Workgroup: Network Working Group
Internet-Draft: draft-templin-dtn-ltpfrag-16
Published: 23 October 2023
Intended Status: Informational
Expires: 25 April 2024
Authors: F. L. Templin, Ed.
Boeing Research & Technology

LTP Fragmentation

Abstract

The Licklider Transmission Protocol (LTP) provides a reliable datagram convergence layer for the Delay/Disruption Tolerant Networking (DTN) Bundle Protocol. In common practice, LTP is often configured over UDP/IP sockets and inherits its maximum segment size from the maximum-sized UDP/IP datagram, however when this size exceeds the path maximum transmission unit a service known as IP fragmentation must be engaged. This document discusses LTP interactions with IP fragmentation and mitigations for managing the amount of IP fragmentation employed.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Licklider Transmission Protocol (LTP) [RFC5326] provides a reliable datagram convergence layer for the Delay/Disruption Tolerant Networking (DTN) Bundle Protocol (BP) [RFC9171]. In common practice, LTP is often configured over the User Datagram Protocol (UDP) [RFC0768] and Internet Protocol (IP) [RFC0791] using the "socket" abstraction. LTP inherits its maximum segment size from the maximum-sized UDP/IP datagram (i.e., 64KB minus header sizes), however when that size exceeds the maximum transmission unit the path can support a service known as IP fragmentation must be engaged.

LTP breaks BP bundles into "blocks", then further breaks these blocks into "segments". The segment size is a configurable option and represents the largest atomic portion of data that LTP will require underlying layers to deliver as a single unit. The segment size is therefore also known as the "retransmission unit", since each lost segment must be retransmitted in its entirety. Experimental and operational evidence has shown that on robust networks increasing the LTP segment size (up to the maximum UDP/IP datagram size of slightly less than 64KB) can result in substantial performance increases over smaller segment sizes. However, the performance increases must be tempered with the amount of IP fragmentation invoked as discussed below.

When LTP presents a segment to the operating system kernel (e.g., via a sendmsg() system call), the UDP layer prepends a UDP header to create a UDP datagram. The UDP layer then presents the resulting datagram to the IP layer for packet framing and transmission over a networked path. The path is further characterized by the path Maximum Transmission Unit (Path-MTU) which is a measure of the smallest link MTU (Link-MTU) among all links in the path.

When LTP presents a segment to the kernel that is larger than the Path-MTU, the resulting UDP datagram is presented to the IP layer which in turn performs IP fragmentation to break the datagram into fragments that are no larger than the Path-MTU. For example, if the LTP segment size is 64KB and the Path-MTU is 1280 octets IP fragmentation results in 50+ fragments that are transmitted as individual IP packets. (Note that for IPv4 [RFC0791], fragmentation may occur either in the source host or in a router in the network path, while for IPv6 [RFC8200] only the source host may perform fragmentation.)

Each IP fragment is subject to the same best-effort delivery service offered by the network according to current congestion and/or link signal quality conditions; therefore, the IP fragment size becomes known as the "loss unit". Especially when the packet loss rate is non-negligible, however, performance can suffer dramatically when the loss unit is significantly smaller than the retransmission unit during periods of congestion. In particular, if even a single IP fragment of a fragmented LTP segment is lost then the whole LTP segment is considered lost and must be retransmitted in its entirety. Since LTP does not support flow control or congestion control, this can result in a cascading flood of redundant information when fragments are systematically lost in transit due to congestion or disruption.

This document discusses LTP interactions with IP fragmentation and mitigations for managing the amount of fragmentation employed. It further discusses methods for increasing LTP performance even when IP fragmentation is engaged.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

3. IP Fragmentation Issues

IP fragmentation is a fundamental service of the Internet Protocol, yet it has long been understood that its use can be problematic in some environments. Beginning as early as 1987, "Fragmentation Considered Harmful" [FRAG] outlined multiple issues with the service including a performance-crippling condition that can occur at high data rates when the loss unit is considerably smaller than the retransmission unit during intermittent and/or steady-state loss conditions.

Later investigations also identified the possibility for undetected corruption at high data rates due to a condition known as "ID wraparound" when the 16-bit IP identification field (aka the "IP ID") increments such that new fragments overlap with existing fragments still alive in the network and with identical ID values [RFC4963][RFC6864]. Although this condition is most acute for the IPv4 protocol (and much less so for IPv6 where the IP ID is 32-bits in length), the IPv4 concerns along with the fact that IPv6 does not permit routers to perform "network fragmentation" have led many to discourage the use of fragmentation whenever possible.

