QUIC working group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

A. Joseph
T. Li
UCLA
Z. He
Y. Cui
Tsinghua University
L. Zhang
UCLA

March 5, 2018

A Comparison between SCTP and QUIC
draft-joseph-quic-comparison-quic-sctp-00

Abstract

To cumulate design lessons from our protocol development efforts, this document provides a preliminary comparison between two transport protocol designs, Stream Control Transport Protocol (SCTP) and Quick UDP Internet Connections (QUIC). We identify their commonalities and differences, summarize the characteristics of QUIC which we believe represent progresses in transport protocol designs. We hope this draft useful in helping others to gain further understanding of both SCTP and QUIC, and in future protocol design efforts.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2018.

Table of Contents

1. Introduction

Quick UDP Internet Connections (QUIC) builds upon the design lessons learned from many existing protocols. The purpose of this draft is to draw parallels and display differences between the two transport protocols, Stream Control Transport Protocol (SCTP) and Quick UDP Internet Connections (QUIC), with a hope to gain deeper insight from the comparison, extract general lessons, and trends.

As such, this draft is not intended to be a deep dive into the inner working details of the protocols, but rather a high level view of similar core functionality and mechanisms of the protocols. These two protocols were developed years apart and for very different purposes, however as transport protocols they share a number of similarities. However it should be noted that at the time of this writing, the QUIC specification is still under active development. This means that parts of the specifications are still incomplete or missing at this time. This draft focuses on what has been documented so far; the reader should keep in mind that the QUIC protocol might have changed after the time of this draft's publication.

2. Background

The SCTP protocol was originally developed in year 2000 to transport Public Switched Telephone Network (PTSN) signaling messages over IP; it was later adapted to be a general purpose transport protocol [RFC4960]. The motivation of developing a new transport protocol came from perceived drawbacks of using TCP for transmitting PTSN messages: HOL-Blocking, lack of host multi-homing support, mismatch between TCP's byte-stream data model and PTSN applications's message-oriented communication, and TCP's vulnerability to SYN attacks. SCTP uses multiple substreams to mitigate HOL blocking, enables each transport connection to utilize multiple interfaces, and reliably delivers application messages instead of byte streams.

The development of the QUIC protocol was started by Jim Roskind's team at Google in 2012, aiming to remove identified performance bottlenecks in transport protocols. As Internet bandwidths continue to increase due to technology advances and infrastructure buildout, the Round Trip Time (RTT) became a physical upper bound of the speed of light. The existing transport protocols take multiple RTTs to deliver a web page's contents [QUIC-DESIGN]. Using multiple TCP connections to improve performance has its own limitations: it forces client applications to bind to many different sockets to send out multiple separate requests, resulting in redundant connection setup and bandwidth wastage as well as inefficient allocation of computer resources. QUIC developed an innovative design for connection setup that integrates transport protocol and TLS functions to minimize RTT.

Similar to STCP, QUIC developed support for multiple substreams instead of using multiple transport connections[QUIC-DESIGN]. QUIC's predecessor was another transport protocol from Google called SPDY, which ran over a single TCP connection and routinely with SSL [QUIC-DESIGN]. The lessons learned from SPDY drove many of the design decisions for QUIC, including the decision to run over UDP instead of TCP to avoid TCP's HOL-Blocking, an innovative congestion control scheme, and considerations for mobile devices in connection teardown [QUIC-DESIGN].

3. A High Level Overview of the Two Protocols

3.1. SCTP

A SCTP connection is comprised of an association between two endpoints, each is defined by a set of IP addresses and a port number. A SCTP connection is referred to as an association so the rest of this draft will use this term. While a primary IP address is used for each endpoint, each end may inform the other end a set of addresses it may use to transmit packets. Moving away from TCP's approach of one-header-fitting-all, STCP designed multiple separate data structures called "chunks" to carry association control information and applications data messages. SCTP communicates with an Upper Layer Protocol (ULP) through the use of message primitives ASSOCIATE and SHUTDOWN. These primitives are how applications are able to communicate with SCTP to setup and teardown association.

SCTP supports multiple message substreams by letting each of the two endpoints negotiate with the other on the number of substreams they can support at the association setup time, and ensures in-order delivery of messages in substream to the ULP through the use of a substream sequence number. To provide reliable message delivery for all substreams, SCTP assigns each data chunk a unique Transmission Sequence Number (TSN). Note that the TSN is on per association basis, not per substream. It works in an identical way to TCP sequence number in ensuring reliable delivery, except that the former names a data chunk while the latter a data byte. A data may contain either a data message, or a segment of a data message. SCTP uses the same congestion avoidance and control mechanisms as TCP, and similar selectively acknowledgement scheme, except that it designed a dedicated SACK chunk, as opposed to TCP's use of its option field for SACK.

3.2. QUIC

A QUIC connection is comprised of an association between two endpoints defined by a pair of IP address and port number (at the time of this writing, QUIC's multihoming/multipath support is still

under development). The IP addresses and ports can change in the middle of a connection. A fundamental difference between QUIC and TCP or SCTP is that QUIC is a user space transport protocol, which allows rapid protocol revision without having to wait for system upgrades. To support rapid protocol revision, QUIC's connection setup goes through a negotiation process that involves determining the lowest common version supported between the two endpoints and a cryptographic handshake which incorporates TLS to provide a secure connection.

Within a QUIC connection, substreams can be started at any time, excluding tear down phase. Either endpoint can start a substream, which can be either bidirectional or unidirectional. QUIC inherits TCP's byte stream data model. Dedicated control structures called "frames" are used to communicate with and carry byte data to endpoints.

Similar to SCTP, QUIC has a dedicated SACK frame to carry selective acknowledgement, although the semantics of QUIC SACK differs from that of SCTP in important ways. SACK informs which packets are delivered to the other end; un-ACKed packets are considered lost. No QUIC packet is ever retransmitted, packet numbers always increases monotonically. From each received SACK frame, a QUIC endpoint can infer which byte frames have been received by the other end. To ensure reliable in-order data delivery of each byte stream to the application, the sender will retransmit the byte frames that are not acknowledged. The new frames may repackage the missing byte offsets.

As another difference from SCTP, QUIC practices flow control both on a connection basis and on per substream basis, by advertising the max amount of data allowed on a connection, as well as per stream. If an endpoint transmits more than advertised, the entire connection is torn down.

