TSVWG
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2020

M. Welzl
University of Oslo
C. Bormann, Ed.
Universitaet Bremen TZI
March 09, 2020

LOOPS Generic Information Set
draft-welzl-loops-gen-info-03

Abstract

LOOPS (Local Optimizations on Path Segments) aims to provide local (not end-to-end but in-network) recovery of lost packets to achieve better data delivery in the presence of losses. [I-D.li-tsvwg-loops-problem-opportunities] provides an overview over the problems and optimization opportunities that LOOPS could address.

The present document is a strawman for the set of information that would be interchanged in a LOOPS protocol, without already defining a specific data packet format.

The generic information set needs to be mapped to a specific encapsulation protocol to actually run the LOOPS optimizations. The current version of this document contains sketches of bindings to GUE [I-D.ietf-intarea-gue] and Geneve [I-D.ietf-nvo3-geneve].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Today's networks exhibit a wide variety of data rates and, relative to those, processing power and memory capacities of nodes acting as routers. For instance, networks that employ tunneling to build overlay networks may position powerful virtual router nodes in the network to act as tunnel endpoints. The capabilities available in the more powerful cases provide new opportunities for optimizations.

LOOPS (Local Optimizations on Path Segments) aims to provide local (not end-to-end but in-network) recovery of lost packets to achieve better data delivery. [I-D.li-tsvwg-loops-problem-opportunities] provides an overview over the problems and optimization opportunities that LOOPS could address. One simplifying assumption (Section 3) in the present document is that LOOPS segments operate independently from each other, each as a pair of a LOOPS Ingress and a LOOPS Egress node.

The present document is a strawman for the set of information that would be interchanged in a LOOPS protocol between these nodes, without already defining a specific data packet format. The main body of the document defines a mode of the LOOPS protocol that is based on traditional tunneling, the "tunnel mode". Appendix B is an even rougher strawman of a radically different, alternative mode that we call "transparent mode", as well as a slightly more conventional "hybrid mode" (Appendix B.3). These different modes may be applicable to different usage scenarios and will be developed in parallel, with a view of ultimately standardizing one or more of them.

For tunnel mode, the generic information set needs to be mapped to a specific encapsulation protocol to actually run the LOOPS optimizations. LOOPS is not tied to any specific overlay protocol,

but is meant to run embedded into a variety of tunnel protocols. LOOPS information is added as part of a tunnel protocol header at the LOOPS ingress as shown in Figure 1. The current version of this document contains sketches of bindings to GUE [I-D.ietf-intarea-gue] and Geneve [I-D.ietf-nvo3-geneve].

```
          +------------------------------------+
          |           Outer header             |
          +------------------------------------+
        / |         Tunnel Base Header         |
      /   +------------------------------------+\
 Tunnel   |    +-------------------------+     | \
 Header   ~    |    LOOPS Information    |     ~  Tunnel Header
      \   |    +-------------------------+     |  Extensions
        \ +------------------------------------+ /
          |           Data packet              |
          +------------------------------------+
```

Figure 1: Packet in Tunnel with LOOPS Information

Figure 2 is extracted from the LOOPS problems and opportunities document [I-D.li-tsvwg-loops-problem-opportunities]. It illustrates the basic architecture and terms of the applicable scenario of LOOPS. Not all of the concepts introduced in the problems and opportunities document are actually used in the current strawman specification; Section 3 lays out some simplifying assumptions that the present proposal makes.

```
                                                      ON=overlay node
                                                      UN=underlay node

   +---------+                                               +---------+
   |   App   | <---------------- end-to-end ---------------> |   App   |
   +---------+                                               +---------+
   |Transport| <---------------- end-to-end ---------------> |Transport|
   +---------+                                               +---------+
   |         |                                               |         |
   |         |        +--+  path  +--+  path segment2  +--+  |         |
   |         |        |  |<-seg1->|  |<-------------->  |  |  |         |
   | Network |  +--+  |ON|  +--+  |ON|  +--+   +----+  |ON|  | Network |
   |         |--|UN|--|  |--|UN|--|  |--|UN|---| UN |--|  |--|         |
   +---------+  +--+  +--+  +--+  +--+  +--+   +----+  +--+  +---------+
     End Host                                                  End Host
                       <----------------------------------->
                        LOOPS domain: path segments enabling
                        optimization for local in-network recovery
```

Figure 2: LOOPS Usage Scenario

1.1. 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.

