ROLL P. Thubert, Ed.
Internet-Draft
Intended status: Standards Track
Expires: May 29, 2015
November 25, 2014

P. Thubert, Ed.
cisco
J. Hui
Cisco

LLN Fragment Forwarding and Recovery
draft-thubert-6lo-forwarding-fragments-02

Abstract

In order to be routed, a fragmented packet must be reassembled at every hop of a multihop link where lower layer fragmentation occurs. Considering that the IPv6 minimum MTU is 1280 bytes and that an an 802.15.4 frame can have a payload limited to 74 bytes in the worst case, a packet might end up fragmented into as many as 18 fragments at the 6LoWPAN shim layer. If a single one of those fragments is lost in transmission, all fragments must be resent, further contributing to the congestion that might have caused the initial packet loss. This draft introduces a simple protocol to forward and recover individual fragments that might be lost over multiple hops between 6LoWPAN endpoints.

Table of Contents

1. Introduction

In most Low Power and Lossy Network (LLN) applications, the bulk of the traffic consists of small chunks of data (in the order few bytes to a few tens of bytes) at a time. Given that an 802.15.4 frame can carry 74 bytes or more in all cases, fragmentation is usually not required. However, and though this happens only occasionally, a number of mission critical applications do require the capability to transfer larger chunks of data, for instance to support a firmware upgrades of the LLN nodes or an extraction of logs from LLN nodes. In the former case, the large chunk of data is transferred to the LLN node, whereas in the latter, the large chunk flows away from the LLN node. In both cases, the size can be on the order of 10K bytes or more and an end-to-end reliable transport is required.

Mechanisms such as TCP or application-layer segmentation will be used to support end-to-end reliable transport. One option to support bulk

data transfer over a frame-size-constrained LLN is to set the Maximum Segment Size to fit within the link maximum frame size. Doing so, however, can add significant header overhead to each 802.15.4 frame. This causes the end-to-end transport to be intimately aware of the delivery properties of the underlaying LLN, which is a layer violation.

An alternative mechanism combines the use of 6LoWPAN fragmentation in addition to transport or application-layer segmentation. Increasing the Maximum Segment Size reduces header overhead by the end-to-end transport protocol. It also encourages the transport protocol to reduce the number of outstanding datagrams, ideally to a single datagram, thus reducing the need to support out-of-order delivery common to LLNs.

[RFC4944] defines a datagram fragmentation mechanism for LLNs. However, because [RFC4944] does not define a mechanism for recovering fragments that are lost, datagram forwarding fails if even one fragment is not delivered properly to the next IP hop. End-to-end transport mechanisms will require retransmission of all fragments, wasting resources in an already resource-constrained network.

Past experience with fragmentation has shown that missassociated or lost fragments can lead to poor network behavior and, eventually, trouble at application layer. The reader is encouraged to read [RFC4963] and follow the references for more information. That experience led to the definition of the Path MTU discovery [RFC1191] protocol that limits fragmentation over the Internet.

For one-hop communications, a number of media propose a local acknowledgment mechanism that is enough to protect the fragments. In a multihop environment, an end-to-end fragment recovery mechanism might be a good complement to a hop-by-hop MAC level recovery. This draft introduces a simple protocol to recover individual fragments between 6LoWPAN endpoints. Specifically in the case of UDP, valuable additional information can be found in UDP Usage Guidelines for Application Designers [RFC5405].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Readers are expected to be familiar with all the terms and concepts that are discussed in "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and

Goals" [RFC4919] and "Transmission of IPv6 Packets over IEEE 802.15.4 Networks" [RFC4944].

ERP

Error Recovery Procedure.

6LoWPAN endpoints

The LLN nodes in charge of generating or expanding a 6LoWPAN header from/to a full IPv6 packet. The 6LoWPAN endpoints are the points where fragmentation and reassembly take place.

3. Rationale

There are a number of uses for large packets in Wireless Sensor Networks. Such usages may not be the most typical or represent the largest amount of traffic over the LLN; however, the associated functionality can be critical enough to justify extra care for ensuring effective transport of large packets across the LLN.

The list of those usages includes:

Towards the LLN node:

Packages of Commands: A number of commands or a full configuration can by packaged as a single message to ensure consistency and enable atomic execution or complete roll back. Until such commands are fully received and interpreted, the intended operation will not take effect.

Firmware update: For example, a new version of the LLN node software is downloaded from a system manager over unicast or multicast services. Such a reflashing operation typically involves updating a large number of similar LLN nodes over a relatively short period of time.