Even in the modern era, investigators have seen fit to declare "IP Fragmentation Considered Fragile" in an Internet Engineering Task Force (IETF) Best Current Practice (BCP) reference [RFC8900]. Indeed, the BCP recommendations cite the Bundle Protocol LTP convergence layer as a user of IP fragmentation that depends on some of its properties to realize greater performance. However, the BCP summarizes by saying:

> "Rather than deprecating IP fragmentation, this document recommends that upper-layer protocols address the problem of fragmentation at their layer, reducing their reliance on IP fragmentation to the greatest degree possible."

This conclusion was based on the historical state of IP fragmentation and did not seem to consider the opportunity for forward-looking improvements. With the advent of "Identification Extension for the Internet Protocol" [I-D.templin-intarea-ipid-ext], however, the status of IP fragmentation may soon need to be recharacterized from "fragile" to "robust". We therefore next discuss our systematic approach to LTP fragmentation while considering IP fragmentation as a potentially useful tool for performance maximization.

4. LTP Fragmentation

In common LTP implementations over UDP/IP (e.g., the Interplanetary Overlay Network (ION)), performance is greatly dependent on the LTP segment size. This is due to factors including that larger segments reduce the number of segments LTP has to manage and that larger segments presented to UDP/IP as single units incur only a single system call with a single data copy from application to kernel space via the sendmsg() system call. Once inside the kernel, each segment incurs UDP/IP encapsulation and IP fragmentation.

During fragmentation, each fragment is transmitted immediately following the previous without delay so that the fragments appear as a "burst" of consecutive packets over the network path resulting in high network utilization during the burst period. Additionally, the use of IP fragmentation with a larger segment size conserves header framing octets since the upper layer headers only appear in the first IP fragment as opposed to appearing in all fragments.

Conventional wisdom has for many decades suggested that in order to avoid retransmission congestion (i.e., especially when fragment loss probability is non-negligible) the LTP segment size should be set to no larger than the Path-MTU. Assuming the minimum IPv4 Effective MTU to Receive (EMTU_R) of 576 octets, however, transmission of 64KB of data using a 576B segment size would require well over 100 independent sendmsg() system calls and data copies as opposed to just one when the largest segment size is used. This greatly reduces the theoretical bandwidth advantage offered by IP fragmentation bursts. Therefore, a means for providing the best aspects of both large segment fragment bursting and small segment retransmission efficiency would seem beneficial.

Common operating systems such as linux provide the sendmmsg() ("send multiple messages") system call that allows the LTP application to present the kernel with a vector of up to 1024 segments instead of just a single segment. This theoretically affords the bursting behavior of IP fragmentation coupled with the retransmission efficiency of employing small segment sizes. (Note that LTP receivers can also use the recvmmsg() ("receive multiple messages") system call to receive a vector of segments from the kernel in case multiple recent packet arrivals can be combined.)

A first approach to performance maximization therefore analyzed implementations of LTP that employ a large block size, a conservative segment size and a new configuration option known as the "Burst-Limit" which determines the number of segments that can be presented in a single sendmmsg() system call. When the implementation forwards an LTP block, it carves Burst-Limit-many segments from the block and presents the vector of segments to sendmmsg(). The kernel will prepare each segment as an independent UDP/IP packet and transmit them into the network as a burst in a fashion that parallels IP fragmentation. The loss unit and retransmission unit will be the same, therefore loss of a single segment does not result in a retransmission congestion event.

It should be noted that the Burst-Limit is bounded only by the LTP block size and not by the maximum UDP/IP datagram size. Therefore, each burst can in practice convey significantly more data than a single IP fragmentation event. It should also be noted that the segment size can still be made larger than the Path-MTU in low-loss environments without danger of triggering retransmission storms due to loss of IP fragments. This would result in combined large UDP/IP message transmission and IP fragmentation bursting for increased network utilization in more robust environments. Finally, both the Burst-Limit and UDP/IP message sizes need not be static values, and can be tuned to adaptively increase or decrease according to time varying network conditions.