This document makes use of the terminology defined in [I-D.li-tsvwg-loops-problem-opportunities]. This section defines additional terminology used by this document.

Data packets: The payload packets that enter and exit a LOOPS segment.

LOOPS Segment: A part of an end-to-end path covered by a single instance of the LOOPS protocol, the sub-path between the LOOPS Ingress and the LOOPS Egress.

LOOPS Ingress: The node that forwards data packets and forward information into the LOOPS segment, potentially performing retransmission and forward error correction based on acknowledgements and measurements received from the LOOPS Egress.

LOOPS Egress: The node that receives the data packets and forward information from the LOOPS ingress, sends acknowledgements and measurements back to the LOOPS ingress (reverse information),

potentially recovers data packets from forward error correction information received.

LOOPS Nodes: Collective term for LOOPS Ingress and LOOPS Egress in a LOOPS Segment.

Forward Information: Information that is added to the stream of data packets in the forward direction by the LOOPS Ingress.

Reverse Information: Information that flows in the reverse direction, from the LOOPS Egress back to the LOOPS Ingress.

Setup Information: Information that is not transferred as part of the Forward or Reverse Information, but is part of the setup of the LOOPS Nodes.

PSN: Packet Sequence Number, a sequence number identifying a data packet.

Sender: Original sender of a packet on an end-to-end path that includes one or more LOOPS segment(s).

Receiver: Ultimate receiver of a packet on an end-to-end path that includes one or more LOOPS segment(s).

2. Challenges

LOOPS has to perform well in the presence of some challenges, which are discussed in this section.

2.1. No Access to End-to-End Transport Information

LOOPS is defined to be independent of the content of the packets being forwarded: there is no dependency on transport-layer or higher information. The intention is to keep LOOPS useful with a traffic mix that may contain encrypted transport protocols such as QUIC as well as encrypted VPN traffic.

2.2. Path Asymmetry

A LOOPS segment is defined as a unidirectional forwarding path. The tunnel might be shared with a LOOPS segment in the inverse direction; this then allows to piggyback Reverse Information on encapsulated packets on that segment. But there is no guarantee that the inverse direction of any end-to-end-path crosses that segment, so the LOOPS optimizations have to be useful on their own in each direction.

2.3. Reordering vs. Spurious Retransmission

The end-to-end transport layer protocol may have its own retransmission mechanism to recover lost packets. When LOOPS recovers a loss, ideally this local recovery would avoid the triggering of a retransmission at the end-to-end sender.

Whether this is possible depends on the specific end-to-end mechanism used for triggering retransmission. When end-to-end retransmission is triggered by receiving a sequence of duplicate acknowledgements (DUPACKs), and with more than a few packets in flight, the recovered packet is likely to be too late to fill the hole in the sequence number space that triggers the DUPACK detection.

(Given a reasonable setting of parameters, the local retransmission will still arrive earlier than the end-to-end retransmission and will possibly unblock application processing earlier; with spurious retransmission detection, there also will be little long-term effect on the send rate.)

The waste of bandwidth caused by a DUPACK-based end-to-end retransmission can be avoided when the end-to-end loss detection is based on time instead of sequence numbers, e.g., with RACK [I-D.ietf-tcpm-rack]. This requires a limit on the additional latency that LOOPS will incur in its attempt to recover the loss locally. In the present version of this document, opportunity to set such a limit is provided in the Setup Information. The limit can be used to compute a deadline for retransmission, but also can be used to choose FEC parameters that keep extra latency low.

