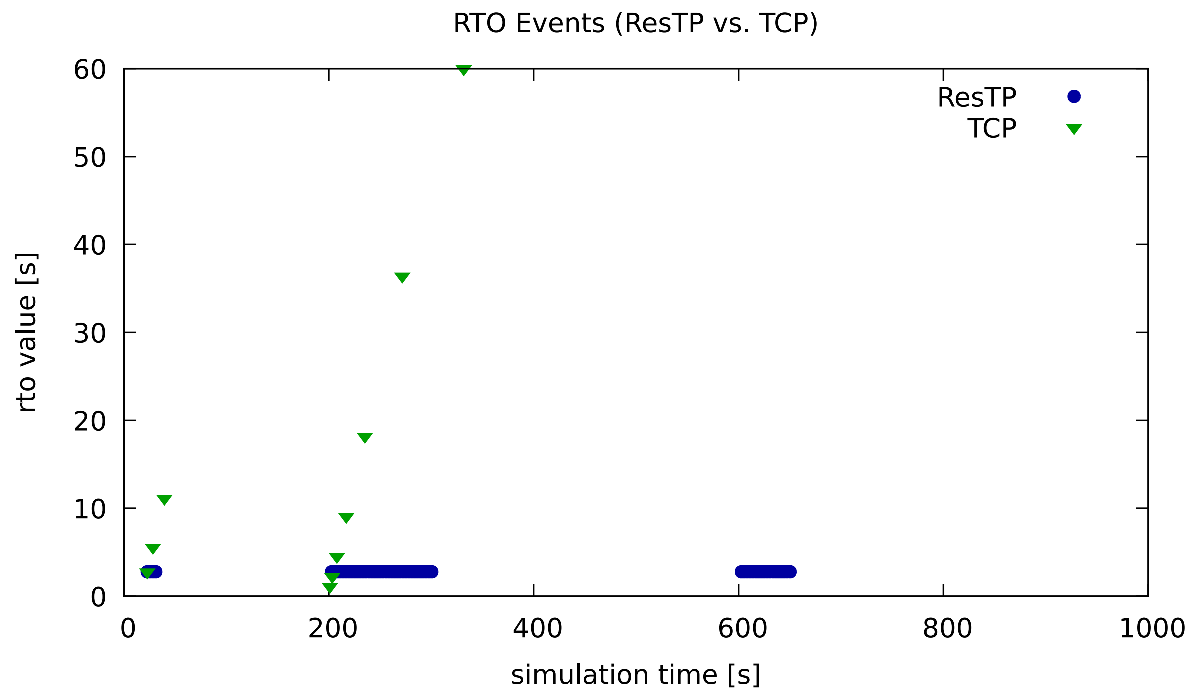

Figure 4.11: ResTP vs. TCP RTO events

outages occur. When the duration of an outage is short, ResTP achieves higher thoughput than TCP. When the duration of an outage is long, ResTP is able to survive the loss while TCP has to abort its whole connection.

4.5 Single-Path ResTP with Real-Time Applications

In this set of simulations, we compare the performance of ResTP with UDP and TCP when they transport real-time multimedia data. The simulation analysis serves two purposes:

- It shows that ResTP can be configured to satisfy the application's transport-layer preference without any modifications required to the application.

- It shows that ResTP can be configured to achieve both timeliness and reliability, a property that cannot be found in UDP or TCP.

4.5.1 Simulation Setup and Topology

The topology we use in this set of simulations is illustrated in Figure 4.12. The two access links connecting our voice sender and receiver with the two routers R0 and R1 have a bandwidth of 100 Mb/s with a negligible delay. The bottleneck link connecting the two routers is where we introduce extra latency and packet losses into the network. For the results presented in Table 4.18, the bottleneck link has a delay of 30 ms and a loss rate of 0.02 PER.

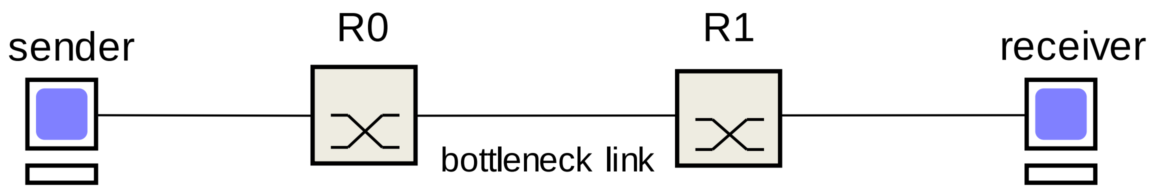

Figure 4.12: Topology for simulating ResTP vs. UDP and TCP with VoIP traffic

We use the voice model proposed for the LAA Wi-Fi coexistence project in ns-3 [97] with a few modifications to suit our needs, including the capability to use UDP, TCP, and ResTP as the transport-layer protocols. This voice model generates VoIP traffic based on a configurable packet size and packet transmission interval. These two attributes are calculated according to the voice codec we want to study.

Our experiments use the G.711 VoIP codec technology with a bit rate of 64 Kb/s, a voice payload size of 160 bytes, and a voice packet inter-arrival time of 20 ms.

The application exports trace sources for each packet that it sends and receives. Every packet carries a sequence number and a timestamp. Based on a packet's timestamp (the time the client transmits the packet) and its arrival time at the server, its latency can be calculated. A packet is lost if it causes a gap in the data sequence passed to the application, and a packet is delayed if its latency exceeds a predefined threshold, which is set to 160 ms for our study. This latency threshold is calculated based on the playback delay calculation suggested for TCP-RTM. Basically, when TCP and fully-reliable ResTP are used, some delay generated due to the ARQ technique is inevitable. This latency is proportional to the network RTT. It takes at least 3/2 RTT for the voice sender to learn about a loss and for the retransmitted packet to reach the receiver.

4.5.2 ResTP Configuration

For this set of simulations, we compare three different ResTP configurations with UDP and TCP.

CM	**FC**	**CC**	**Ordering**	**Reliability**	**ACK**
none	none	none	none	none	none

Table 4.15: Unreliable ResTP (UDP-like) configuration for multimedia real-time applications

ResTP-3WH-FUL-ARQ (Table 4.16) employs the exact mechanisms adopted by TCP. We include this configuration to show that ResTP can achieve comparable performance to TCP, although not ideal for real-time multimedia applications.

CM	**FC**	**CC**	**Ordering**	**Reliability**	**ACK**
3WH	windowed	BIC	ordered	fully reliable with ARQ	positive ACK

Table 4.16: ResTP-3WH-FUL-ARQ configuration for multimedia real-time applications

ResTP-3WH-PAR-ARQ (Table 4.17) provides a partially-ordered data delivery service. This configuration is expected to perform better than ResTP-3WH-FUL-ARQ and TCP and is designed for those multimedia applications that usually prefer the full set of services (congestion, error, and flow control) provided by TCP but do not want to trade reliability for timeliness.

CM	**FC**	**CC**	**Ordering**	**Reliability**	**ACK**
3WH	windowed	BIC	partially-ordered	partially reliable with ARQ	positive ACK

Table 4.17: ResTP-3WH-PAR-ARQ configuration for multimedia real-time applications

4.5.3 Simulation Results and Analysis

Table 4.18 summarizes the number of voice packets sent, received, lost, and delay along with the received and sent ratio, the latency mean, and latency standard deviation for UDP, TCP, and the three ResTP configurations included in this study. The bottleneck delay is 30 ms (equivalent to a 60-ms RTT), and the PER is 2%. Every simulation has a duration of 120 ms.

Protocol	**Sent**	**Rcvd**	**Rcvd/Sent**	**Lost**	**Delayed**	**Lat.Mean**	**Lat.Stddev**
UDP	4949	4862	0.982	87	0	32.046	0.000
ResTP-unreliable	4949	4862	0.982	87	0	32.048	0.000
TCP (without SACK)	4949	4178	0.844	559	212	49.015	55.709
TCP (with SACK)	4949	4394	0.888	459	96	41.924	35.627
ResTP-3WH-FUL-ARQ	4949	4773	0.964	0	176	52.096	73.228
ResTP-3WH-PAR-ARQ	4949	4922	0.995	3	24	37.130	19.440

Table 4.18: Voice flow statistics for UDP, TCP, and different ResTP configurations when RTT is 60 ms, and PER is 0.02.

First, when ResTP is configured as a light-weight protocol that only performs simple multiplexing/demultiplexing as UDP, it performs as well as the standard protocol (ResTP-unreliable vs. UDP). The two protocols correct no packet losses as we can see that the 2% network loss rate is reflected by the 87 (out of 4949 packets sent) packets lost at the application layer.

Second, we can see the impact of SACK on error correction when comparing the performance of TCP with and without SACK enabled. SACK allows the receiver to inform the sender the exact lost packets.

Third, correct framing impacts the overall performance. ResTP-3WH-FUL-ARQ employs the same set of algorithms as TCP (with SACK), except that its framing module 3.5.3 can ensure that each application-layer packet is written and read as a separate ResTP segment.