4. A Comparative Examination of Specific Protocol Mechanisms

4.1. Packet Structure

The packet structures of both SCTP and QUIC break away from TCP's one-header-fits-all design. Instead, they used dedicated control chunks for connections setup, teardown and SACK.

4.1.1. SCTP

The SCTP [RFC4960] packet structure contains a common header (Figure 1) with attached DATA chunks (Figure 2) which is analogous to QUIC's Short Header (Figure 3) and STREAM frames (Figure 4). The common header contains fields for the set of source and destination

IP addresses and ports, verification tag, and a checksum of the packet. The DATA chunk has a type and reserved fields of 0. The U bit set to 1 indicates that the data is unordered and the value in the Stream Sequence Number (SSN) can be ignored. The SSN indicates what number message the chunk contains for the related Stream, and also determines the order the messages will be delivered to the ULP (unless it is meant to be unordered). SSN always starts from 0 and increments up to 65535 with wrap around. The Transmission Sequence Number (TSN) enumerates the Chunks attached to the common header and increment sequentially with wrap around over the lifetime of an association. TSNs range from 0 to 4294967295, and can start at a random value in the range. TSNs are repeated during retransmission of packets to ensure reliable delivery. The Length field indicates the length of the DATA Chunk including the 16 bytes of the fields starting from the Type field. The Stream identifier is the number identifying the stream the chunk belongs to. The Payload Protocol Identifier is not relevant for the purposes of this paper and is not used by the SCTP protocol itself, but is intended for use by middleboxes. The User Data field contains user data which is padded at the end to a 4 byte boundary of all-zero bytes.

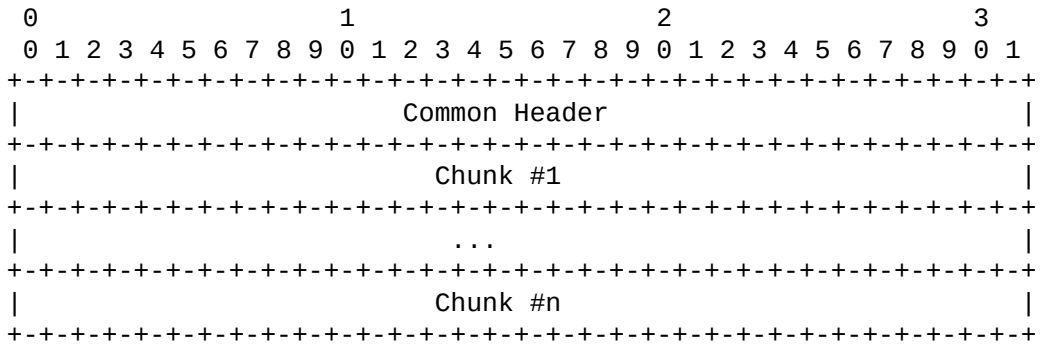

Figure 1: SCTP Packet Structure

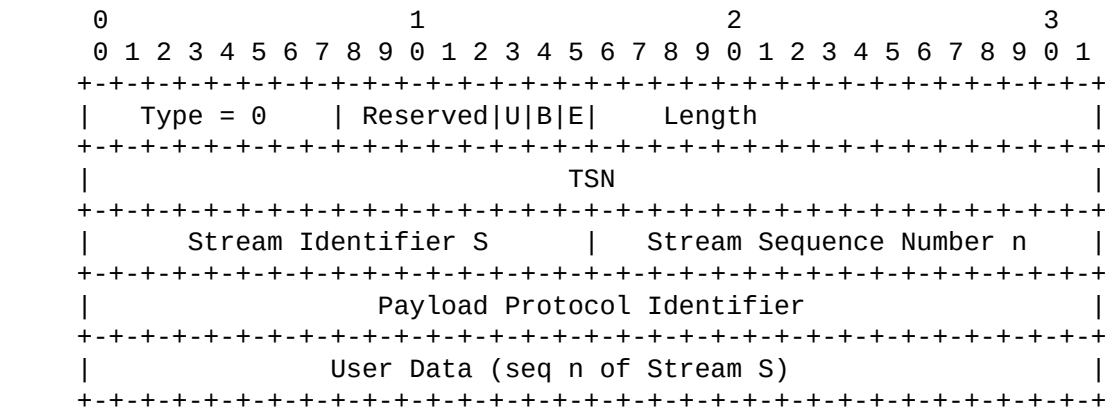

Figure 2: SCTP DATA Chunk

4.1.2. QUIC

The QUIC packet structure consists of a common header called a short header (Figure 3) and attached Frames in the protected payload (Figure 4). A user data payload bearing packet sent after connection is set up is called a STREAM frames (Figure 4), analogous to SCTP's DATA Chunks (Figure 2), and is the primary frame used for data transfer [QUIC-TRANSPORT]. The Type field indicates the type of Frame it is: the range 0x10-0x17 indicates a STREAM Frame. The lower three bites of the Type field also encode whether certain fields are in the frame. 0x04 is the OFF bit, if set to 1, there is an offset field, if set to 0, the frame starts from byte offset of 0 or there is no data. 0x02 is the Length bit, if set to 1, there is a length field, if set to 0, the length of the data extends to the end of the packet. Finally the 0x01 is the FIN bit, if set to 1, it is the final frame in a stream [QUIC-TRANSPORT]. The Stream ID identifies the stream the frame belongs to, as well as if the stream is bidirectional or unidirectional and if the server or the client created the stream. If the second least significant bit is 1, the stream is unidirectional, if it is 0, the stream is bidirectional. If the least significant bit is 1, the server initiated the stream, if it is 0, the client initiated the stream. The offset field indicates the byte offset that the STREAM Frame is carrying, and the length field indicates the length of the stream data. There can be multiple STREAM Frames per QUIC packet/header. [QUIC-TRANSPORT]