From the LLN node:

Waveform captures: A number of consecutive samples are measured at a high rate for a short time and then transferred from a sensor to a gateway or an edge server as a single large report.

Data logs: LLN nodes may generate large logs of sampled data for later extraction. LLN nodes may also generate system logs to assist in diagnosing problems on the node or network.

Large data packets: Rich data types might require more than one fragment.

Uncontrolled firmware download or waveform upload can easily result in a massive increase of the traffic and saturate the network.

When a fragment is lost in transmission, all fragments are resent, further contributing to the congestion that caused the initial loss, and potentially leading to congestion collapse.

This saturation may lead to excessive radio interference, or random early discard (leaky bucket) in relaying nodes. Additional queuing and memory congestion may result while waiting for a low power next hop to emerge from its sleeping state.

To demonstrate the severity of the problem, consider a fairly reliable 802.15.4 frame delivery rate of 99.9% over a single 802.15.4 hop. The expected delivery rate of a 5-fragment datagram would be about 99.5% over a single 802.15.4 hop. However, the expected delivery rate would drop to 95.1% over 10 hops, a reasonable network diameter for LLN applications. The expected delivery rate for a 1280-byte datagram is 98.4% over a single hop and 85.2% over 10 hops.

Considering that [RFC4944] defines an MTU is 1280 bytes and that in most incarnations (but 802.15.4G) a 802.15.4 frame can limit the MAC payload to as few as 74 bytes, a packet might be fragmented into at least 18 fragments at the 6LoWPAN shim layer. Taking into account the worst-case header overhead for 6LoWPAN Fragmentation and Mesh Addressing headers will increase the number of required fragments to around 32. This level of fragmentation is much higher than that traditionally experienced over the Internet with IPv4 fragments. At the same time, the use of radios increases the probability of transmission loss and Mesh-Under techniques compound that risk over multiple hops.

4. Requirements

This paper proposes a method to recover individual fragments between LLN endpoints. The method is designed to fit the following requirements of a LLN (with or without a Mesh-Under routing protocol):

Number of fragments

The recovery mechanism must support highly fragmented packets, with a maximum of 32 fragments per packet.

Minimum acknowledgment overhead

Because the radio is half duplex, and because of silent time spent in the various medium access mechanisms, an acknowledgment consumes roughly as many resources as data fragment.

The recovery mechanism should be able to acknowledge multiple fragments in a single message and not require an acknowledgment at all if fragments are already protected at a lower layer.

Controlled latency

The recovery mechanism must succeed or give up within the time boundary imposed by the recovery process of the Upper Layer Protocols.

Support for out-of-order fragment delivery

A Mesh-Under load balancing mechanism such as the ISA100 Data Link Layer can introduce out-of-sequence packets.

The recovery mechanism must account for packets that appear lost but are actually only delayed over a different path.

Optional congestion control

The aggregation of multiple concurrent flows may lead to the saturation of the radio network and congestion collapse.

The recovery mechanism should provide means for controlling the number of fragments in transit over the LLN.

5. Overview

Considering that a multi-hop LLN can be a very sensitive environment due to the limited queuing capabilities of a large population of its nodes, this draft recommends a simple and conservative approach to congestion control, based on TCP congestion avoidance.

Congestion on the forward path is assumed in case of packet loss, and packet loss is assumed upon time out. The draft allows to control the number of outstanding fragments, that have been transmitted but for which an acknowledgment was not received yet. It must be noted that the number of outstanding fragments should not exceed the number of hops in the network, but the way to figure the number of hops is out of scope for this document.

Congestion on the forward path can also be indicated by an Explicit Congestion Notification (ECN) mechanism. Though whether and how ECN [RFC3168] is carried out over the LoWPAN is out of scope, this draft

provides a way for the destination endpoint to echo an ECN indication back to the source endpoint in an acknowledgment message as represented in Figure 5 in Section 6.2.

It must be noted that congestion and collision are different topics. In particular, when a mesh operates on a same channel over multiple hops, then the forwarding of a fragment over a certain hop may collide with the forwarding of a next fragment that is following over a previous hop but in a same interference domain. This draft enables an end-to-end flow control, but leaves it to the sender stack to pace individual fragments within a transmit window, so that a given fragment is sent only when the previous fragment has had a chance to progress beyond the interference domain of this hop. In the case of 6TiSCH [I-D.ietf-6tisch-architecture], which operates over the TimeSlotted Channel Hopping [I-D.ietf-6tisch-tsch] (TSCH) mode of operation of IEEE802.14.5, a fragment is forwarded over a different channel at a different time and it make full sense to fire a next fragment as soon as the previous fragment has had its chance to be forwarded at the next hop, retry (ARQ) operations included.