5. Beyond "sendmmsg()"

In actual practice, implementation experience with the ION-DTN distribution along with two recent studies have demonstrated only very limited performance increases for employing sendmmsg() for transmission over UDP/IP sockets. A first study used sendmmsg() as part of an integrated solution to produce 1M packets per second assuming only raw data transmission conditions [MPPS], while a second study focused on performance improvements for the QUIC reliable transport service [QUIC]. In both studies, the use of sendmmsg() alone produced modest increases but complimentary enhancements were identified that when combined with sendmmsg() produced considerable additional increases.

In [MPPS], additional enhancements such as using recvmmsg() and configuring multiple receive queues at the receiver were introduced in an attempt to achieve greater parallelism and engage multiple processors and threads. However, the system was still limited to a single thread until multiple receiving processes were introduced using the "SO_REUSEPORT" socket option. By having multiple receiving processes (each with its own socket buffer), the performance advantages of parallel processing were employed to achieve the 1M packets per second goal.

In [QUIC], a new feature available in recent linux kernel versions was employed. The feature, known as "Generic Segmentation Offload (GSO) / Generic Receive Offload (GRO)" allows an application to provide the kernel with a "super-buffer" containing up to 64 separate upper layer protocol segments. When the application presents the super-buffer to the kernel, GSO segmentation then sends up to 64 separate UDP/IP packets in a burst. (Note that GSO requires each UDP/IP packet to be no larger than the Path-MTU so that receivers can invoke GRO without interactions with IP reassembly.)

The GSO facility can be invoked by either sendmsg() (i.e., a single super-buffer) or sendmmsg() (i.e., multiple super-buffers), and the study showed a substantial performance increase over using just sendmsg() and sendmmsg() alone.

For LTP fragmentation, our ongoing efforts explore using these techniques in a manner that parallels the effort undertaken for QUIC. Using these higher-layer segmentation management facilities is consistent with the guidance in "IP Fragmentation Considered Fragile" that states:

> "Rather than deprecating IP fragmentation, this document recommends that upper-layer protocols address the problem of fragmentation at their layer, reducing their reliance on IP fragmentation to the greatest degree possible."

By addressing fragmentation at their layer, the LTP/UDP functions can then be tuned to minimize IP fragmentation in environments where it may be problematic or to adaptively engage IP fragmentation in environments where performance gains can be realized without risking sustained loss and/or data corruption.

6. Advanced LTP Performance Enhancement

Some modern operating systems include Generic Segment Offload (GSO) and Generic Receive Offload (GRO) services. For example, GSO/GRO support has been included in linux beginning with kernel version 4.18. Some network drivers and network hardware also support GSO/GRO at or below the operating system network device driver interface layer to provide benefits of delayed segmentation and/or early reassembly. The following sections discuss LTP interactions with GSO and GRO.

6.1. LTP and GSO

GSO allows LTP implementations to present the sendmsg() or sendmmsg() system calls with "super-buffers" that include up to 64 LTP segments which the kernel will subdivide into individual UDP/IP datagrams. LTP implementations enable GSO either on a per-socket basis using the "setsockopt()" system call or on a per-message basis for sendmsg()/sendmmsg() as follows:

```
/* Set LTP segment size */
unsigned integer gso_size = SEGSIZE;
...
/* Enable GSO for all messages sent on the socket */
setsockopt(fd, SOL_UDP, UDP_SEGMENT, &gso_size, sizeof(gso_size)));
...
/* Alternatively, set per-message GSO control */
cm = CMSG_FIRSTHDR(&msg);
cm->cmsg_level = SOL_UDP;
cm->cmsg_type = UDP_SEGMENT;
cm->cmsg_len = CMSG_LEN(sizeof(uint16_t));
*((uint16_t *) CMSG_DATA(cm)) = gso_size;
```

Implementations must set SEGSIZE to a value no larger than the path MTU via the underlying network interface, minus header overhead; this ensures that UDP/IP datagrams generated during GSO segmentation will not incur local IP fragmentation prior to transmission (Note: the linux kernel returns EINVAL if SEGSIZE encodes a value that exceeds the Path-MTU.)

Implementations should therefore dynamically determine SEGSIZE for paths that traverse multiple links through Packetization Layer Path MTU Discovery for Datagram Transports [RFC8899] (DPMTUD). Implementations should set an initial SEGSIZE to either a known minimum MTU for the path or to the protocol-defined minimum path MTU. Implementations may then dynamically increase SEGSIZE without service interruption if the discovered Path-MTU is larger.