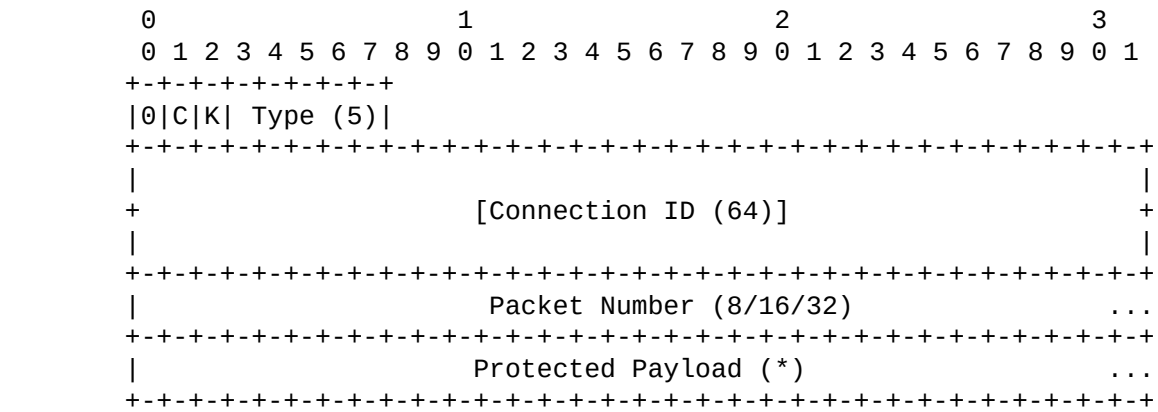

Figure 3: Short Header

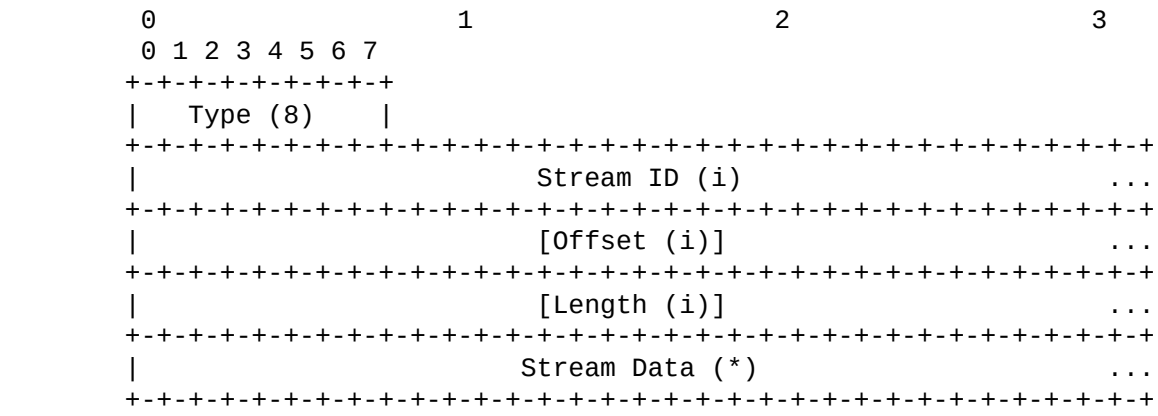

Figure 4: STREAM Frame

4.2. Connection Setup

For connection setup, SCTP adopts the 4-way handshake with digitally signed state cookie for preventing denial-of-service attacks (SYN-flooding). The state cookie is sent by the server in response to the client's INIT message, and contains all of the state that the server needs to ensure that the association is valid, including Message Authentication Code (MAC) [RF2104], a timestamp, and the cookie lifespan. The cookie contains all the information needed for SCTP association setup, so the server's SCTP stack does not need to keep information about the associating client.

For connection setup, QUIC directly incorporates TLS key negotiation process with the transport handshake, establishing secure connection using 1-RTT with successful version negotiation, and 0-RTT for connection resumptions. During initial connection setup, the server

gives the client a cryptographic cookie known as Source Address Token (client IP and timestamp) for source address validation. It also sends the Server Config containing the server's long-term Diffie-Hellman public value and server preference. These information can be used for subsequent connections. This provides a secure and efficient way for establishing connections, yet unlike traditional syn-cookies for preventing syn-flood attack which are designed for single use, QUIC's longterm cookies might bring potentials for new types of attacks (e.g. replay attack). QUIC server also adopts stateless address validation, the cookie stores all state necessary to continue the connection.

4.2.1. SCTP

While SCTP is similar to TCP where a connection is defined by a pair IP addresses and port numbers, SCTP is slightly different by defining a set of possible IP addresses and port numbers in its common header. This is to facilitate SCTP's multihoming features, since messages can be sent or received at any of these addresses, even though there is a primary address specified. A normal SCTP association begins when an endpoint "A" sends an INIT chunk to the other endpoint. The INIT chunk (Figure 5) will contain a Verification Tag value which is a random number between the range of 1 and 4294967295. The Verification Tag can be used as the initial TSN as well. The other endpoint "Z" will respond with an INIT ACK chunk containing its own Verification Tag as well as as generating and sending a State Cookie back. Endpoint "A" will then respond with a COOKIE ECHO chunk which might contain DATA chunks as well. Endpoint "Z" will acknowledged the COOKIE ECHO with a COOKIE ACK chunk, which can also be bundled with other DATA chunks. The ULP of each of the endpoints will then be notified that a successful association has been established. Within the INIT and INIT ACK chunks that were sent by each endpoint, the number of outbound and inbound streams accepted by each endpoint were communicated. The endpoints will take the minimum of each of their preferred outbound streams and the minimum inbound stream of the other endpoint, minus 1: min(local outbound stream, remote minimum inbound streamC) - 1. All SCTP substreams are unidirectional. The State Cookie that is sent out during connection setup contains a Message Authentication Code (MAC) [RF2104], a timestamp and the lifespan for the cookie. The entire SCTP association setup results in a 4-way handshake in order to avoid a SYN-flood situation. Once a connection is set up, it is possible that SCTP will fragment its chunks in order to avoid IP fragmentation. Fragmentation is done by a source host only and the peer endpoint will reassemble once it is received.