From the standpoint of a source 6LoWPAN endpoint, an outstanding fragment is a fragment that was sent but for which no explicit acknowledgment was received yet. This means that the fragment might be on the way, received but not yet acknowledged, or the acknowledgment might be on the way back. It is also possible that either the fragment or the acknowledgment was lost on the way.

Because a meshed LLN might deliver frames out of order, it is virtually impossible to differentiate these situations. In other words, from the sender standpoint, all outstanding fragments might still be in the network and contribute to its congestion. There is an assumption, though, that after a certain amount of time, a frame is either received or lost, so it is not causing congestion anymore. This amount of time can be estimated based on the round trip delay between the 6LoWPAN endpoints. The method detailed in [RFC6298] is recommended for that computation.

The reader is encouraged to read through "Congestion Control Principles" [RFC2914]. Additionally [RFC2309] and [RFC5681] provide deeper information on why this mechanism is needed and how TCP handles Congestion Control. Basically, the goal here is to manage the amount of fragments present in the network; this is achieved by to reducing the number of outstanding fragments over a congested path by throttling the sources.

Section 7 describes how the sender decides how many fragments are (re)sent before an acknowledgment is required, and how the sender adapts that number to the network conditions.

6. New Dispatch types and headers

This specification extends "Transmission of IPv6 Packets over IEEE 802.15.4 Networks" [RFC4944] with 4 new dispatch types, for Recoverable Fragments (RFRAG) headers with or without Acknowledgment Request, and for the Acknowledgment back, with or without ECN Echo.

```
     Pattern    Header Type
   +------------+------------------------------------------------+
   | 11  101000 | RFRAG      - Recoverable Fragment              |
   | 11  101001 | RFRAG-AR   - RFRAG with Ack Request            |
   | 11  101010 | RFRAG-ACK  - RFRAG Acknowledgment              |
   | 11  101011 | RFRAG-AEC  - RFRAG Ack with ECN Echo           |
   +------------+------------------------------------------------+
```

Figure 1: Additional Dispatch Value Bit Patterns

In the following sections, the semantics of "datagram_tag," "datagram_offset" and "datagram_size" and the reassembly process are changed from [RFC4944] Section 5.3. "Fragmentation Type and Header." The size and offset are expressed on the compressed packet per [RFC6282] as opposed to the uncompressed - native packet - form.

6.1. Recoverable Fragment Dispatch type and Header

```
                        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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |1 1 1 0 1 0 0 X|datagram_offset|         datagram_tag          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |Sequence |     datagram_size    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                             X set == Ack Requested
```

Figure 2: Recoverable Fragment Dispatch type and Header

X: 1 bit; When set, the sender requires an Acknowledgment from the receiver

Sequence: 5 bits; The sequence number of the fragment. Fragments are numbered [0..N] where N is in [0..31].

6.2. Fragment acknowledgment Dispatch type and Header

The specification also defines a 4-octet acknowledgment bitmap that is used to carry selective acknowledgments for the received fragments. A given offset in the bitmap maps one to one with a given sequence number.

The offset of the bit in the bitmap indicates which fragment is acknowledged as follows:

```
                    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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Acknowledgment Bitmap                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 ^                   ^
 |                   |    bitmap indicating whether:
 |                   +--- Fragment with sequence 10 was received
 +----------------------- Fragment with sequence 00 was received
```

Figure 3: Acknowledgment bitmap encoding

So in the example below Figure 4 it appears that all fragments from sequence 0 to 20 were received but for sequence 1, 2 and 16 that were either lost or are still in the network over a slower path.

```
                    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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1|0|0|1|1|1|1|1|1|1|1|1|1|1|1|1|0|1|1|1|1|0|0|0|0|0|0|0|0|0|0|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Expanding 3 octets encoding

The acknowledgment bitmap is carried in a Fragment Acknowledgment as follows:

```
                    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
                +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                |1 1 1 0 1 0 1 Y|         datagram_tag          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Acknowledgment Bitmap (32 bits)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Fragment Acknowledgment Dispatch type and Header

Y: 1 bit; Explicit Congestion Notification (ECN) signalling

When set, the sender indicates that at least one of the acknowledged fragments was received with an Explicit Congestion Notification, indicating that the path followed by the fragments is subject to congestion.

acknowledgment Bitmap

An acknowledgment bitmap, whereby but at offset x indicates that fragment x was received.