2.4. Informing the End-to-End Transport

Congestion control at the end-to-end sender is used to adapt its sending rate to the network congestion status. In typical TCP senders, packet loss implies congestion and leads to a reduction in sending rate. With LOOPS operating, packet loss can be masked from the sender as the loss may have been locally recovered. In this case, rate reduction may not be invoked at the sender. This is a desirable performance improvement if the loss was a random loss.

If LOOPS successfully conceals congestion losses from the end-to-end transport protocol, that might increase the rate to a level that congests the LOOPS segment, or that causes excessive queueing at the LOOPS ingress. What LOOPS should be able to achieve is to let the end host sender invoke the rate reduction mechanism when there is a congestion loss no matter if the lost packet was recovered locally.

As with any tunneling protocol, information about congestion events inside the tunnel needs to be exported to the end-to-end path the tunnel is part of. See e.g., [RFC6040] for a discussion of how to do this in the presence of ECN. A more recent draft, [I-D.ietf-tsvwg-tunnel-congestion-feedback], proposes to activate ECN for the tunnel regardless of whether the end-to-end protocol signals the use of an ECN-capable transport (ECT), which requires more complicated action at the tunnel egress.

A sender that interprets reordering as a signal of packet loss (DUPACKs) initiates a retransmission and reduces the sending rate. When spurious retransmission detection (e.g., via F-RTO [RFC5862] or DSACK [RFC3708]) is enabled by the TCP sender, it will often be able undo the unnecessary window reduction. As LOOPS recovers lost packets locally, in most cases the end host sender will eventually find out its reordering-based retransmission (if any) is spurious. This is an appropriate performance improvement if the loss was a random loss. For congestion losses, a congestion event needs to be signaled to the end-to-end transport.

If the end-to-end transport is ECN-capable (which is visible at the IP level), congestion loss events can easily be signaled to them by setting the CE (congestion experienced) mark. If LOOPS detects a congestion loss for a non-ECT packet, it needs to signal a congestion loss event by introducing a packet loss. This can be done by choosing not to retransmit or repair the packet loss locally in this case. Note that one congestion loss per end-to-end RTT is sufficient to provide the rate reduction, so LOOPS may still be able to recover most packets, in particular for burst losses. (As LOOPS does not interact with the end-to-end transport, it does not know the end-to-end RTT. Some lower bound derived from configuration and measurements could be used instead.)

2.5. Congestion Detection

Properly informing the end-to-end transport protocol about congestion loss events requires distinguishing these from random losses. In some special cases, distinguishing information may be available from a link layer (e.g., see Section 3 of [I-D.li-tsvwg-loops-problem-opportunities]). By enabling ECN inside the tunnel, congestion events experienced at ECN-capable routers will usually be identified by the CE mark, which clearly rules out a random loss.

In the general case, the segment may be composed of hops without such special indications. In these cases, some detection mechanism is required to provide this distinguishing information. The specific mechanism used by an implementation is out of scope of LOOPS, but

LOOPS will need to provide measurement information for this mechanism. For instance, congestion detection might be based on path segment latency information, the proper measurement of which therefore requires special attention in LOOPS.

3. Simplifying assumptions

The above notwithstanding, Implementations may want to make use of indicators such as transport layer port numbers to partition a tunnel flow into separate application flows, e.g., for active queue management (AQM). Any such functionality is orthogonal to the LOOPS protocol itself and thus out of scope for the present document.

One observation that simplifies the design of LOOPS in comparison to that of a reliable transport protocol is that LOOPS does not _have_ to recover every packet loss. Therefore, probabilistic approaches, and simply giving up after some time has elapsed, can simplify the protocol significantly.

For now, we assume that LOOPS segments that may line up on an end-to-end path operate independently of each other. Since the objective of LOOPS ultimately is to assist the end-to-end protocol, it is likely that some cooperation between them would be beneficial, e.g., to obtain some measurements that cover a larger part of the end-to-end path. For instance, cooperating LOOPS segments could try to divide up permissible increases to end-to-end latency between them. This is out of scope for the present version.

Another simplifying assumption is that LOOPS nodes have reasonably precise absolute time available to them, so there is no need to burden the LOOPS protocol with time synchronization. How this is achieved is out of scope.