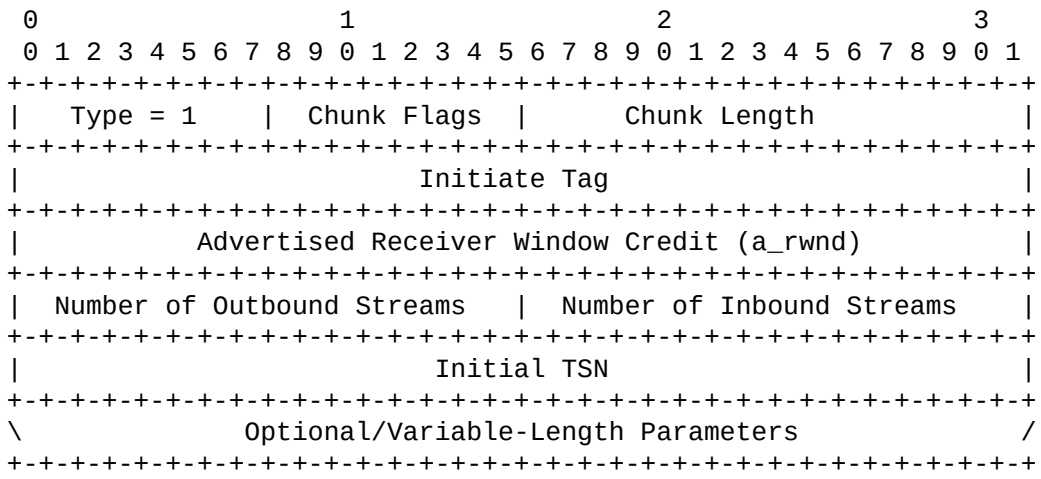

Figure 5: SCTP INIT Chunk

4.2.2. QUIC

A normal QUIC connection begins with version negotiation between two endpoints. An Initial packet with a long header is sent out by a client to determine if both endpoints support the same version of QUIC. The Version Negotiation packet (Figure 6) is sent by the server if the client that sent out the initial packet is attempting to create a new connection and the client's version is not accepted by the server. If the Initial packet's version is supported by the server or the client responds to the server's Version Negotiation packet with a supported version, the handshake process continues. After a version is settled on by both endpoints, the transport and cryptographic handshake begins. Stream 0, is a reserved substream that is used for the cryptographic handshake process. The current version of QUIC uses TLS 1.3 to encrypt the connection and authenticate the server and optionally authenticate the client. QUIC is able to reduce handshake delay caused by RTT by combining the transport and cryptographic handshake together to provide a secure connection from the start of a connection. QUIC embeds the TLS functionality within the protocol itself, without having to run a separate TLS handshake and session after the transport handshake. During the cryptographic handshake, each endpoint advertises transport parameters that define the initial parameters for the connection. These transport parameters includes values that determine the maximum amount of data that can be transmitted per stream, as well as per connection data maximums. These values are updated during the lifetime of a connection to facilitate flow control. Once a connection is established, substreams can be created during any point of the connection lifetime. QUIC also supports both

unidirectional and bidirectional substreams which is determined during sub stream setup.

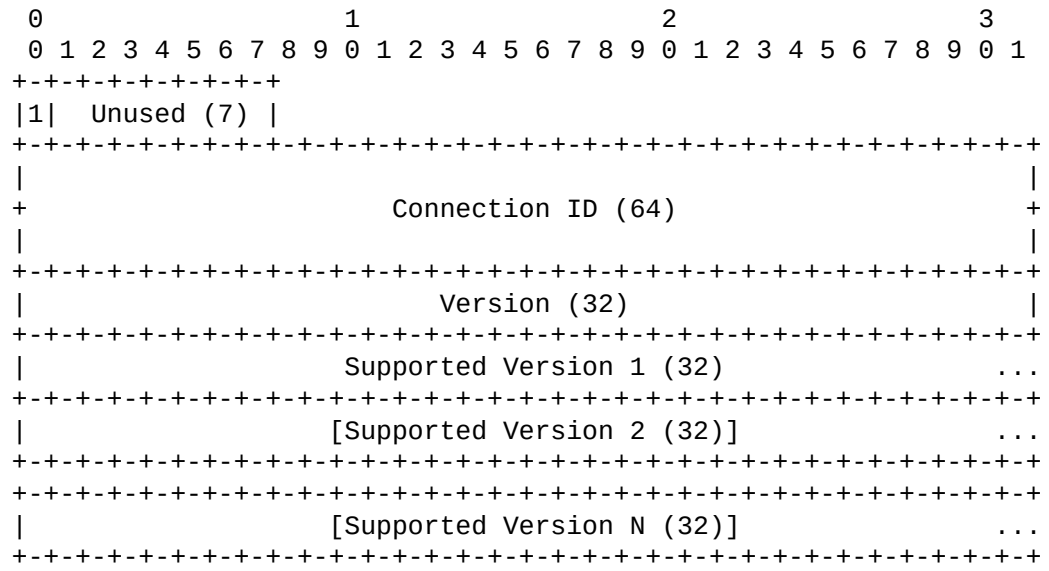

Figure 6: QUIC Version Negotiation Packet

4.3. Substreams

One of the major features of both of these protocols are multiplexed streams. An issue with TCP is that a dropped packet can block message delivery for all application-level streams since TCP uses a single-byte stream abstraction. This blockage is called Head-of-Line Blocking. SCTP and QUIC solve for this by supporting multiplex streams within the protocol itself. If a dropped packet occurs for either protocol, in-order messages/byte streams can still be delivered for other streams if they are available.

SCTP intended to get rid of HOL-Blocking by substreams, but its Transmission Sequence Number (TSN) couples together the transmission of all data chunks. For SCTP, each packet contains different data chunks from different streams identified by Stream ID (SID), if the data chunk of one streams is lost, the data of other streams should still be received by the application. However, a TSN is assigned to every data chunk in the association. For SCTP Cumulative ACK, the value of the Cumulative TSN ACK parameter is the last TSN received before a break in the sequence of received TSNs. The next TSN value following this one has not yet been received at the endpoint sending the SACK. This parameter therefore acknowledges receipt of all TSNs less than or equal to its value. As a result, in SCTP if a packet is

lost, all the packets with TSN after this lost packet cannot be received until it is retransmitted.

QUIC adopts two levels of numbering. User data is uniquely identified by stream ID and offset, it also has a monotonically increasing packet sequence number.The packet sequence number is used for congestion control and loss detection and it numbers all the packets (SCTP don't number control packets). QUIC selective ack, acknowledges packet sequence number of the last received packet, and QUIC retransmits the lost packet using a new sequence number. As a result, the congestion window could open up for more packets, and the lost packet does not affect the packets following it from being received, thus avoiding the HOL-Blocking problem. However, as QUIC SACK tells which packets get lost but does not retransmit the lost packet, QUIC has to keep internal mapping of which stream frame in which packet, to know which one needs to be retransmitted, which introduces additional processing.