7. Fragments Recovery

The Recoverable Fragments header RFRAG and RFRAG-AR deprecate the original fragment headers from [RFC4944] and replace them in the fragmented packets. The Fragment Acknowledgment RFRAG-ACK is introduced as a standalone header in message that is sent back to the fragment source endpoint as known by its MAC address. This assumes that the source MAC address in the fragment (is any) and datagram_tag are enough information to send the Fragment Acknowledgment back to the source fragmentation endpoint.

The 6LoWPAN endpoint that fragments the packets at 6LoWPAN level (the sender) controls the Fragment Acknowledgments. If may do that at any fragment to implement its own policy or perform congestion control which is out of scope for this document. When the sender of the fragment knows that an underlying mechanism protects the Fragments already it MAY refrain from using the Acknowledgment mechanism, and never set the Ack Requested bit. The 6LoWPAN endpoint that recomposes the packets at 6LoWPAN level (the receiver) MUST acknowledge the fragments it has received when asked to, and MAY slightly defer that acknowledgment.

The sender transfers a controlled number of fragments and MAY flag the last fragment of a series with an acknowledgment request. The received MUST acknowledge a fragment with the acknowledgment request bit set. If any fragment immediately preceding an acknowledgment request is still missing, the receiver MAY intentionally delay its acknowledgment to allow in-transit fragments to arrive. delaying the acknowledgment might defeat the round trip delay computation so it should be configurable and not enabled by default.

The receiver interacts with the sender using an Acknowledgment message with a bitmap that indicates which fragments were actually received. The bitmap is a 32bit SWORD, which accommodates up to 32 fragments and is sufficient for the 6LoWPAN MTU. For all n in [0..31], bit n is set to 1 in the bitmap to indicate that fragment with sequence n was received, otherwise the bit is set to 0. All zeros is a NULL bitmap that indicates that the fragmentation process was canceled by the receiver for that datagram.

The receiver MAY issue unsolicited acknowledgments. An unsolicited acknowledgment enables the sender endpoint to resume sending if it had reached its maximum number of outstanding fragments or indicate that the receiver has cancelled the process of an individual datagram. Note that acknowledgments might consume precious resources

so the use of unsolicited acknowledgments should be configurable and not enabled by default.

The sender arms a retry timer to cover the fragment that carries the Acknowledgment request. Upon time out, the sender assumes that all the fragments on the way are received or lost. The process must have completed within an acceptable time that is within the boundaries of upper layer retries. The method detailed in [RFC6298] is recommended for the computation of the retry timer. It is expected that the upper layer retries obey the same or friendly rules in which case a single round of fragment recovery should fit within the upper layer recovery timers.

Fragments are sent in a round robin fashion: the sender sends all the fragments for a first time before it retries any lost fragment; lost fragments are retried in sequence, oldest first. This mechanism enables the receiver to acknowledge fragments that were delayed in the network before they are actually retried.

When the sender decides that a packet should be dropped and the fragmentation process canceled, it sends a pseudo fragment with the datagram_offset, sequence and datagram_size all set to zero, and no data. Upon reception of this message, the receiver should clean up all resources for the packet associated to the datagram_tag. If an acknowledgment is requested, the receiver responds with a NULL bitmap.

The receiver might need to cancel the process of a fragmented packet for internal reasons, for instance if it is out of recomposition buffers, or considers that this packet is already fully recomposed and passed to the upper layer. In that case, the receiver SHOULD indicate so to the sender with a NULL bitmap. Upon an acknowledgment with a NULL bitmap, the sender MUST drop the datagram.

8. Forwarding Fragments

This specification enables intermediate routers to forward fragments with no intermediate reconstruction of the entire packet. Upon the first fragment, the routers lay an label along the path that is followed by that fragment (that is IP routed), and all further fragments are label switched along that path. As a consequence, alternate routes not possible for individual fragments. The datagram tag is used to carry the label, that is swapped at each hop.