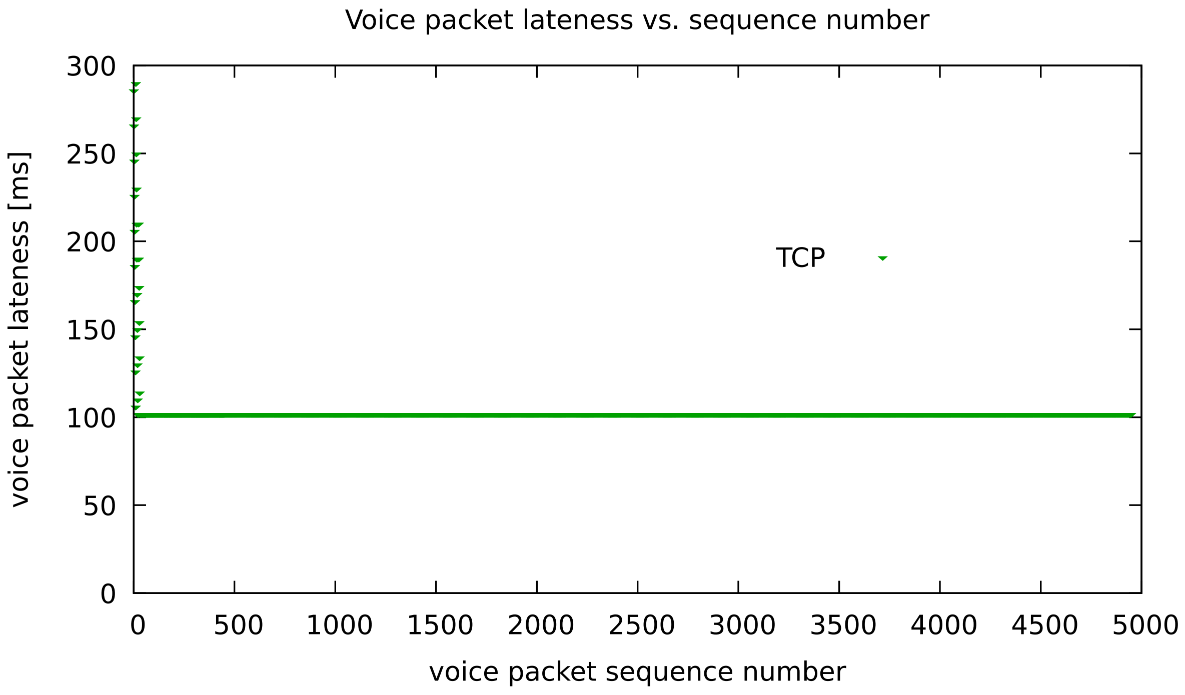

Figure 4.13: Voice packet lateness vs. sequence number when using TCP with three-way handshake

Fourth, if we want to achieve both timeliness and reliability, ResTP-3WH-PAR-ARQ is the configuration to use. Out of the 87 lost packets, the partially-reliable mechanism can correct 84 of them although this error correction capability results in 24 packets delayed. However, overall, ResTP-3WH-PAR-ARQ achieves the least amount of packets lost and delayed. In addition, some of the delayed packets are not caused by the ARQ technique,

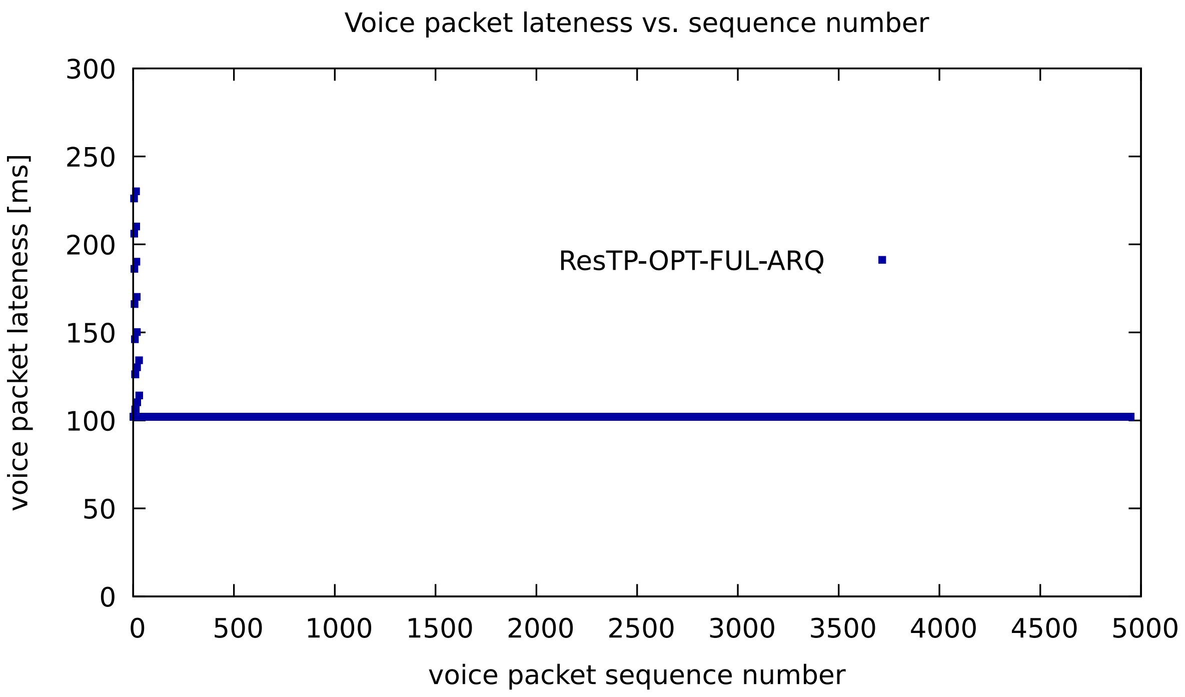

Figure 4.14: Voice packet lateness vs. sequence number when using ResTP with opportunistic handshake

but the three-way handshake. In Figures 4.13 and 4.14, we plot the delays of all packets against their sequence numbers when there are no errors introduced into the network. We can see that in this scenario, all packet delays occur at the beginning of the connection, and the number of delayed packets are reduced when ResTP OPT is used. Hence, we can replace the 3WH in ResTP-3WH-PAR-ARQ with ResTP OPT to further improve the performance, especially to protect the protocol against an initial SYN loss.

Finally, when losses occur, the congestion control algorithm that we select for ResTP also plays an important role in its overall performance. We note that here, we select BIC instead of NewReno because NewReno always reduces the sender's transmission rate by half without considering the source of packet losses.

4.6 ResTP-GeoDivRP Under Network Challenges

In the last set of simulations, we perform some preliminary simulations to demonstrate the ResiliNets stack (Figure 3.28)'s performance in face of regional challenges in comparison to MPTCP [98]. Through these simulations, we show that with the geographically-separated paths provided by GeoDivRP, the ResTP-GeoDivRP stack can achieve higher throughput and better resilience than MPTCP when challanges occur.

More details about this comparison and the GeoDivRP design and implementation can be found in one of our publications [99]. [1]

4.6.1 Simulation Setup and Topology

To study the performance of the ResTP-GeoDivRP protocol stack under different network challenges, a set of challenge profiles have been developed. Figure 4.15 illustrates these profiles on the Sprint physical network [100]. The green circles depict the Midwest challenge profile, which represents a super storm sweeping from the southwest to the northeast direction. The blue circles depict the coastline challenge profile, which represents a hurricane on the East Coast. The red circles depict the cascading challenge profile, which represents a power blackout affecting a region growing in size.

In this work, we focus on the impact of the cascading profile (red circles) affecting the region where the most shortest paths occur. Figure 4.16 shows the Sprint topology with the three challenge scenarios originating around Nashville and growing larger in range. The small green circle challenge occurs at $20 - 40$ s, the yellow circle challenge occurs at $60 - 80$ s, and the large red circle one occurs at $100 - 120$ s.

[1] My contribution for this work is only the initial multipath ResTP implementation, which follows the same MPTCP model used for comparison with ResTP. Simulations and other work are contributed by the other authors.

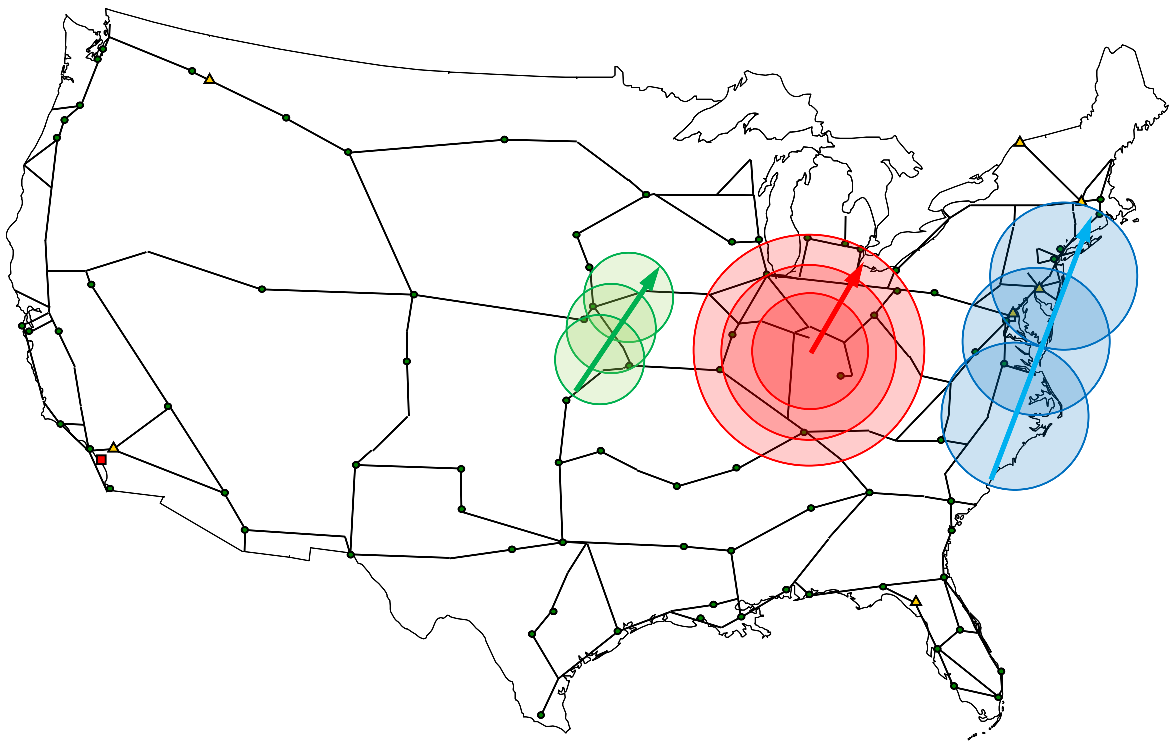

Figure 4.15: Sprint network topology challenge profile

Traffic starts from node 1 at Oklahoma City to node 3 at Washington D.C. Each link has a bandwidth of 100 Mb/s.