LOOPS nodes are created and set up (information about their peers, parameters) by some control plane mechanism that is out of scope for this specification. This means there is no need in the LOOPS protocol itself to manage setup information.

4. LOOPS Architecture

From the above, the following architecture is derived for LOOPS.

LOOPS governs the segment from an ingress node to an egress node, which is part of one or more end-to-end paths. Often, a LOOPS segment will operate on aggregate traffic from many such end-to-end paths.

The LOOPS protocol itself does not define how a LOOPS segment and the protocol entities in the ingress and egress node are set up. We expect that a _setup protocol_ on the control plane will provide some _setup information_ to the two nodes, including when to start and to tear down processing.

Each LOOPS segment governs traffic on one direction in the segment. The LOOPS ingress adds _forward information_ to that traffic; the LOOPS egress removes the forward information and sends some _reverse information_ to inform the behavior of the ingress.

Hence, in the data plane, forward information is added to each data packet. Reverse information can be sent in separate packets (e.g., Geneve control-only packets [I-D.ietf-nvo3-geneve]) and/or piggybacked on a related, reverse-direction LOOPS flow, similar to the way the forward information for that flow is carried. The setup protocol is used to provide the relationship between the LOOPS segments in the two directions that is used for piggybacking reverse information.

The above describes the "tunnel mode". A transparent mode is described in Appendix B, which does not modify the data packets and therefore needs to send any forward information (if needed, e.g., for FEC) in separate packets, usually aggregated.

The LOOPS _generic information set_ defines what information is provided as setup information, forward information, and reverse information. _Bindings_ map this information set to specific control plane and data plane protocols, including defining the specific encoding being used. Where separate packets (outside the data plane protocols being used) need to be sent, a special UDP-based protocol needs to be defined as well. The various bindings aim for some commonality, so that an implementation for multiple bindings does not need to support gratuitous variety between them.

5. LOOPS Generic Information Set

This section sketches a generic information set for the LOOPS protocol. Entries marked with (*) are items that may not be necessary and probably should be left out of an initial specification.

5.1. Setup Information

Setup Information might include:

o encapsulation protocol in use, and its vital parameters

- identity of LOOPS ingress and LOOPS egress; information relevant for running the encapsulation protocol such as port numbers

- target maximum latency increase caused by the operation of LOOPS on this segment

- maximum retransmission count (*)

5.2. Forward Information

In the forward information, we have identified:

- tunnel type (a few bits, meaning agreed between Ingress and Egress)

- packet sequence number PSN (20+ bits), counting the data packets forwarded transmitted by the LOOPS ingress (i.e., retransmissions re-use the PSN)

- an "ACK desirable" flag (one bit, usually set for a certain percentage of the data packets only)

- an optional blob, to be echoed by the egress

- anything that the FEC scheme needs.

The first four together (say, 3+24+4+1) might even fit into 32 bits, but probably need up to 48 bits total. FEC info of course often needs more space.

(Note that in this proposal there is no timestamp in the forward information; see Section 6.3.)

24 bits of PSN, minus one bit for sequence number arithmetic, gives 8 million packets (or 2.4 GB at typical packet sizes) per worst-case RTT. So if that is, say, 30 seconds, this would be enough to fill 640 Mbit/s.

5.3. Reverse Information

For the reverse information, we have identified:

- one optional block 1, possibly repeated:

  * PSN being acknowledged

  * absolute time of reception for the packet acknowledged (PSN)

   * the blob, if present, echoed back

- one optional block 2, possibly repeated:

   * an ACK bitmap (based on PSN), always starting at a multiple of 8

   * a delta indicating the end PSN of the bitmap (actually the first PSN that is beyond it), using (Acked-PSN & ~7) + 8*(delta+1) as the end of the bitmap. Acked-PSN in that formula is the previous block 1 PSN seen in this packet, or 0 if none so far.