4.3.1. SCTP

SCTP substream setup requires the number of substreams as well as their Stream IDs be declared at association setup. SCTP does not have the functionality to start streams during a association since it does not differentiate between client-initiated and server-initiated streams. Additionally, streams persist through the lifetime of an active association. SCTP's INIT Chunk declares two fields, Number of Outbound Streams (OS) and Number of Inbound Streams (NIS), to help negotiate the number of streams to be created during an association. The number of streams is not negotiated in the traditional sense, but instead the minimum of the requested streams and offered streams is taken. For example, if a receiver advertises 6 streams in its MIS field, and a sender advertises 12 streams in its OS field, the number of streams to the receiver will be 6. If either of these fields is set to 0, the association will be aborted. If a sender is limited to less streams than was requested, it can communicate to its application layer it failed to secure the number of streams that was required, and the application can decide to continue the association or abort it. SCTP substreams are only unidirectional and each stream's sequence number must start from 0. Since streams are created during association setup, if the number of streams needs to be changed, the association needs to be torn down and re-setup.

4.3.2. QUIC

QUIC has the functionality to start or teardown a substream during any point of the connection (aside from connection teardown). The parity of the Stream ID allows QUIC to spawn new substreams on either the client or server side without the need to undergo negotiations

between each side to decide on an ID. Since parity is decided by the least significant bit, a client only picks even Stream IDs, and the server only picks odd Stream IDs. QUIC substreams support unidirectional and bidirectional streams which is determined by setting the second to least significant bit in the Stream ID. The bit reservations allow for streams to be started at anytime during the lifetime of a connection without the need for negotiation. Each endpoint is restricted to their own non-overlapping range of IDs, thus canceling out the need to negotiate for an ID in order to avoid conflicts.The protocol defines several transport parameters and frames to define and control the behavior of the streams. The MAX_STREAM_DATA frame is advertised by the receiver to flow control individual streams. If a peer attempts to send more data than is advertised, the connection is terminated. MAX_STREAM_ID is a similar frame advertised by the receiver to indicate the maximum number of streams allowed on the connection. If a peer attempts to start a stream with a Stream ID higher than the advertised maximum, the connection is terminated. The sender can communicate with the receiver that it is unable to send more data, or start a new stream through STREAM_BLOCKED and STREAM_ID_BLOCKED frames respectively. The receiver can advertise new data and stream limits any time during a connection and is bound to honor these limits, e.g. a receiver cannot advertise a higher limit and refuse it once a sender starts sending. An important implementation note is that if a QUIC packet is dropped, every stream that was in that packet is blocked. It is up to the QUIC implementation to determine the number of streams to be sent per packet to limit the occurrences of HOL-Blocking. QUIC needs to balance sending data for all its dream with the chance of stream blockage when a dropped packet occurs. Once an endpoint of a stream has finished transmitting its data, it will set the FIN bit on its last STREAM frame or the frame after the last STREAM frame to indicate the stream is closed in the direction of the endpoint, resulting in a half closed stream. Once both endpoints have sent STREAM frames with the FIN bit set, the stream is fully closed.

4.4. Fragmentation

For data model, SCTP uses application-defined messages. However, QUIC adopts the bidirectional byte stream model of TCP, the reasoning of which is probably the desire of close coupling with HTTP/2 that was originally designed to run over TCP. Consequently, not only does it facilitate the movement of applications from TCP to QUIC, but also liberates QUIC from message fragmentation that SCTP has to take care of.

4.4.1. SCTP

In order to avoid IP fragmentation, SCTP fragments its own chunks, so that its packets can fit under the Path MTU [QUIC]. Since SCTP relies on messages as its unit of data, it needs to determine how to fragment and reassemble its payload to keep the rest of the protocol functioning, meaning it needs to keep its headers unfragmented and handle reassembly of the data once it is received. SCTP achieves this by utilizing bit flags in the DATA Chunk header and numeric values in its TSN and SSN fields. The B bit set to 1 indicates that the chunk is the first fragment of a user message. The E bit set to 1 indicates that the chunk is the last fragment of a user message. If both B and E are set to 1, then the message is not fragmented. If both B and E are set to 0, then the chunk is a middle fragment of the user message. The TSN field indicates the Transmission Sequence Number of the DATA Chunk which is used to identify and acknowledge successfully received Chunks. Each DATA Chunk in a packet shares a different sequential TSN and SSN, whereas each fragmented DATA Chunk must also shares a different sequential TSN, but the same SSN among the fragmented DATA Chunks containing the same message. A receiver will then be able to acknowledge all the Chunks it received with its corresponding TSN, and rebuild the underlying messages by matching DATA Chunks with payloads sharing the same SSN.

4.4.2. QUIC

A QUIC packet must contain whole frames, and not have frames split up between packets. A QUIC packet must fit under the Path MTU [QUIC-TRANSPORT]. QUIC can resize packets without the need of complex mechanisms to track fragments of messages like in SCTP since every QUIC data payload is just a byte stream and is easily adjustable through changing byte offset field in the STREAM Frame. There is no message to fragment since the data is already at its most granular form. The actual size of a QUIC packet is determined by implementation of the protocol and how the application using it behaves. The current draft does not go into much detail on how to pack QUIC packets with frames aside from recommending to pack as many frames as possible to minimize per-packet bandwidth and computational cost [QUIC-TRANSPORT]. However a balance needs to occur. If there are too many frames in a packet, and the packet is lost, all those streams are blocked, if there are too little frames, there is increased per-packet bandwidth and computational cost.

4.5. Reliability and Congestion Control

Reliable delivery in transport protocols is defined as providing the abstraction of guaranteeing delivery of every message on an active connection . Congestion control is defined to be how an endpoint

limits the number of messages it sends out on a network in order to prevent the network from becoming clogged and dropping packets. QUIC and SCTP both provide reliable delivery as well as forms of congestion control. SCTP borrows most of its congestion control concepts from TCP and QUIC utilizes TCP's and its own mechanisms. The ACK blocks indicate ranges of packet numbers that were received below the Largest Acknowledged, with GAP blocks indicating gaps in the packet number series. This is unlike SCTP's Cumulative TSN ACK which tracks the lowest contiguous acknowledged TSN.