The dashed lines represent the three node-disjoint paths calculated using Suurballe's algorithm [101,102] used by MPTCP for transmitting the traffic across the network. The solid lines represent the three geographically-disjoint paths calculated by GeoDivRP and utilized by ResTP. While MPTCP uses St. Louis, Kansas City, and Atlanta as its next hops for the three subflows, ResTP uses Omaha, Nashville, and Houston as the next hops.

4.6.2 ResTP Configuration

In this early work of the ResiliNets protocol stack development, ResTP has a similar implementation as MPTCP [98]. Its number of paths **k** header field is set to 3, and the

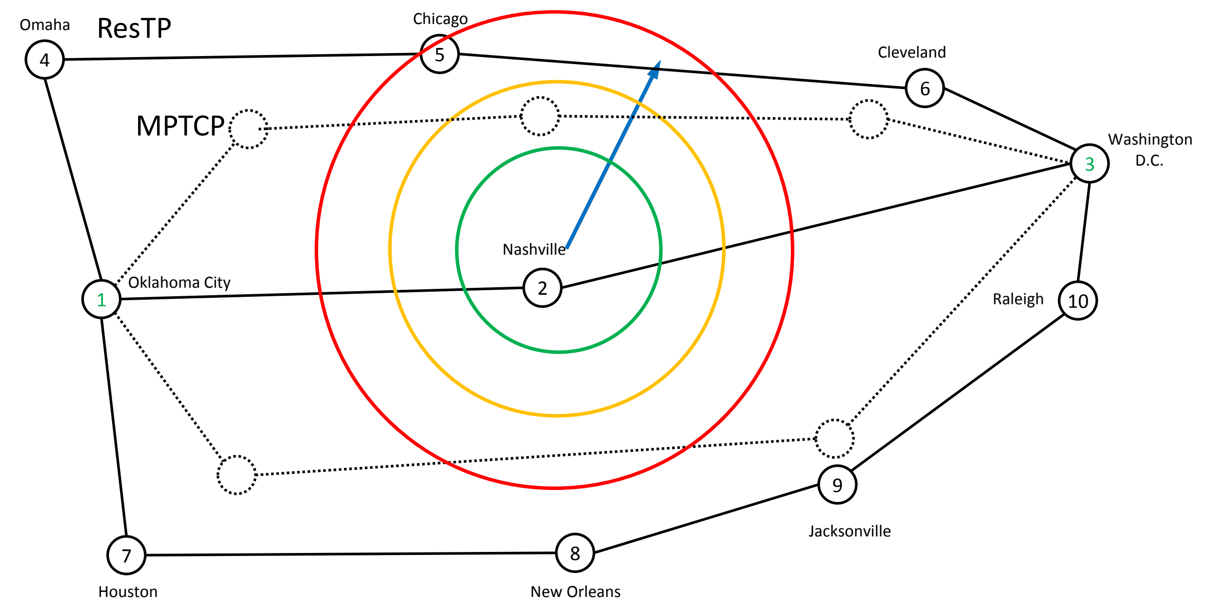

Figure 4.16: Sprint network topology with multiple paths (dashed lines are paths used by MPTCP, and solid lines are paths used by ResTP)

protocol operates in its spreading mode with all the three paths utilized for transporting traffic. The only main difference between ResTP and MPTCP in this simulation is the paths that they use for data transfer.

4.6.3 Simulation Results and Analysis

Figure 4.17 shows the average throughput of ResTP in comparison with MPTCP when the three cascading challenge scenarios are applied to the Sprint network.

At 20 s when the first challenge starts, the throughput of both ResTP and MPTCP are degraded with the failure of the shortest path used by both protocols. From 60 − 80 s when the second challenge occurs, MPTCP achieves only half of the throughput achieved by ResTP because it looses 2 out of the 3 subflows in operation. From 100 − 120 s, the throughput of MPTCP reaches 0 since all the three paths that it uses fail due to the third challenge. Overall, ResTP (by using the paths provided by GeoDivRP) achieves around

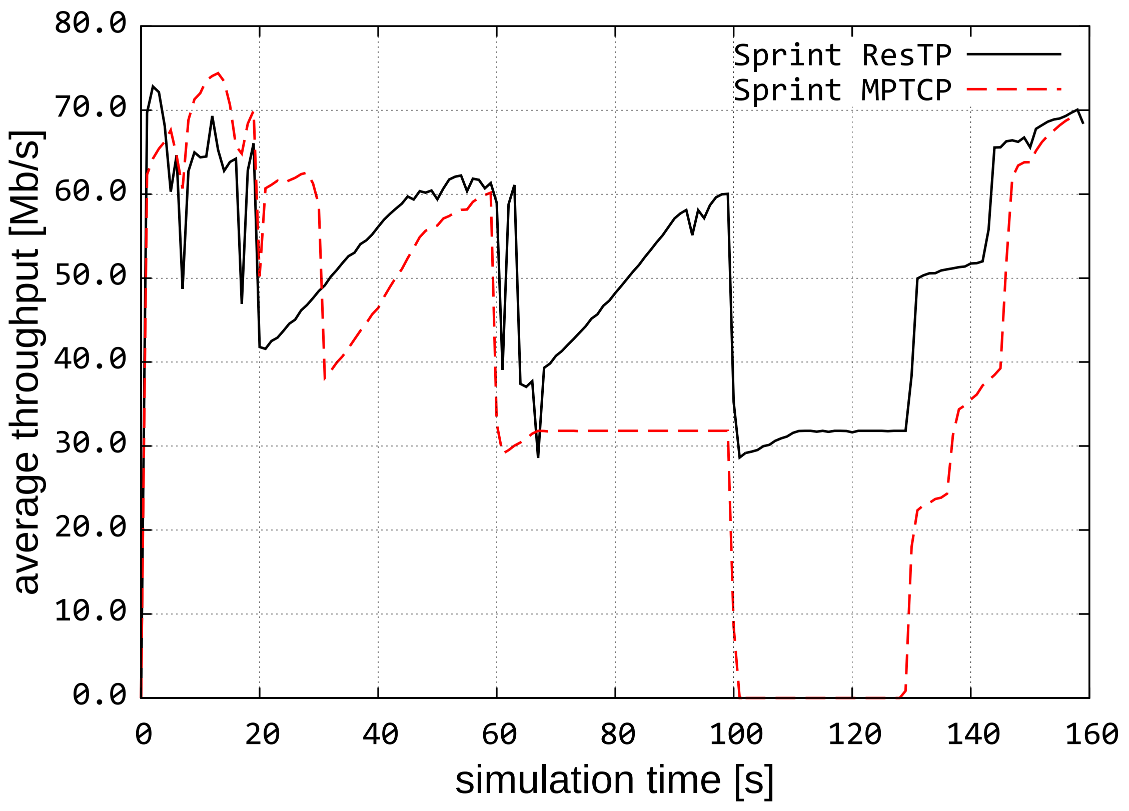

Figure 4.17: Sprint network ResTP throughput compared to MPTCP

30% to 40% higher throughput than MPTCP when using the geographically-disjoint paths provided by GeoDivRP.

Page left intentionally blank.

Chapter 5

Why ResTP and Its Deployability?

When we design ResTP, we are fully aware that the ossification of the current Internet's transport layer has limited networked applications to choose only UDP or TCP despite their limitations and the presence of many better transport protocols having been proposed. However, as we have seen, UDP and TCP are too constrained for the Future Internet (FI), in which we expect to see new technology and application types emerge. Every time a new network challenge arises or a new application becomes the killer app, these protocols, especially TCP will have to be modified and extended. Even an extension of TCP is limited by its allowed option length and its rigidly intertwined flow/error/congestion control property. Furthermore, although we have seen a plethora of TCP variants (and we expect to see a plethora of MPTCP variants) proposed to keep the standard protocol up-to-date with the networking advancements, not many of them have gained wide acceptance and adoption in practice.

We believe that the FI needs a more flexible transport-layer protocol than the current UDP, TCP, or even MPTCP. This flexibility allows the protocol to smoothly evolve with the advancement of networking technology, allowing it to serve different types of applications and operate over different types of networks. This premise motivated the design of ResTP.

We understand that for ResTP to be widely adopted and deployed, more studies are needed. Moreover, the successful deployment and adoption of a transport-layer protocol depend on several factors, considering the complexity of the Internet [103]. However, based on our simulation analysis and results presented in this dissertation, we believe that ResTP is a potential candidate for the FI.

We note that ResTP is also very useful in transport-layer protocol studies and research given that new algorithms can be easily implemented and tested in ResTP.

Chapter 6

Conclusion and Future Work

In this dissertation, we present the design, implementation, and evaluation analysis of a general-purpose, composible transport-layer protocol of which the development is solely motivated by the determination to increase resilience and survivability. The composibility of ResTP allows the protocol to provide different sets of services to different application classes depending on their specific requirements. All ResTP components are pluggable and completely independent of each other. The modification of a component should not affect the others and even the failure of a component should allow the protocol to continue to operate.

Our simulation results show that ResTP can be utilized the same way as TCP and UDP and achieve a similar performance under the same network environments. However, unlike TCP and UDP with a fixed set of services, ResTP can have multiple configurations. The opportunistic connection establishment is shown to perform better than the conventional 3-way handshake and TCP Fast Open. The multipath mode of ResTP, when incorporating with the GeoDivRP routing protocol, forms a ResiliNets protocol stack that provides higher throughput and better resilience than MPTCP in face of regional challenges.

Specifically, we show that:

- ResTP can perform as well as TCP when handling file transfer. With this type of traffic, ResTP can be configured with either ARQ and HARQ.
- ResTP with its opportunistic connection establishment handles web transfers better than TCP and MPTCP because this handshaking technique takes less time to complete than the standard three-way handshake, therefore reducing the overall flow completion time for short flows. ResTP even performs better than TCP Fast Open when signaling messages are lost.
- ResTP provides multiple configurations for real-time multimedia applications such as VoIP. ResTP-unreliable is suitable for those applications that prefer the simplicity of UDP, and ResTP-3WH-PAR-ARQ is suitable for those applications that concern about the network congestion and corruption, but also require timeliness in data delivery. ResTP-unreliable performs as well as UDP, and ResTP-3WH-PAR-ARQ performs much better than TCP.
- ResTP can survive link outage in a network that has no assurance of an episodic connectivity such as the satellite communication while TCP cannot.
- ResTP-GeoDivRP can survive different network challenges when operating together and exploiting cross-layering.