6.2. LTP and GRO

GRO allows the kernel to return "super-buffers" that contain multiple concatenated received segments to the LTP implementation in recvmsg() or recvmmsg() system calls, where each concatenated segment is distinguished by an LTP segment header per [RFC5326]. LTP implementations enable GRO on a per-socket basis using the "setsockopt()" system call, then optionally set up per receive message ancillary data to receive the segment length for each message as follows:

```
/* Enable GRO */
unsigned integer use_gro = 1; /* boolean */
setsockopt(fd, SOL_UDP, UDP_GRO, &use_gro, sizeof(use_gro)));
...
/* Set per-message GRO control */
cmsg->cmsg_len = CMSG_LEN(sizeof(int));
*((int *)CMSG_DATA(cmsg)) = 0;
cmsg->cmsg_level = SOL_UDP;
cmsg->cmsg_type = UDP_GRO;
...
/* Receive per-message GRO segment length */
if ((segmentLength = *((int *)CMSG_DATA(cmsg))) <= 0)
     segmentLength = messageLength;
```

Implementations include a pointer to a "use_gro" boolean indication to the kernel to enable GRO; the only interoperability requirement therefore is that each UDP/IP packet includes an integral number of properly-formed LTP segments. The kernel and/or underlying network hardware will first coalesce multiple received segments into a larger single segment whenever possible and/or return multiple coalesced or singular segments to the LTP implementation so as to maximize the amount of data returned in a single system call. The "super-buffer" thus prepared MUST contain at most 64 segments where each non-final segment MUST be equal in length and the final segment MUST NOT be longer than the non-final segment length.

Implementations that invoke recvmsg( ) and/or recvmmsg() will therefore receive "super-buffers" that include one or more concatenated received LTP segments. The LTP implementation accepts all received LTP segments and identifies any segments that may be missing. The LTP protocol then engages segment report procedures if necessary to request retransmission of any missing segments.

6.3. LTP GSO/GRO Over OMNI Interfaces

LTP engines produce UDP/IP packets that can be forwarded over an underlying network interface as the head-end of a "link-layer service that transits IP packets". UDP/IP packets that enter the link near-end are deterministically delivered to the link-far end modulo loss due to corruption, congestion or disruption. The link-layer service is associated with an MTU that deterministically establishes the maximum packet size that can transit the link. The link-layer service may further support a segmentation and reassembly function with fragment retransmissions at a layer below IP; in many cases, these timely link-layer retransmissions can reduce dependency on (slow) end-to-end retransmissions.

LTP engines that connect to networks traversed by paths consisting of multiple concatenated links must be prepared to adapt their segment sizes to match the minimum MTU of all links in the path. This could result in a small SEGSIZE that would interfere with the benefits of GSO/GRO layering. However, nodes that configure LTP engines can also establish an Overlay Multilink Network Interface (OMNI) [I-D.templin-intarea-omni] that spans the multiple concatenated links while presenting an assured (64KB-1) MTU to the LTP engine.

The OMNI interface internally uses IPv6 fragmentation as an OMNI Adaptation Layer (OAL) service invisible to the LTP engine to allow timely link-layer retransmissions of lost fragments where the retransmission unit matches the loss unit. The LTP engine can then dynamically vary its SEGSIZE (up to a maximum value of (64KB-1) minus headers) to determine the size that produces the best performance at the current time by engaging the combined operational factors at all layers of the multi-layer architecture. This dynamic factoring coupled with the ideal link properties provided by the OMNI interface support an effective layering solution for many DTN networks.

When an LTP/UDP/IP packet is transmitted over an OMNI interface, the OAL inserts an IPv6 header and performs IPv6 fragmentation to produce fragments small enough to fit within the Path-MTU. The OAL then replaces the IPv6 encapsulation headers with OMNI Compressed Headers (OCHs) which are significantly smaller that their uncompressed IPv6 header counterparts and even smaller than the IPv4 headers would have been had the packet been sent directly over a physical interface such as Ethernet using IPv4 fragmentation. These fragments are finally wrapped in lower layer headers to produce "carrier packets" as necessary to transit the path.

The end result is that the first fragment produced by the OAL will include a small amount of additional overhead to accommodate the OCH encapsulation header while all additional fragments will include only an OCH header which is significantly smaller than even an IPv4 header. The act of forwarding the large LTP/UDP/IP packet over the OMNI interface will therefore produce a considerable overhead savings in comparison with direct Ethernet transmission.