4.5.1. SCTP

SCTP ensures reliable in-order delivery of data through the use of the TSN. Unlike QUIC's Packet Number, TSN is not a monotonically increasing value. TSNs are used to identify and acknowledge chunks by a receiver, and if a sender does not receive an acknowledgement in a certain amount of time, it knows what chunks to retransmit because of their associated TSN. TSNs are used to track missing chunks, and chunks are retransmitted with the same TSN that they had when they were originally dropped so the receiver knows it is no longer missing a chunk. This allows SCTP to guarantee reliable delivery of DATA Chunks. Since TSN is separate from SSN, the in-order delivery mechanism for streams is kept separate from the reliable delivery mechanism. SSN controls in-order delivery to the ULP, while TSN controls reliable delivery between endpoints. TSN is also agnostic to what stream it belongs to. SCTP keeps track of the Cumulative TSN ACK, the last TSN an endpoint has received before a break in the series of TSN values. Every TSN below the Cumulative TSN ACK value is contiguously acknowledged by the receiver. If a receiver has gaps in TSNs that were not received, it will communicate only what it has received, leaving the sender to determine what is missing. A receiver sends out a SACK Chunk (Figure 7) to acknowledge the receipt of TSNs. GAP blocks are attached to the SACK Chunk to acknowledge sequences of TSN values above the Cumulative TSN ACK. A GAP block indicates ranges of TSNs that are acknowledged by the receiver. Gap Ack Block Start indicates the inclusive start offset of TSNs from the Cumulative TSN ACK. Gap Ack Block End indicates the inclusive end offset of TSNs from the Cumulative TSN ACK. A sender determines what TSNs are missing through repeated GAP blocks containing the same gaps in TSN ranges, which indicate the same chunks are missing repeatedly. The sender will then retransmit the missing chunks. Congestion control in SCTP is governed by the same mechanisms that TCP utilizes such as slow start, fast retransmit and retransmission timer.

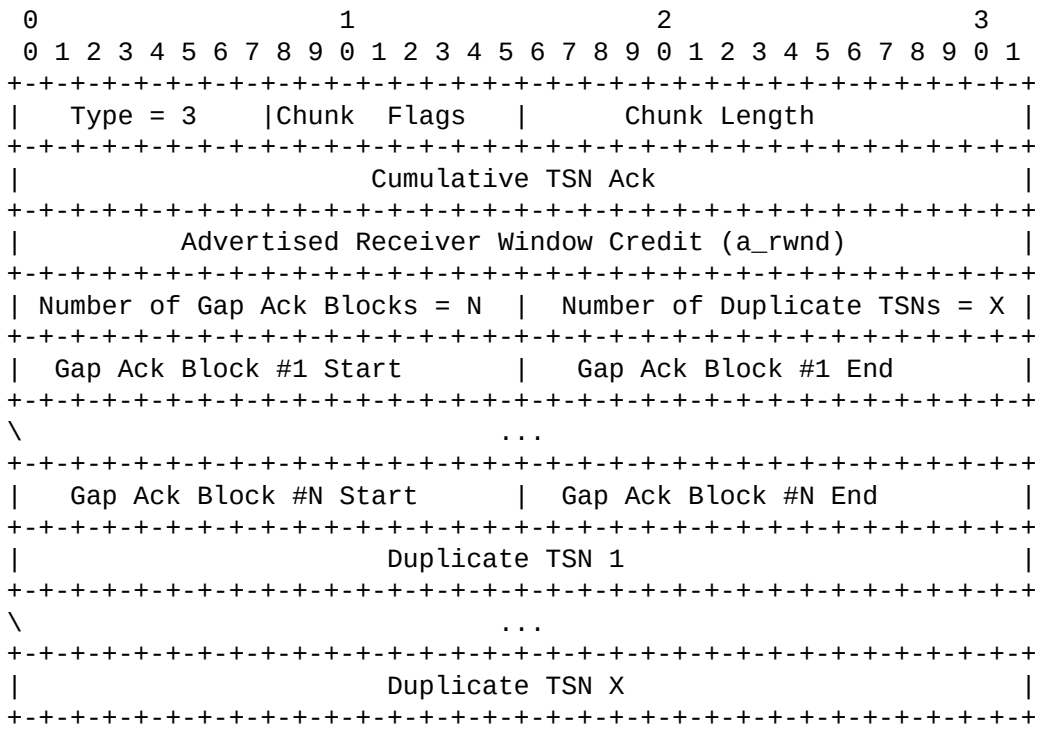

Figure 7: SCTP SACK Chunk

4.5.2. QUIC

QUIC ensures reliable in-order delivery of data through the use of the byte offset field in STREAM frames. If a packet is dropped, the individual frames within the packet will be retransmitted, not the packet itself. This means that a new packet with a new packet number will be constructed, and the dropped frames will be attached and sent with it. The packet number in a QUIC packet is always monotonically increasing, or in other words, a duplicate packet number will never be sent making it easy to distinguish acknowledgements of retransmission from the original packets [QUIC-RECOVERY]. This plays into the stream abstraction concept that is present within QUIC: there is always a constant stream of data being sent on a connection. It is up to the implementation to decide how many packets to use to resend dropped frames. Additionally, since endpoints know which sent packets of theirs is missing, they know what byte offsets are missing, allowing them the ability to resize frames for transmission as they see fit. At time of writing, the QUIC draft does not specify how the frames are resized [QUIC-TRANSPORT]. The ACK Delay field indicates the time in microseconds that the largest acknowledged packet was received by which facilitates creating an accurate RTT timer [QUIC-RECOVERY]. The Largest Acknowledged field in the ACK

Frame (Figure 8) indicates the largest packet number that was received. The reason QUIC tracks the latest packet number is due to the packet number always being monotonically increasing allowing transmission order to be easily tracked. SCTP tracks the lowest contiguous TSN in its Cumulative TSN ACK field since SCTP might retransmit TSNs which is not an issue with QUIC. Just as SCTP utilizes the same mechanisms as TCP for congestion control, so does QUIC, however with some important modifications. QUIC simplifies its congestion control and loss detection by splitting out its source of information for reliable delivery: stream id and byte offsets, from its source of information for transmission order: monotonically increasing packet numbers. SCTP and TCP both conflate reliable delivery and transmission order into one source of information, the TSNs. Another simplification that QUIC brings is that QUIC ACK's are always honored, and never reneged upon, unlike SCTP which uses a SACK similar to TCP and can be reneged [QUIC-RECOVERY]. TCP's congestion control algorithms such as slow start, fast retransmit, and RTT timers are still used in QUIC, just adapted to use its packet number as well as some other minute differences [QUIC-RECOVERY].