For our future work, we plan to design and implement dynamic data loss identification for ResTP so that the protocol can learn about different type of losses and adjust its configuration in run time. We also plan to extend our study of ResTP in its multipath mode. We also want to implement ResTP in the Linux kernel and study the protocol using real networks.

Appendix A

ResTP Implementation, Verification, and Validation in ns-3

We implemented ResTP in an open-source network simulator for a better and easier evaluation of our design. We can obtain valuable insights about the behaviors of ResTP across a wide range of network conditions and traffic classes using simulation. We choose ns-3 [6] as our platform due to the variety of network technologies that it supports. The ns-3 system is actively maintained and up-to-date with new models being reviewed and merged on a regular basis. Furthermore, it can interact with the real world, creating networks that contain both real and simulated components, which is a useful feature for our study of ResTP in the future. In addition, this implementation of ResTP in ns-3 provides a prototype of functional protocol software.

A.1 Implementation

Our implementation of ResTP in ns-3 is greatly inspired by the TCP module in this platform and the recent TCP refactoring work that splits the congestion component from the main socket `TcpSocketBase` class as part of the ns-3 community's ongoing effort to modularize the TCP module and simplify its base code [104]. The multipath extension of ResTP is based on the recent MPTCP model [105, 106] that is undergoing the ns-3 review process for merging into the standard release.

Because ResTP is designed as a flexible protocol that is capable of supporting alternative algorithms, its implementation needs to be highly modular to reduce the complexity and increase the extensibility of the model should the protocol be extended to accommodate new functionalities. Each service supported by ResTP (flow control, error control, congestion control, ...) is an independent and pluggable software component of the whole protocol so that the unplugging of any of them should leave the rest intact. This approach allows the entire ResTP module to be easily extended when needed, and to be maintained with the minimal cost.

The ResTP model resides in the same ns-3 Internet module as TCP and UDP. Figure A.1 illustrates some of the main ResTP classes and their relationships.

- `ResTPSocket`: The abstract class `ResTPSocket` is the base class of all ResTP sockets.

- `ResTPSocketBase`: `ResTPSocketBase` is a subclass of `ResTPSocket`. It provides a single-path socket interface to the upper application layer. `ResTPSocketBase` also connects with and collaborates the operations of individual components that implement different ResTP transport-layer services.

- `MpResTPSocketBase`: `MpResTPSocketBase` has the same functionalities as `ResTPSocketBase`, but for multi-path ResTP sockets.

- `ResTPL4Protocol`: `ResTPL4Protocol` handles ResTP socket creation and multiplexing (sending packets down the stack) and demultiplexing (receiving packets from the lower network layer and forwarding them to the correct ResTP socket).

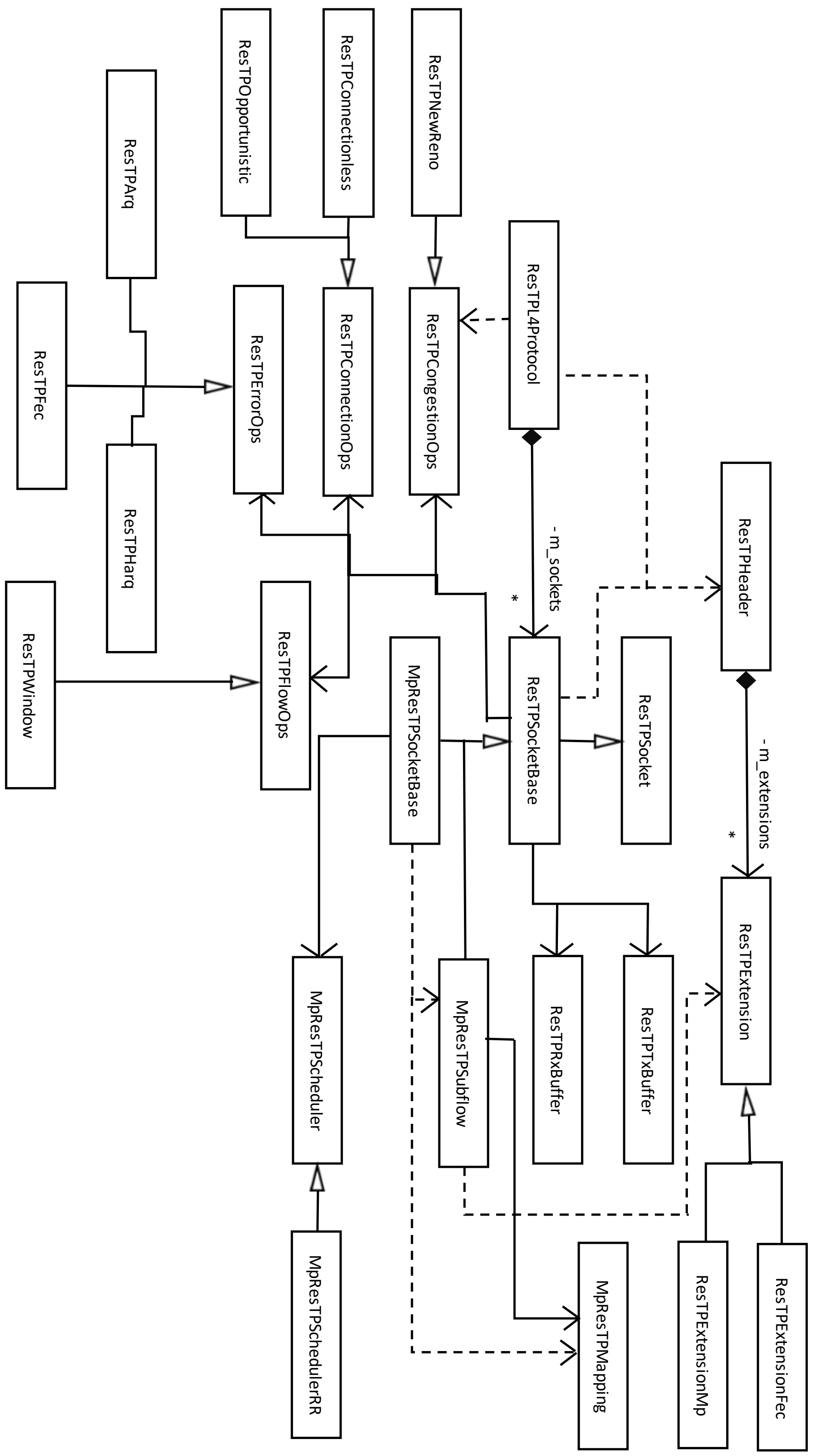

Figure A.1: ResTP class diagram in ns-3

- `ResTPTxBuffer`: `ResTPTxBuffer` implements the ResTP sending buffer used to store all the data that the sending application wishes to transmit across the network. Depending on what services are enabled for a particular ResTP configuration, this buffer handles stored data differently. For example, if quasi-reliable data delivery is enabled, the buffer removes originally transmitted packets after they are encoded inside an FEC packet. On the other hand, if fully-reliable data delivery is required, transmitted packets are deleted from this buffer only after they are acknowledged.

- `ResTPRxBuffer`: `ResTPRxBuffer` implements the reordering buffer for ResTP. Similar to `ResTPTxBuffer`, `ResTPRxBuffer` behaves differently according to different services.

- `ResTPHeader`: `ResTPHeader` implements all the fields in the basic ResTP header described in Figure 3.1. It also includes methods for appending additional extenstions to the basic header depending on how ResTP is configured.

- `ResTPExtension`: `ResTPExtension` is the base class for all ResTP extension implementations, including `ResTPExtensionFec` and `ResTPExtensionMp` shown in the class diagram and others defined and explained in Section 3 but not shown due to space limitation and readability purpose.

- `ResTPCongestionOps`: `ResTPCongestionOps` is the ResTP congestion base class. Any specific congestion control algorithm for both single-path and multi-path ResTP such as the NewReno algorithm (`ResTPNewReno`) shown in the class diagram should be implemented as a derived class of this class.

- `ResTPErrorControlOps`: `ResTPErrorControlOps` is the ResTP error control base class. Any specific error control algorithm (`ResTPFec` for FEC, `ResTPArq` for ARQ, or `ResTPHarq` for HARQ) should be implemented as a derived class of this class.

- `ResTPConnectionOps`: `ResTPConnectionOps` is the ResTP connection management base class. Each connection establishment scheme supported by ResTP, including our proposed opportunistic algorithm (`ResTPOpportunistic`) should be implemented as a derived class of `ResTPConnectionOps`.

- `ResTPFlowOps`: `ResTPFlowOps` is the ResTP flow control base class. Each flow control algorithms supported by ResTP should be implemented as a derived class of this class.

- `MpResTPSubflow`: `MpResTPSubflow` handles individual subflow of a multipath ResTP connection.

- `MpResTPMapping`: `MpResTPMapping` maps ResTP connection-level sequence number (DSN) to the subflow-level sequence number (SSN).

- `MpResTPScheduler`: `MpResTPScheduler` is the base class for all multipath ResTP scheduling implementations in ns-3. Any specific scheduling algorithm such as the round robin scheduler (`MpResTPSchedulerRR`) shown in the class diagram should be implemented as a derived class of this class.

A.2 Verification

We verify the correctness of our ResTP implementation by writing several unit tests to ensure all the implemented functionalities of ResTP working as intended. Each of these unit tests is created as a TestSuite class that inherits from the base class `TestSuite`. Each test suite contains multiple test cases to completely exercise a given ResTP functionality. All of the tests can be run using the Python program *test.py*. More information about the ns-3 testing framework can be found in the documentation of each release [107].

Some ResTP unit tests include:

- `ResTPHeaderTestSuite`: The test ensures correct header serialization and deserialization, with and without extensions.

- `ResTPExtensionTestSuite`: The test ensures correct serialization and deserialization of each ResTP extension.

- `ResTPTxBufferTestSuite`: The test ensures correct packet (and size) are added to and extracted from the buffer. Correct sequence number assignment is also verified in this test.