Using the OMNI interface with its OAL service in addition to the GSO/GRO mechanism, an LTP engine can therefore theoretically present concatenated LTP segments in a "super-buffer" of up to (64 * ((64KB-1) minus headers)) octets for transmission in a single sendmsg() system call, and may present multiple such "super-buffers" in a single system call when sendmmsg() is used. (Note however that existing implementations limit the maximum-sized "super-buffer" to only 64KB total.) In the future, this service may realize even greater benefits through the use of advanced IP Jumbograms ("advanced jumbos") [I-D.templin-intarea-parcels] over paths that support them.

6.4. IP Parcels

The so-called "super-buffers" discussed in the previous sessions can be applied for GSO/GRO only when the LTP application endpoints are co-resident with the OAL source and destination, respectively. However, it may be desirable for the future architecture to support network forwarding for these "super-buffers" in case the LTP source and/or destination are located one or more IP networking hops away from nodes that configure their respective source and destination OMNI interfaces. Moreover, if the OMNI virtual link spans multiple OMNI intermediate nodes on the path from the OAL source to the OAL destination it may be desirable to keep the "super-buffers" together as much as possible as they traverse the intermediate hops. For this reason, a new construct known as the "IP Parcel" has been specified [I-D.templin-intarea-parcels].

An IP parcel is a special form of an IP Jumbogram that includes a non-zero value in the IP {Total, Payload} Length field. The value sets the segment size for the first segment included in the parcel, while the value coded in the Jumbo Payload header provides the full length of the parcel and determines the number of segments included. Each segment "shares" a single IP header, and the parcel can be broken down into sub-parcels if necessary to traverse paths with length restrictions. The parcel therefore is a "packet-of-packets" that offers more efficient packaging in the same way that postal service shipping parcels containing multiple items offer more efficient shipping.

IP parcels as well as another new form of IP Jumbogram known as the "advanced jumbo" can also be forwarded as whole packets over paths that traverse links with sufficiently large MTUs (e.g., space domain laser links). The source performs path probing to determine whether IP parcels and/or advanced jumbos are supported, after which it may begin forwarding packets that employ these new constructs. A full discussion of IP parcels and advanced jumbos is found in [I-D.templin-intarea-parcels].

6.5. IP Fragmentation Revisited

More recent studies have demonstrated a clear performance advantage for LTP/UDP when using conventional segment sizes that significantly exceed the Path-MTU. For example, widely-deployed LTP/UDP implementations show a multiplicative performance increase for using maximum-sized conventional LTP segments in comparison to smaller segments. Indeed, increasing the LTP segment size in live network tests over 100Gbps links significantly exceeded the performance characteristics for Path-MTU or smaller-sized segments.

Significant performance increases were also observed when the Path-MTU itself was increased, with the greatest performance occurring when both the segment size and Path-MTU approached their maximum values. When the segment size exceeds the Path-MTU, fragmentation at some layer is a natural consequence but in our experiments IP fragmentation had no adverse performance impact. This proves that using larger LTP segments (and therefore reducing the number of segments LTP must manage) is the key enabler for greater performance.

The questions of how to avoid the possibility of reassembly corruption due to IP ID wraparound at high data rates and how to mitigate congestive fragment loss have for many decades impeded full dependence on fragmentation. However, these questions and others have now been addressed in "Identification Extension for the Internet Protocol" [I-D.templin-intarea-ipid-ext]. The new ability to extend the IP Identification field to 32-bit, 64-bit or even larger sizes obviates the vulnerability documented in [RFC4963], while the fragmentation control message feedback supports adaptive congestion mitigation. These new functions allow LTP senders and receivers to fully engage the fragmentation and reassembly services which were always intended as core aspects of the Internet architecture. This results in an internetworking service for LTP that is adaptive, efficient and not subject to wasted transmissions.

The encouraging results with conventional segment sizes as large as 65535 octets invites the question of whether even greater performance increases are possible using still larger segments. Such large segments must be carried in packets known as jumbograms for which no fragmentation and reassembly are possible. Although no links currently configure MTUs larger than 65535 octets, future experiments with larger link MTU sizes using Forward Error Correction (FEC) instead of traditional packet integrity checks should yield even greater performance benefits. These links can only be discovered and utilized using Path MTU Discovery (PMTUD).