8.1. Upon the first fragment

In route over the L2 source changes at each hop. The label that is formed adnd placed in the datagram tag is associated to the source MAC and only valid (and unique) for that source MAC. Say the first fragment has:

Source IPv6 address = IP_A (maybe hops away)

Destination IPv6 address = IP_B (maybe hops away)

Source MAC = MAC_prv (prv as previous)

Datagram_tag= DT_prv

The intermediate router that forwards individual fragments does the following:

a route lookup to get Next hop IPv6 towards IP_B, which resolves as IP_nxt (nxt as next)

a ND resolution to get the MAC address associated to IP_nxt, which resolves as MAC_nxt

Since it is a first fragment of a packet from that source MAC address MAC_prv for that tag DT_prv, the router:

cleans up any leftover resource associated to the tupple (MAC_prv, DT_prv)

allocates a new label for that flow, DT_nxt, from a Least Recently Used pool or some siumilar procedure.

allocates a Label swap structure indexed by (MAC_prv, DT_prv) that contains (MAC_nxt, DT_nxt)

allocates a Label swap structure indexed by (MAC_nxt, DT_nxt) that contains (MAC_prv, DT_prv)

swaps the MAC info to from self to MAC_nxt

Swaps the datagram_tag to DT_nxt

At this point the router is all set and can forward the packet to nxt.

8.2. Upon the next fragments

Upon next fragments (that are not first fragment), the router expects to have already Label swap structure indexed by (MAC_prv, DT_prv). The router:

lookups up the Label swap entry for (MAC_prv, DT_prv), which resolves as (MAC_nxt, DT_nxt)

swaps the MAC info to from self to MAC_nxt;

Swaps the datagram_tag to DT_nxt

At this point the router is all set and can forward the packet to nxt.

if the Label swap entry for (MAC_src, DT_src) is not found, the router builds an RFRAG-ACK to indicate the error. The acknowledgment message has the following information:

MAC info set to from self to MAC_prv as found in the fragment

Swaps the datagram_tag set to DT_prv

Bitmap of all zeroes to indicate the error

At this point the router is all set and can send the RFRAG-ACK back ot the previous router.

8.3. Upon the fragment acknowledgments

Upon fragment acknowledgments next fragments (that are not first fragment), the router expects to have already Label swap structure indexed by (MAC_nxt, DT_nxt). The router:

lookups up the Label swap entry for (MAC_nxt, DT_nxt), which resolves as (MAC_prv, DT_prv)

swaps the MAC info to from self to MAC_prv;

Swaps the datagram_tag to DT_prv

At this point the router is all set and can forward the RFRAG-ACK to prv.

if the Label swap entry for (MAC_nxt, DT_nxt) is not found, it simply drops the packet.

if the RFRAG-ACK indicates either an error or that the fragment was fully receive, the router schedules the Label swap entries for recycling. If the RFRAG-ACK is lost on the way back, the source may retry the last fragments, which will result as an error RFRAG-ACK from the first router on the way that has already cleaned up.

9. Security Considerations

The process of recovering fragments does not appear to create any opening for new threat compared to "Transmission of IPv6 Packets over IEEE 802.15.4 Networks" [RFC4944].

10. IANA Considerations

Need extensions for formats defined in "Transmission of IPv6 Packets over IEEE 802.15.4 Networks" [RFC4944].

11. Acknowledgments

The author wishes to thank Jay Werb, Christos Polyzois, Soumitri Kolavennu, Pat Kinney, Margaret Wasserman, Richard Kelsey, Carsten Bormann and Harry Courtice for their contributions and review.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, September 2007.

[RFC6282] Hui, J. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, September 2011.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

12.2. Informative References

[I-D.ietf-6tisch-architecture]
Thubert, P., Watteyne, T., and R. Assimiti, "An Architecture for IPv6 over the TSCH mode of IEEE 802.15.4e", draft-ietf-6tisch-architecture-04 (work in progress), October 2014.

[I-D.ietf-6tisch-tsch]
Watteyne, T., Palattella, M., and L. Grieco, "Using IEEE802.15.4e TSCH in an IoT context: Overview, Problem Statement and Goals", draft-ietf-6tisch-tsch-03 (work in progress), October 2014.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J., and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.

[RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, August 2007.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, July 2007.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

Authors' Addresses

Pascal Thubert (editor)
Cisco Systems, Inc
Building D
45 Allee des Ormes - BP1200
MOUGINS - Sophia Antipolis 06254
FRANCE

Phone: +33 497 23 26 34
Email: pthubert@cisco.com

Jonathan W. Hui
Cisco Systems
560 McCarthy Blvd.
MILPITAS, California 95035
USA

Email: johui@cisco.com