- `ResTPArqTestSuite`: The test ensures correct ARQ implementation. The sender correctly generates packet with the right sequence number for transmissions and correctly removes acknowledged data from the transmit buffer after receiving an ACK. The receiver correctly generates ACK packet with the right sequence number after receiving a new packet.

- `ResTPFecTestSuite`: The test ensures correct FEC implementation. The sender correctly encodes an FEC packet, and the receiver correctly decodes an FEC packet and recovers the lost data.

- `ResTPHarqTestSuite`: The test ensures correct HARQ implementation.

- `ResTPOptTestSuite`: The test ensures correct opportunistic connection establishment implementation.

- `MpResTPMappingTestSuite`: The test ensures connection-level sequence numbers map correctly with subflow-level sequence numbers.

A.3 Validation

We validate our ResTP model by using it to simulate same network scenarios as UDP and TCP and compare their performance. The UDP and TCP performance are predictable in these cases.

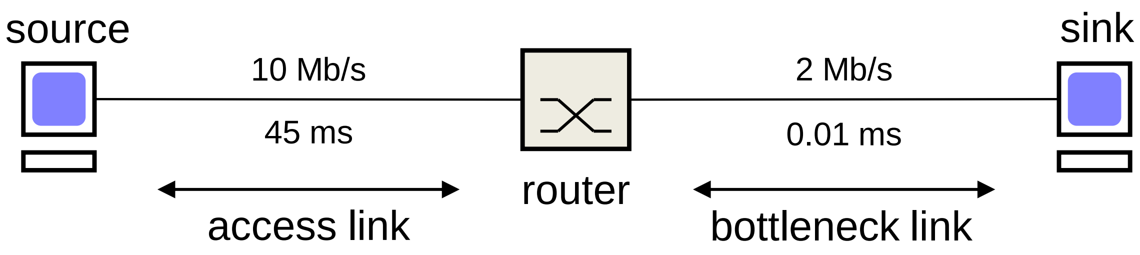

Figure A.2: Single router simulation topology

A.3.1 Simulation Topology

The simulation topology that we use for validating ResTP is illustrated in Figure A.2, which consists a single traffic generator (source) communicating with a receiver (sink) through a router. The access link between the source and the router has bandwidth of 10 Mb/s with a negligible delay of 0.01 ms while the bottleneck link between the router and the sink has a bandwidth of 2 Mb/s and a delay of 45 ms. Packet errors are introduced into the bottleneck link with rates ranging from 0 to 0.5 using `ns3::RateErrorModel`. Specifically, packet error rate (PER) used in our simulations have values of 0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, and 0.5. The router implements a drop-tail queue that has a size of the bandwidth-$\times$-delay product (BDP). All the simulation parameters are summarized in Table A.1 for result reproduction when needed.

Parameter	**Values**
Access link bandwidth	10 Mb/s
Access link propagation delay	0.01 ms
Bottleneck link bandwidth	2 Mb/s
Bottleneck link propagation delay	10 ms – 300 ms
Packet ADU size	1400 B
Error model	Rate Error Model
Packet error rate	[0, 0.5]
Application type	Bulk send (TCP) and CBR (UDP)
CBR rate	1.12 Mb/s
Simulation time	100 s
Queue size	BDP
Queue type	Drop-tail
Congestion control algorithm	NewReno

Table A.1: Simulation parameters

A.3.2 ResTP vs. UDP Using CBR Traffic

We compare ResTP with UDP using CBR traffic. To be compatible with UDP, ResTP is configured to operate in its unreliable mode that provides only the multiplexing and demultiplexing service. The flow, congestion, and error control components are unplugged. The connection management component is plugged in with the connectionless scheme enabled. ResTP utilizes the basic ResTPDU (Figure 3.1) with none of the bits in the **Flags**, **Flow**, **P**, and **Error** header fields enabled. The value in **Cong** is 0.

We simulate CBR traffic using `ns3::OnOffApplication`. The sending application transmits data at a rate of 1.12 Mb/s with an application data unit (ADU) of 1400 bytes during its ON time, which is equivalent to a transmission of $10,000$ packets during our 100-second simulation period. Each simulation for a PER value is repeated 10 times with varying random seed, resulting in a total of 100 runs to obtain a 95% confidence interval.

Figure A.3 plots the throughput of ResTP and UDP as the packet error rate increases. As expected, they achieve very similar performance. Their throughputs keep dropping

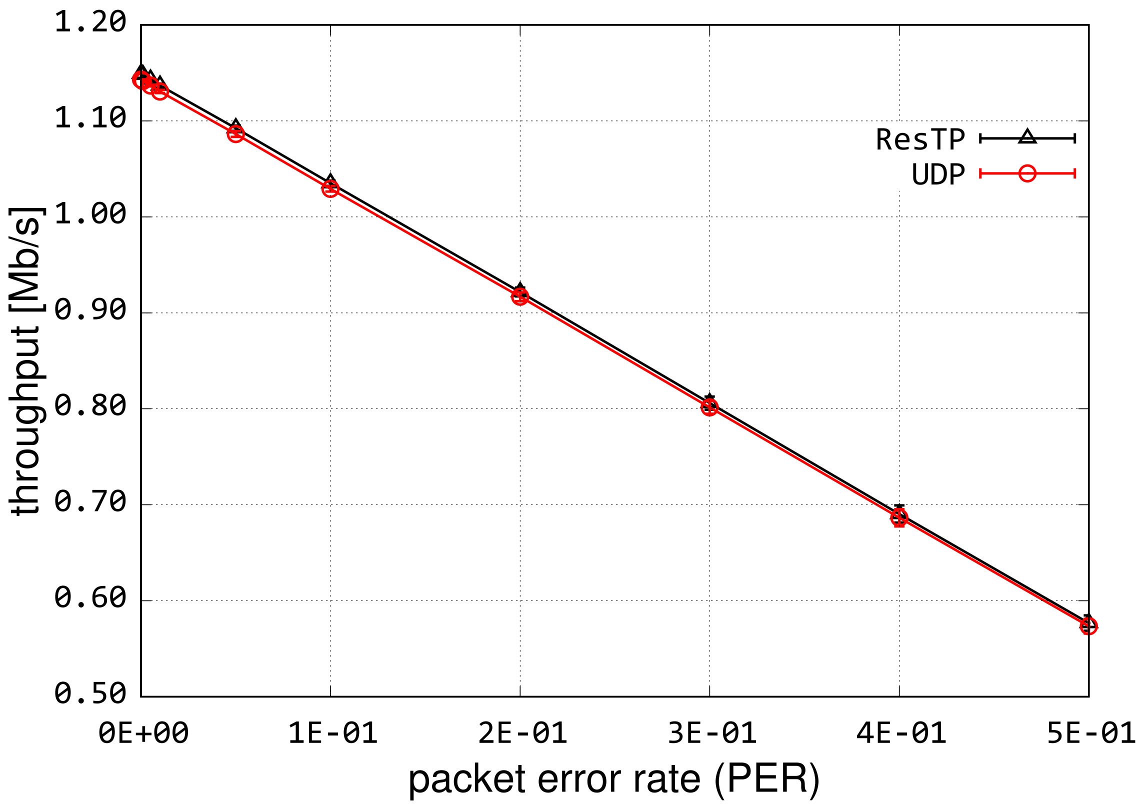

Figure A.3: Throughput of ResTP vs. UDP using CBR traffic over lossy link

with increasing PER given that both UDP and ResTP in its unreliable mode implement no mechanisms to recover from losses.

A.3.3 ResTP vs. TCP Using Bulk Data

The next step in our ResTP validation process is to compare the performance of ResTP with TCP using bulk data transfer. We simulate two different configurations of ResTP. The ResTP-ARQ+NewReno configuration implements the 3-way handshake connection establishment technique, the ARQ with positive ACK error control mechanism, and the NewReno congestion control algorithm as in TCP. The ResTP-ARQ configuration unplugs the congestion control module and only performs the ARQ error correction (similar to Reliable Data Protocol (RDP) [108]). As we mention previously, this capability is impossible for TCP given its rigidly intertwined transfer control. For both of these configurations, ResTP utilizes the basic ResTPDU depicted in Figure 3.1 with the ARQ with positive ACK extension in Figure 3.7. We use large receive buffers for both TCP and ResTP so that it does not impact our results.

We simulate bulk data traffic using `ns3::BulkSendApplication`, in which the sending application generates and transmits data as fast as possible until the application is stopped at the end of our 100-second simulation duration. Both ResTP and TCP have a sending buffer to store the application data until they can be transmitted. Each simulation for a value of PER is replicated 20 times, resulting in a total of 200 simulation runs for each protocol.

Figures A.4 and A.5 show the results obtained when all three protocols (two ResTP configurations and TCP) attempt to transport bulk data across a lossy link. Overall, their throughputs degrade with increasing PER. When PER is low, the ARQ error correction technique can make up the lost packets through retransmissions. When PER is high,