In conclusion, the answer to the LTP/UDP performance optimization question under conventional packet sizes is not simply unmitigated fragmentation and reassembly but rather intelligently managed and adaptive services that can tune the system for optimum performance under any conditions. As a result, "Identification Extension for the Internet Protocol" provides a near-term solution for LTP performance maximization, while "IP Parcels and Advanced Jumbos" promises to advance performance to its ultimate aspirations.

7. Implementation Status

Supporting code for invoking the sendmmsg() facility is included in the official ION source code distribution, beginning with release ion-4.0.1.

Working code for GSO/GRO has been incorporated into a pre-release of ION and scheduled for integration following the next major release.

An implementation of LTP/UDP/IP Parcels over OMNI interfaces is available in github. A multi-threaded LTP receiver implementation is currently under investigation.

8. IANA Considerations

This document introduces no IANA considerations.

9. Security Considerations

Communications networking security is necessary to preserve confidentiality, integrity and availability.

10. Acknowledgements

The NASA Space Communications and Networks (SCaN) directorate coordinates DTN activities for the International Space Station (ISS) and other space exploration initiatives.

Akash Agarwal, Madhuri Madhava Badgandi, Keith Philpott, Bill Pohlchuck, Vijayasarathy Rajagopalan, Bhargava Raman Sai Prakash and Eric Yeh are acknowledged for their significant contributions. Tyler Doubrava was the first to mention the "sendmmsg()" facility. Scott Burleigh provided review input, and David Zoller provided useful perspective.

Honoring life, liberty and the pursuit of happiness.

11. References

11.1. Normative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5326] Ramadas, M., Burleigh, S., and S. Farrell, "Licklider Transmission Protocol - Specification", RFC 5326, DOI 10.17487/RFC5326, September 2008, <https://www.rfc-editor.org/info/rfc5326>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

11.2. Informative References

[FRAG] Mogul, J. and C. Kent, "Fragmentation Considered Harmful, ACM Sigcomm 1987", August 1987.

[I-D.templin-intarea-ipid-ext]
Templin, F., "Identification Extension for the Internet Protocol", Work in Progress, Internet-Draft, draft-templin-intarea-ipid-ext-21, 13 October 2023, <https://datatracker.ietf.org/doc/html/draft-templin-intarea-ipid-ext-21>.

[I-D.templin-intarea-omni]
Templin, F., "Transmission of IP Packets over Overlay Multilink Network (OMNI) Interfaces", Work in Progress, Internet-Draft, draft-templin-intarea-omni-49, 18 October 2023, <https://datatracker.ietf.org/doc/html/draft-templin-intarea-omni-49>.

[I-D.templin-intarea-parcels]
Templin, F., "IP Parcels and Advanced Jumbos", Work in Progress, Internet-Draft, draft-templin-intarea-parcels-79, 13 October 2023, <https://datatracker.ietf.org/doc/html/draft-templin-intarea-parcels-79>.

[MPPS] Majkowski, M., "How to Receive a Million Packets Per Second, https://blog.cloudflare.com/how-to-receive-a-million-packets/", June 2015.

[QUIC] Ghedini, A., "Accelerating UDP Packet Transmission for QUIC, https://calendar.perfplanet.com/2019/accelerating-udp-packet-transmission-for-quic/", December 2019.

[RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, DOI 10.17487/RFC2675, August 1999, <https://www.rfc-editor.org/info/rfc2675>.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <https://www.rfc-editor.org/info/rfc4963>.

[RFC6864] Touch, J., "Updated Specification of the IPv4 ID Field", RFC 6864, DOI 10.17487/RFC6864, February 2013, <https://www.rfc-editor.org/info/rfc6864>.

[RFC8899] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

[RFC8900] Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", BCP 230, RFC 8900, DOI 10.17487/RFC8900, September 2020, <https://www.rfc-editor.org/info/rfc8900>.

[RFC9171] Burleigh, S., Fall, K., and E. Birrane, III, "Bundle Protocol Version 7", RFC 9171, DOI 10.17487/RFC9171, January 2022, <https://www.rfc-editor.org/info/rfc9171>.

Author's Address

Fred L. Templin (editor)
Boeing Research & Technology
P.O. Box 3707
Seattle, WA 98124
United States of America

Email: fltemplin@acm.org