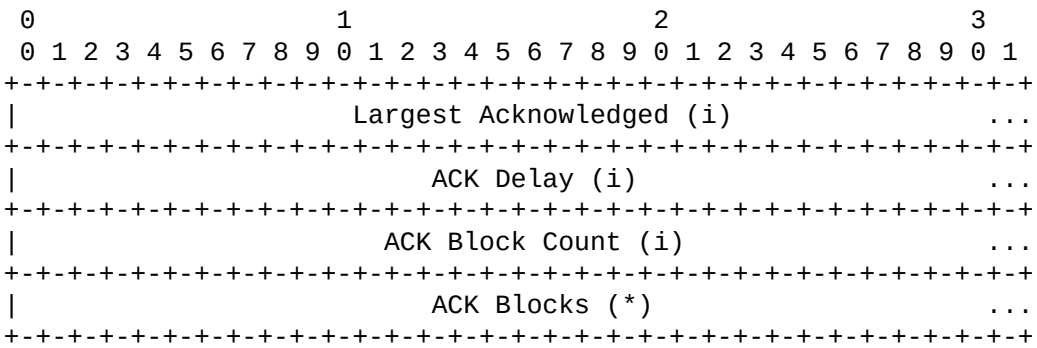

Figure 8: QUIC ACK Frame

4.6. Flow Control

Flow control is defined as the pressure or limit a receiving endpoint advertises to a sender in order to prevent the receiver from being overwhelmed and drop packets. Flow control is similar to congestion control, but whereas congestion control focuses on preventing congestion on the network or system, flow control focuses on preventing an endpoint from being overwhelmed. A common flow control concept is a sliding window, in which an endpoint advertises an amount of bytes its sending counterpart can transmit. Both of these protocols practice a form of sliding window. Unlike UIC, there is no flow control data that is sent between sender and receiver on a per stream basis, but rather flow control is done on a per association basis.

4.6.1. SCTP

Flow Control in SCTP is done only on a per association basis using mechanisms similar to TCP as defined in TCP Congestion Control [RFC2581]. When a receiver sends out a SACK Chunk (Figure 7), it includes a field called Advertiser Receiver Window Credit (a_rwnd). This value represents the remaining available buffer space at the receiver. Since SACKs can be received out of order, a sender will not necessarily assume they have a_rwnd amount of buffer space to send. At the start of a SCTP association, each endpoint will receive the a_rwnd of its peer in the INIT (Figure 5) or INIT ACK Chunk, and will take that to be the actual receiving window (rwnd) of the corresponding endpoint. As the association lifetime continues, each endpoint will subtract the size of DATA chunks that are sent or retransmitted to a peer from the peer's rwnd. This is because the sender assumes the peer's buffer space will be taken up by the transmitted chunk. Each endpoint will also add the size of DATA chunks that are marked for retransmission. With each SACK an endpoint receives, it will update its rwnd according to a_rwnd in the SACK, minus any outstanding bytes from missing chunks that have not be acknowledged yet.

4.6.2. QUIC

Flow Control in QUIC is done on both a connection and substream basis. The most important parameters for flow control in QUIC are the transport parameters MAX_DATA and MAX_STREAM_DATA parameters. These two parameters are communicated during connection setup, and also have corresponding Frames which can be communicated during a connection. Once a value is advertised for these parameters by an endpoint, the endpoint must honor it. MAX_DATA indicates the maximum amount of data that can be communicated on a connection. MAX_STREAM_DATA indicates the maximum amount of data that can be communicated on a stream basis. It is up to each endpoint to divide up the data between all of its streams. As the connection and stream lifetime continues, endpoints will advertise higher MAX_DATA and MAX_STREAM_DATA to flow control its sending peer. If either of these variables are disobeyed by a sender on any of the streams, the entire connection is torn down. An exception is made for Stream 0, which is reserved for the cryptographic handshake on setup. None of the byte usage of Stream 0 is counted towards the limits of the transport parameters [QUIC-TLS]. Since QUIC utilizes a byte stream paradigm and byte offsets are communicated in STREAM frames, data usage is easily calculated on both endpoints by recording largest received byte offsets. This leads to virtually no chance of an endpoint breaking this agreement unless there is a bug in its implementation or it is a malicious actor.

4.7. Connection Teardown

4.7.1. SCTP

Once it is time for a SCTP association to end, the endpoints engage in a 3-way handshake to shutdown the association. The ULP will send out a SHUTDOWN primitive to the lower layer where it will wait for all its sent chunks to be acknowledged or retransmit missing ones. The endpoint will then send out a SHUTDOWN chunk to initiate a clean close of the association after it has confirmed its peer has received all sent data. On receipt of the SHUTDOWN chunk, the peer endpoint will stop accepting data from its ULP and confirm it has received all data and then respond with a SHUTDOWN-ACK. Finally, the initiating endpoint will send out a SHUTDOWN-Complete chunk to close the association.

4.7.2. QUIC

Once it is time for a QUIC connection to shut down, an endpoint sends out a closing frame, CONNECTION_CLOSE or APPLICATION_CLOSE to its peer and enters a closing state in which it discards all internal state except what is required to build closing frames. If there are open substreams when the frame is received, the streams are implicitly closed. If the initiator of the shutdown receives packets while it is in a closing state, it replies to each of them with either a closing frame. The receiver of the closing frame enters a draining state in which it does not send anymore packets and discards internal state. Before the receiver enters the draining state, it can also send a closing frame. At most, a QUIC connection teardown is a two-way handshake unless there are dropped packets from the initiator. Another way the connection might close down is implicitly through no network activity, resulting in the endpoints timing out.

4.8. Other Differences between QUIC and SCTP

SCTP supports multi-homing. Specifically, an endpoint can include multiple IP addresses in the INIT or INIT ACK chunk, so the other endpoint can establish a multi-path connection with the endpoint. When one of the connections times out, a chunk can be retransmitted through another active connection, increasing the resilience of SCTP connection. Nonetheless, QUIC itself does not support multi-homing. Instead, there exists an Multipath Extension for QUIC Draft working in progress to add multipath capability into QUIC protocol [MULTIPATH-QUIC] .