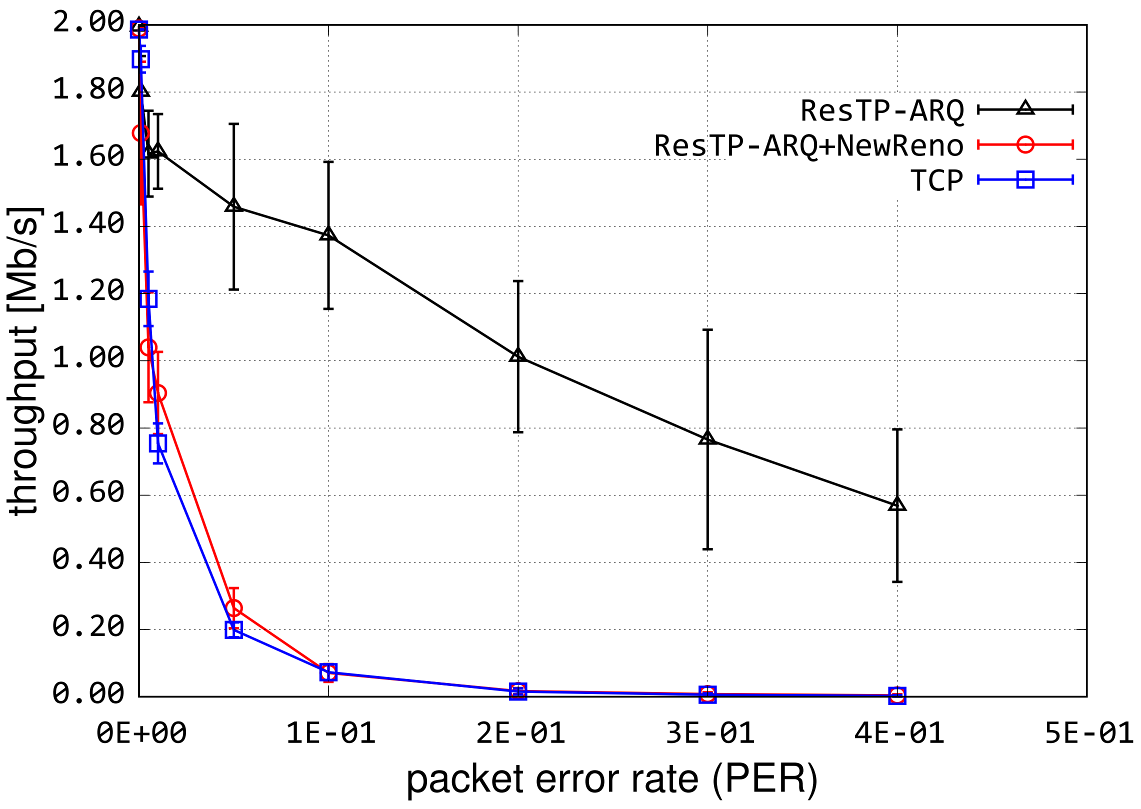

Figure A.4: Throughput of ResTP vs. TCP using bulk send over lossy link

more retransmissions are required and retransmission timeouts (RTOs) also happen more frequently, causing a degradation in their performances.

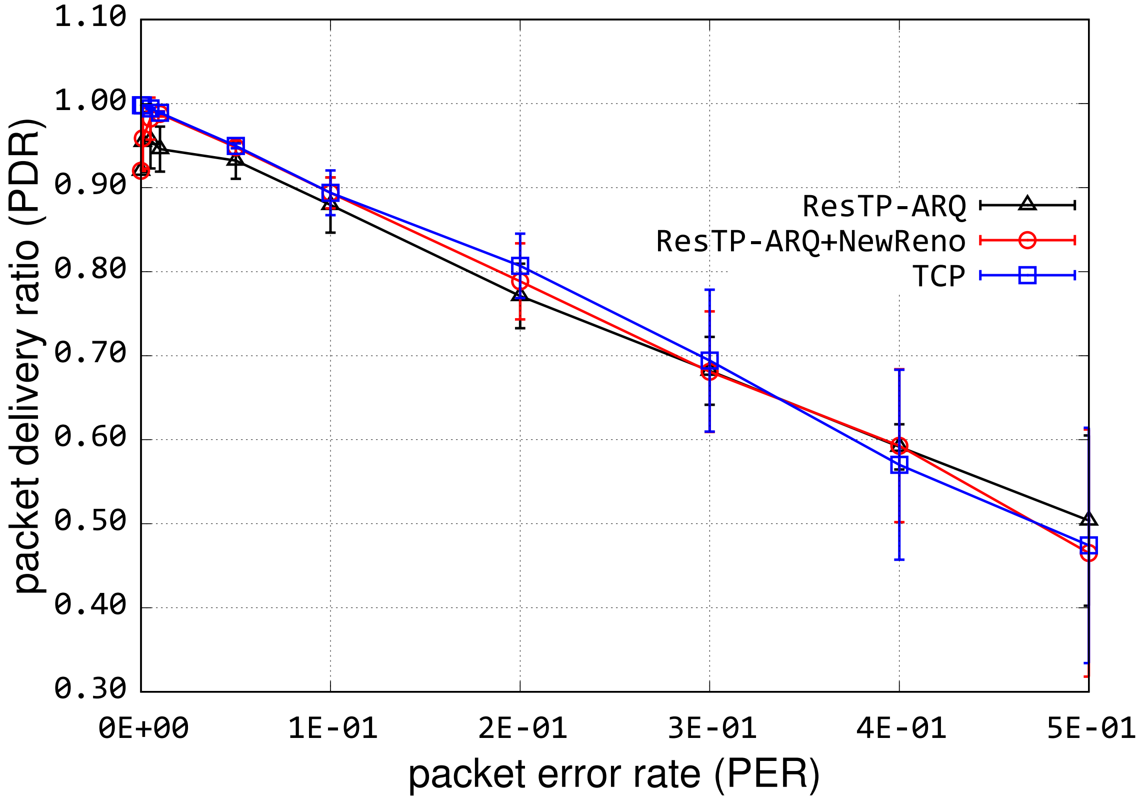

Figure A.5: Throughput of ResTP vs. TCP using bulk send over lossy link

When ResTP is configured with the same algorithms as TCP, we expect a similar performance between the two protocols. This is shown clearly in the throughputs of ResTP-ARQ+NewReno and TCP. ResTP-ARQ is able to achieve higher throughput than both TCP and ResTP-ARQ+NewReno. This is due to the well-known limitation of the NewReno congestion control algorithm when dealing with corruption-based losses. Every time the sender receives three duplicate ACKs due to a loss, NewReno enters its recovery phase and halves its congestion window while the ARQ performs a retransmission. The higher the number of losses, the more often the window is reduced. At the PER value of 0.4 when the majority of losses are triggered by RTOs, and on each RTO, NewReno re-initializes the congestion window to 1, which completely hinders the growth of this window.

We are unable to obtain the average throughput for ResTP-ARQ+NewReno and TCP at PER of 0.5 because at some runs among the total 20 runs, the receiver receives only a single packet, causing the throughput value that is defined as the ratio of number of received bytes and receive duration to be undefined.

A.3.4 ResTP Opportunistic vs. Three-Way Handshake

We also perform a validation of our ResTP opportunistic handshake technique by comparing it with the standard three-way handshake using the same topology as in the previous validations. Both the access and the bottleneck links have the same bandwidth of 5 Mb/s. We vary the bottleneck delay from 10 ms to 300 ms, and trigger a drop of the initial SYN control packet. The traffic generator sends data to the sink as fast as possible until the simulation ends at 100 second using `ns3::BulkSendApplication`. We first simulate ResTP with opportunistic, and then use our protocol for three-way handshake.

Figure A.6 plots the average throughput of both algorithms when the bottleneck delay increases. Overall, the opportunistic approach achieves a higher throughput than the 3-way handshake approach as expected.

We take a closer look at the reason behind the greater performance of the opportunistic mechanism by examining their instantaneous throughput at the 100 ms delay. As shown in Figure A.7, when the initial SYN is dropped, ResTP with the 3-way handshake connection establishment has to wait for 3 seconds to receive a retransmission of the SYN. On the other hand, with the opportunistic approach, the protocol can start its initial setup upon the arrival of the data packet that is transmitted immediately after the initial SYN. Because this packet also carries a SYN flag in its header, the receiver can treat it

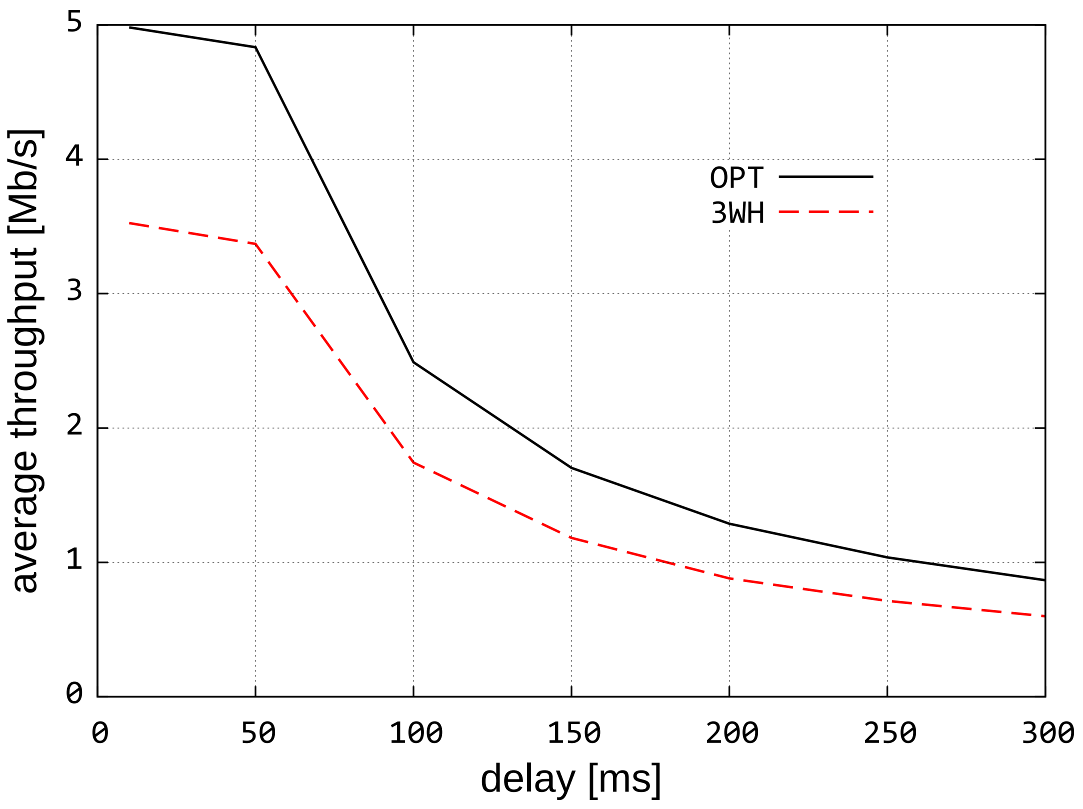

Figure A.6: Average throughput of OPT vs. 3WH with increasing bottleneck delay

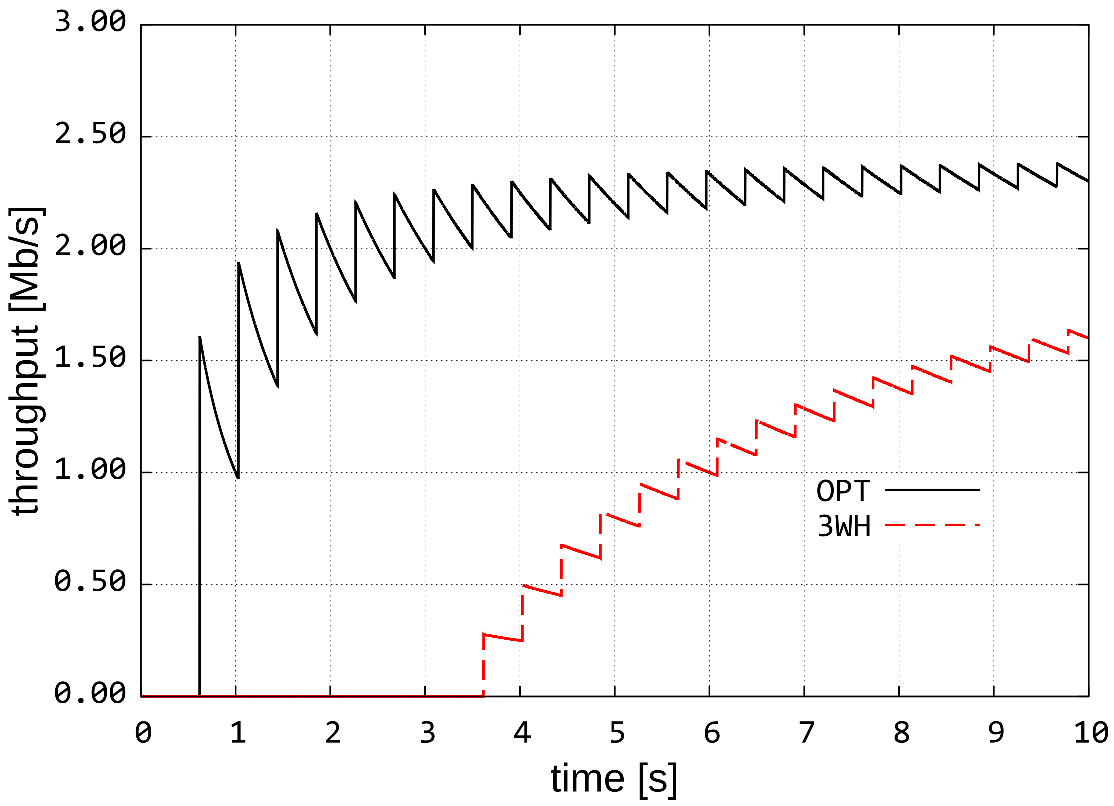

Figure A.7: Instantaneous throughput of OPT and 3WH when SYN dropped

as a connection request and respond with a SYN-ACK, which allows the connection to proceed. This behavior is especially beneficial to short flows that tend to have a strict restriction in their completion time. The benefit is even more significant when these flows have to traverse through long-delay channels.

Bibliography

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

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

[3] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001.

[4] F. Gont and S. Bellovin. Defending against Sequence Number Attacks. RFC 6528(Standards Track), 2012.

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

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

[7] Justin P. Rohrer, Ramya Naidu, and James P. G. Sterbenz. Multipath at the Transport Layer: An End-to-End Resilience Mechanism. In *Proceedings of the IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, pages 1–7, St. Petersburg, Russia, October 2009.

[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, San Diego, CA, October 27–30 2008.

[9] Justin P. Rohrer, Abdul Jabbar, Egemen K. Çetinkaya, Erik Perrins, and James P.G. Sterbenz. Highly-Dynamic Cross-Layered Aeronautical Network Architecture. *Aerospace and Electronic Systems, IEEE Transactions on*, 47(4):2742 –2765, October 2011.

[10] Justin P. Rohrer, Kamakshi Sirisha Pathapati, Truc Anh N. Nguyen, and James P. G. Sterbenz. Opportunistic transport for disrupted airborne networks. In *Proceedings of the IEEE Military Communications Conference (MILCOM)*, pages 737–745, Orland, FL, November 2012.

[11] 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.

[12] Yufei Cheng, Junyan Li, and James P. G. Sterbenz. Path Geo-diversification: Design and Analysis. In *Proceedings of the 5th IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, Almaty, September 2013.

[13] Yufei Cheng, M. Todd Gardner, Junyan Li, Rebecca May, Deep Medhi, and James P.G. Sterbenz. Optimised Heuristics for a Geodiverse Routing Protocol. In *Proceedings of the IEEE 10th International Workshop on the Design of Reliable Communication Networks (DRCN)*, pages 1–9, Ghent, Belgium, April 2014.

[14] Yufei Cheng, M. Todd Gardner, Junyan Li, Rebecca May, Deep Medhi, and James P.G. Sterbenz. Analysing GeoPath Diversity and Improving Routing Performance in Optical Networks. *Computer Networks*, 82:50–67, May 2015.

[15] 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.

[16] Patrick G. Bridges, Gary T. Wong, Matti Hiltunen, Richard D. Schlichting, and Matthew J. Barrick. A Configurable and Extensible Transport Protocol. *IEEE/ACM Trans. Netw.*, 15(6):1254–1265, December 2007.

[17] Matti A Hiltunen, Richard D Schlichting, Xiaonan Han, Melvin M Cardozo, and Rajsekhar Das. Real-time dependable channels: Customizing QoS attributes for distributed systems. *IEEE Transactions on Parallel and Distributed Systems*, 10(6):600–612, 1999.

[18] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824 (Experimental), January 2013.

[19] C. Raiciu, M. Handly, and D. Wischik. Coupled Congestion Control for Multipath Transport Protocols. RFC 6356 (Experimental), 2011.

[20] R. Khalili, T-Labs., N. Gast, M. Popovic, and J.Y. Le Boudec. Mptcp is not pareto-optimal: Performance issues and a possible solution. *Networking, IEEE/ACM Transactions*, 21(5):1651 – 1665, October 2013.

[21] Qiuyu Peng, Anwar Walid, Jaehyun Hwang, and Steven H. Low. Multipath tcp: Analysis, design and implementation. *IEEE/ACM Transactions on networking*, 3(5):6, December 2014.

[22] Yu Cao, Tsinghua Univ, Mingwei Xu, and Xiaoming Fu. Delay-based congestion control for multipath tcp. *Network Protocols (ICNP)*, 20(5):1–10, November 2012.

[23] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo. Impact of Path Characteristics and Scheduling Policies on MPTCP Performance. In *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pages 743–748, May 2014.

[24] ns-3 contributing code. `https://www.nsnam.org/develop/contributing-code/`, September 2018.

[25] 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.

[26] W. A. Doeringer, D. Dykeman, M. Kaiserswerth, B. W. Meister, H. Rudin, and R. Williamson. A survey of light-weight transport protocols for high-speed networks. *IEEE Transactions on Communications*, 38(11):2025–2039, Nov 1990.

[27] Richard W. Watson. The Delta-t Transport Protocol: Features and Experience. In *Local Computer Networks, 1989., Proceedings 14th Conference on*, pages 399–407, October 1989.

[28] D Cheriton and D Cheriton. VMTP: A transport protocol for the next generation of communication systems. *ACM SIGCOMM Computer Communication Review (CCR)*, 16(3):406–415, September 1986.

[29] A. Baratz, J. Gray, P. Green, J. Jaffe, and D. Pozefsky. Sna networks of small systems. *IEEE Journal on Selected Areas in Communications*, 3(3):416–426, May 1985.

[30] James Martin, Kathleen Kavanagh Chapman, et al. *SNA: IBM's networking solution.* Prentice-Hall, Inc., 1987.

[31] Thomas J Routt. Distributed SNA: A network architecture gets on track. In *Systems network architecture*, pages 167–180. IEEE Press, 1992.

[32] RJ Sundstrom, JB Staton, GD Schultz, ML Hess, and GA Deaton. SNA directionsa 1985 perspective. In *AFIPS Conference Proceedings; vol. 55 1986 National Computer Conference*, pages 537–551. AFIPS Press, 1986.

[33] GL Chesson. Datakit software architecture. *Proc. ICC, pages*, 20:1–20, 1979.

[34] A. Fraser. Towards a universal data transport system. *IEEE Journal on Selected Areas in Communications (JSAC)*, 1983.

[35] A. G. Fraser and W. T. Marshall. Data transport in a byte stream network. *IEEE Journal on Selected Areas in Communications*, 7(7):1020–1033, Sep 1989.

[36] D.D. Clark, M.L. Lambert, and L. Zhang. NETBLT: A bulk data transfer protocol. RFC 998 (Experimental), March 1987.

[37] U. Maheshwari. HULA: An Efficient Protocol for Reliable Delivery of Messages. Technical report, Cambridge, MA, USA, 1997.

[38] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323 (Proposed Standard), May 1992.

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

[40] Carl A Sunshine and Yogen K Datal. Connection management in transport protocols. *Computer Networks (1976)*, 2(6):454–473, 1978.

[41]

[42] Y. Cheng, J. Chu, S. Radhakrishnan, and A. Jain. TCP Fast Open. RFC 7413 (Experimental), December 2014.

[43] James F Kurose and Keith W Ross. *Computer networking: a top-down approach*. Pearson Education, Inc, 2017.

[44] Van Jacobson. Congestion Avoidance and Control. In *Symposium proceedings on Communications architectures and protocols*, SIGCOMM '88, pages 314–329, New York, NY, USA, 1988. ACM.

[45] Cui-Qing Yang and A. V. S. Reddy. A taxonomy for congestion control algorithms in packet switching networks. *IEEE Network*, 9(4):34–45, July 1995.

[46] Michael Welzl. *Network congestion control: managing internet traffic*. John Wiley & Sons, 2005.

[47] M. Hock, R. Bless, and M. Zitterbart. Experimental evaluation of BBR congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10, Oct 2017.

[48] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):50:20–50:53, October 2016.

[49] Jim Gettys and Kathleen Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11):40:40–40:54, November 2011.