QUIC greatly resembles the combination of TCP, TLS and HTTP/2. QUIC packets are always encrypted (except for the public header) and authenticated (including the public header). The encryption prevents

middle box parsing the congestion information and breaking with any forward changes, which is currently a problem for TCP. The public header is required either for routing or for decrypting the packet so it is unencrypted. However, this packet is also fully authenticated, preventing in-network tampering. Any modification of the QUIC packet will cause the teardown of the connection. Nevertheless, SCTP protocol itself does not include encryption or authentication, just like TCP.

5. Current Situations of SCTP and QUIC

Temporarily, SCTP is used mostly in the telecom industry. However, as for the IP network, the deployment of SCTP is not much widespread. In-network devices, for example, NAT gateways, does not support SCTP well. NAT gateways need to be upgraded to be SCTP-aware. Nevertheless, the cruel truth is that modification of middle boxes is very expensive, and internet service providers are supposed to seek their own interests to update the devices. As a matter of fact, some firewalls only allow TCP or UDP packets to pass through, which constrains SCTP to very small living space. Considering that MPTCP can meet such needs, there is less motivation to deploy SCTP. The worse thing is that, unlike MPTCP, the SCTP socket APIs differ greatly from TCP, and developers need to update their source code to deploy SCTP, thus significantly impeding the wide deployment of SCTP.

Designed by Google, QUIC is now widely used in Chrome clients accessing Google services. QUIC is deployed as a substitution of SPDY, representing about 7% of the Internet traffic. QUIC works atop of UDP, so mostly in-network devices that support UDP will support QUIC. At least, it is more friendly to middleboxes than SCTP. Since QUIC works in the application layer, it is supposed to be upgraded much more frequently than TCP stack in kernel. Fortunately, QUIC provides a new set of APIs, which are not transparent to the upper applications. Similar to SCTP, developers also need to rewrite the source code to allow the former applications to use QUIC. Tech giants, like Tencent, are trying to deploy QUIC to provide better service for users. With the support of giants and communities, the deployment of QUIC is promising in the future.

6. Conclusions from the comparison

QUIC has adopted a number of features from long years of protocol design efforts. QUIC and SCTP share some commonalities and differences. We conclude some design considerations of QUIC as following.

- Latency: QUIC combines transport and crypto handshakes, utilizing cryptographic cookie for connection resumption, minimizing connection latency.

- Security: QUIC packets are always encrypted (except for the public header) and authenticated (including the public header). QUIC also address the security issues inherent in allowing data exchange during the 0-rtt handshake, through the use of a security token for address validation. However, QUIC's use longterm cryptographic cookie and connection ID brings window for new types of attacks. Balancing tradeoff of gains and losses is always a part of protocol design.

- Compatibility: QUIC runs in userspace, allowing fast deployment and experimentation. As it runs over UDP, it is compatible with most middlebox implementations. QUIC also adopts a fall back mechanism for normal TCP handshake incase one of the parties do not support the protocol. QUIC also adopts congestion control protocol to achieve fairness with existing TCP connections. The compatibility issue is one of the reasons why SCTP was not widely deployed.

- Parallelism: Through stream multiplexing, the missing frames of one stream will not block the delivery of other streams payload data, avoiding HOL-Blocking problem, but also introduces additional processing, as QUIC has to keep internal mapping of which stream frame in which packet, to know which one needs to be retransmitted.

- Flexibility: QUIC has a pluggable congestion control mechanism and has more signaling than TCP, which makes QUIC more informative for congestion control algorithms. It also provides opportunities for further experimentation of congestion control features.

- Fine granularity: QUIC supports the traffic control both in stream and connection level, following HTTP/2.

- Adjustability: The QUIC connection can survive IP address changes and NAT rebinding due to the stable connection ID during connection migration.

- Lightness: QUIC adopts the bidirectional byte stream model of TCP, which facilitates the movement of applications from TCP to QUIC and liberates QUIC from message fragmentation that SCTP has to take care of.

Hopefully these advantages of QUIC can serve as the general principles for future development of QUIC and the design of other incipient protocols.

7. Contributors

Hang Shi
Tsinghua University
P.R. China
Email: shihang7422166@gmail.com

Yuming Hu
Tsinghua University
P.R. China
Email: Kumius@foxmail.com

8. Informative References

[MULTIPATH-QUIC]
Coninck, Q. and O. Bonaventure, "Multipath Extension for QUIC, https://tools.ietf.org/html/draft-deconinck-multipath-quic-00", draft-tsvwg-quic-protocol-00 (work in progress), October 2017.

[QUIC] Iyengar, J. and I. Swett, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2, https://tools.ietf.org/html/draft-tsvwg-quic-protocol-00", draft-tsvwg-quic-protocol-00 (work in progress), June 2015.

[QUIC-DESIGN]
"QUIC: Design Document and Specification Rationale, Jim Roskind, Chromium Contributor", 2012.

[QUIC-RECOVERY]
Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", draft-ietf-quic-recovery-09 (work in progress), January 2018.

[QUIC-TLS]
Thomson, M., Ed. and S. Turner, Ed., "Using Transport Layer Security (TLS) to Secure QUIC", draft-ietf-quic-tls-09 (work in progress), January 2018.

[QUIC-TRANSPORT]
Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-09 (work in progress), January 2018.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, DOI 10.17487/RFC2581, April 1999, <https://www.rfc-editor.org/info/rfc2581>.

[RFC4896] Surtees, A., West, M., and A. Roach, "Signaling Compression (SigComp) Corrections and Clarifications", RFC 4896, DOI 10.17487/RFC4896, June 2007, <https://www.rfc-editor.org/info/rfc4896>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

Authors' Addresses

Arun Joseph
UCLA
Los Angeles
USA

Email: ajoseps@ucla.edu

Tianxiang Li
UCLA
Los Angeles
USA

Email: tianxiang@cs.ucla.edu

Zihao He
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-10-6278-5822
Email: hezihao9512@gmail.com

Yong Cui
Tsinghua University
Beijing 100084
P.R.China

Phone: +86-10-6260-3059
Email: cuiyong@tsinghua.edu.cn

Lixia Zhang
UCLA
Los Angeles
USA

Email: lixia@cs.ucla.edu