[50] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582 (Standards Track), 2012.

[51] 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.

[52] A. Kuzmanovic and E. W. Knightly. TCP-LP: a distributed algorithm for low priority data transfer. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428)*, volume 3, pages 1691–1701 vol.3, March 2003.

[53] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-Speed and Long Distance Networks. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12, April 2006.

[54] 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.

[55] S. Mascolo, C. Casetti, M. Gerla, M.Y. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297. ACM, 2001.

[56] Cheng Peng Fu and S. C. Liew. TCP Veno: TCP enhancement for transmission over wireless access networks. *IEEE Journal on Selected Areas in Communications*, 21(2):216–228, Feb 2003.

[57] Carlo Caini and Rosario Firrincieli. Tcp hybla: a tcp enhancement for heterogeneous networks. *International journal of satellite communications and networking*, 22(5):547–566, 2004.

[58] S. Floyd. HighSpeed TCP for Large Congestion Windows. RFC 3649 (Experimental), December 2003.

[59] Tom Kelly. Scalable TCP: improving performance in highspeed wide area networks. *SIGCOMM Comput. Commun. Rev.*, 33(2):83–91, April 2003.

[60] Lisong Xu, Khaled Harfoush, and Injong Rhee. Binary increase congestion control (BIC) for fast long-distance networks. In *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2514–2524, 2004.

[61] Andrea Baiocchi, Angelo P Castellani, and Francesco Vacirca. YeAH-TCP: yet another highspeed TCP. In *Proc. PFLDnet*, volume 7, pages 37–42, 2007.

[62] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: a new TCP-friendly high-speed TCP variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008.

[63] Haitao Wu, Zhenqian Feng, Chuanxiong Guo, and Yongguang Zhang. ICTCP: Incast Congestion Control for TCP in Data-center Networks. *IEEE/ACM Trans. Netw.*, 21(2):345–358, April 2013.

[64] L. C. Kho, X. Dfago, A. O. Lim, and Y. Tan. A taxonomy of congestion control techniques for tcp in wired and wireless networks. In *2013 IEEE Symposium on Wireless Technology Applications (ISWTA)*, pages 147–152, Sept 2013.

[65] Vassilios Tsaoussidis and Chi Zhang. TCP-Real: receiver-oriented congestion control. *Computer Networks*, 40(4):477–497, 2002.

[66] Feng Wang and Yongguang Zhang. Improving TCP performance over mobile adhoc networks with out-of-order detection and response. In *Proceedings of the ACM*

International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), pages 217–225, Lausanne, Switzerland, 2002. ACM Press.

[67] S. Floyd, M. Handley, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 5348 (Proposed Standard), 2008.

[68] Y. Shu, W. Ge, N. Jiang, Y. Kang, and J. Luo. Mobile-Host-Centric Transport Protocol for lt;newline/ gt;EAST Experiment. *IEEE Transactions on Nuclear Science*, 55(1):209–216, Feb 2008.

[69] Damon Wischik, Mark Handley, and Marcelo Bagnulo Braun. The Resource Pooling Principle. *SIGCOMM Comput. Commun. Rev.*, 38(5):47–52, September 2008.

[70] Qiuyu Peng, Anwar Walid, and Steven H. Low. Multipath TCP Algorithms: Theory and Design. *SIGMETRICS Perform. Eval. Rev.*, 41(1):305–316, June 2013.

[71] Michio Honda, Yoshifumi Nishida, Lars Eggert, Pasi Sarolahti, and Hideyuki Tokuda. Multipath congestion control for shared bottleneck. In *Proc. PFLDNeT workshop*, volume 357, page 378. Citeseer, 2009.

[72] Frank Kelly and Thomas Voice. Stability of End-to-end Algorithms for Joint Routing and Rate Control. *SIGCOMM Comput. Commun. Rev.*, 35(2):5–12, April 2005.

[73] Huaizhong Han, Srinivas Shakkottai, C. V. Hollot, R. Srikant, and Don Towsley. Multi-path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity in the Internet. *IEEE/ACM Trans. Netw.*, 14(6):1260–1271, December 2006.

[74] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In

Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation, NSDI'11, pages 99–112, Berkeley, CA, USA, 2011. USENIX Association.

[75] Christoph Paasch, Simone Ferlin, Ozgu Alay, and Olivier Bonaventure. Experimental evaluation of multipath tcp schedulers. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, pages 27–32. ACM, 2014.

[76] F. Yang, P. Amer, and N. Ekiz. A Scheduler for Multipath TCP. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–7, July 2013.

[77] Robert C Durst, Patrick D Feighery, and Keith L Scott. Why not use the standard internet suite for the interplanetary internet, 2000.

[78] Bruce Elbert. *Introduction to satellite communication*. Artech House, 2008.

[79] Ian F. Akyildiz, Giacomo Morabito, and Sergio Palazzo. TCP-Peach: A New Congestion Control Scheme for Satellite IP Networks. *IEEE/ACM Trans. Netw.*, 9(3):307–321, June 2001.

[80] 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.

[81] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), 2003.

[82] Sam Liang and David Cheriton. TCP-RTM: Using TCP for real time multimedia applications", booktitle = in International Conference on Network Protocols, year = 2002.

[83] Lars-Ake Larzon, Mikael Degermark, and Stephen Pink. *UDP lite for real time multimedia applications*. Citeseer, 1999.

[84] PP-K Lam and Soung C Liew. UDP-Liter: an improved UDP protocol for real-time multimedia applications over wireless links. In *1st International Symposium onWireless Communication Systems, 2004.*, pages 314–318. IEEE, 2004.

[85] Robert C. Durst, Gregory J. Miller, and Eric J. Travis. TCP extensions for space communications. *Wireless Networks*, 3(5):389–403, October 1997.

[86] CCSDS-The Consultative Committee for Space Data Systems. Space Communications Protocol Specification (SCPS)-Transport Protocol (SCPS-TP). `http://public.ccsds.org/publications/archive/714x0b2.pdf`, October 2006.

[87] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. Reducing Web Latency: The Virtue of Gentle Aggression. *SIGCOMM Comput. Commun. Rev.*, 43(4):159–170, August 2013.

[88] T. Flach, N. Dukkipati, and B. Raghavan. TCP Instant Recovery: Incorporating Forward Error Correction in TCP. Internet-Draft, 2013.

[89] Venkata Yedugundla, Simone Ferlin, Thomas Dreibholz, Ozgu Alay, Nicolas Kuhn, Per Hurtig, and Anna Brunstrom. Is Multi-Path Transport Suitable for Latency Sensitive Traffic? *COMNET*, 105, 05 2016.

[90] M. Scharf and A. Ford. Multipath TCP (MPTCP) Application Interface Considerations. RFC 6897 (Informational), 2013.

[91] T. A. N. Nguyen and J. P. G. Sterbenz. Connection Management in a Resilient Transport Protocol. In *DRCN 2017 - Design of Reliable Communication Networks; 13th International Conference*, pages 1–8, 2017.

[92] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, Jeff Bailey, Jeremy Dorfman, Jim Roskind, Joanna Kulik, Patrik Westin, Raman Tenneti, Robbie Shade, Ryan Hamilton, Victor Vasiliev, Wan-Teh Chang, and Zhongyi Shi. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '17, pages 183–196, New York, NY, USA, 2017. ACM.

[93] Dragana Damjanovic, Philipp Gschwandtner, and Michael Welzl. Why is this web page coming up so slow? investigating the loss of SYN packets. In *International Conference on Research in Networking*, pages 895–906. Springer, 2009.

[94] 3gpp http applications. `http://www.nsnam.org`, July 2009.

[95] Satellite network simulator 3 (sns3). `https://www.sns3.org`.

[96] Satellite network simulator 3 (sns3) implementation. `https://github.com/sns3/sns3-satellite`.

[97] LBT Wi-Fi Coexistence Module Documentation. `https://www.nsnam.org/~tomh/ns-3-lbt-documents/html/lbt-wifi-coexistence.html`.

[98] David Gómez Fernández. Multipath tcp in ns-3. `https://github.com/dgomezunican/multipath-ns3.13`, 2013.

[99] Yufei Cheng, Md. Moshfequr Rahman, Truc Anh N. Nguyen, Siddharth Gangadhar, Mohammed J.F. Alenazi, and James P.G. Sterbenz. Cross-layer geodiverse protocol stack for resilient multipath transport and routing using openflow. In *12th International Conference on Design of Reliable Communication Networks*, 2015.

[100] Justin P. Rohrer, Mohammed J. F. Alenazi, and James P. G. Sterbenz. ResiliNets Topology Map Viewer. `http://www.ittc.ku.edu/resilinets/maps/`, January 2011.

[101] J. W. Suurballe. Disjoint Paths in a Network. *Networks*, 4(2):125–145, 1974.

[102] J. W. Suurballe and R. E. Tarjan. A Quick Method for Finding Shortest Pairs of Disjoint Paths. *Networks*, 14(2):325–336, 1984.

[103] Alexandros Kostopoulos, Henna Warma, Tapio Levä, Bernd Heinrich, Alan Ford, and Lars Eggert. Towards multipath tcp adoption: Challenges and opportunities. In *NGI*, pages 1–8. Citeseer, 2010.

[104] Maurizio Casoni and Natale Patriciello. Next-generation TCP for ns-3 simulator. *Simulation Modelling Practice and Theory*, 66:81–93, 2016.

[105] MPTCP Implementation for ns-3 (Code Review). `https://codereview.appspot.com/369810043/`, 2019.

[106] Kashif Nadeem and Tariq M. Jadoon. An ns-3 mptcp implementation. In Trung Q. Duong, Nguyen-Son Vo, and Van Ca Phan, editors, *Quality, Reliability, Security and Robustness in Heterogeneous Systems*, pages 48–60, Cham, 2019. Springer International Publishing.

[107] ns-3 testing framework. `https://www.nsnam.org/docs/release/3.29/manual/html/test-framework.html`, September 2018.

[108] D. Velten, R.M. Hinden, and J. Sax. Reliable Data Protocol. RFC 908 (Experimental), July 1984. Updated by RFC 1151.