Block 1 and Block 2 can be interspersed and repeated. They can be piggybacked on a reverse direction data packet or sent separately if none occurs within some timeout. They will usually be aggregated in some useful form. Block 1 information sets are only returned for packets that have "ACK desirable" set. Block 2 information is sent by the receiver based on some saturation scheme (e.g., at least three copies for each PSN span over time). Still, it might be possible to go down to 1 or 2 amortized bytes per forward packet spent for all this.

The latency calculation is done by the sender, who occasionally sets "ACK desirable", and notes down the absolute time of transmission for this data packet (the timekeeping can be done quite efficiently as deltas). Upon reception of a block 1 ACK, it can then subtract that from the absolute time of reception indicated. This assumes time synchronization between the nodes is at least as good as the precision of latency measurement needed, which should be no problem with IEEE 1588 PTP synchronization (but could be if using NTP-based synchronization only). A sender can freely garbage collect noted down transmission time information; doing this too early just means that the quality of the RTT sampling will reduce.

6. LOOPS General Operation

In the Tunnel Mode described in the main body of this document, LOOPS information is carried by some tunnel encapsulation.

6.1. Initial Packet Sequence Number

There is no connection establishment procedure in LOOPS. The initial PSN is assigned unilaterally by the LOOPS Ingress.

Because of the short time that is usually set in the maximum latency increase, there is little damage from a collision of PSNs with packets still in flight from previous instances of LOOPS.

6.1.1. Minimizing collisions

If desired, collisions can be minimized by assigning initial PSNs randomly, or using stable storage. Random assignment is more useful for longer PSNs, where the likelihood of overlap will be low. The specific way a LOOPS ingress uses stable storage is a local matter and thus out of scope. (Implementation note: this can be made to work similar to secure nonce generation with write attenuation: Say, every 10000 packets, the sender notes down the PSN into stable storage. After a reboot, it reloads the PSN and adds 10000 in sequence number arithmetic [RFC1982], plus maybe another 10000 so the sender does not have to wait for the store operation to succeed before sending more packets.)

6.1.2. Optional Initial PSN procedure

As a potential option (to be discussed), an initial packet sequence number could be communicated using a simple two-bit protocol, based on an I flag (Initial PSN) carried in the forward information and an R flag (Initial PSN Received) in the reverse information. This procedure essentially clears the egress of any previous state, however, the benefits of this procedure are limited.

The initial PSN is assigned unilaterally by the LOOPS ingress, selected randomly. The ingress will keep setting the I flag to one when it starts to send packets from a new beginning or whenever it believes there is a need to notify the egress about a new initial PSN. The ingress will stop setting the I flag when it receives an acknowledgement with the R flag set from the egress.

When the LOOPS egress receives a packets with the I flag set, it stops performing services that assume a sequential PSN. The egress will no longer provide acknowledgement information for the packets with PSN smaller than this new initial PSN (per sequence number arithmetic [IEN74]). The egress sends acknowledgement information back without any delay by echoing the value of the I flag in the R flag. This also means the egress unsets the R flag in subsequent acknowledgements for packets with the I flag unset.

It may happen that the first few packets are lost in an initial PSN assignment process. In this case, the loss of these packets is not detectable by the LOOPS ingress since the first received PSN will be treated as an initial PSN at the egress. This is an acceptable temporary performance degradation: LOOPS does not intend to provide perfect reliability, and LOOPS usually applies to the aggregated traffic over a tunnel so that the initial PSN assignment happens infrequently.

6.2. Acknowledgement Generation

A data packet forwarded by the LOOPS ingress always carries PSN information. The LOOPS egress uses the largest newly received PSN with the "ACK desired" bit as the ACK number in the block 1 part of the acknowledgement. This means that the LOOPS ingress gets to modulate the number of acknowledgement sent by the LOOPS egress. However, whenever an out-of-order packet arrives while there still are "holes" in the PSNs received, the LOOPS receiver should generate a block 2 acknowledgement immediately that the LOOPS sender can use as an ACK list.

Reverse information can be piggybacked in a reverse direction data packet. When the reverse direction has no user data to be sent, a pure reverse information packet needs to be generated. This may be based on a short delay during which the LOOPS egress waits for a data packet to piggyback on. (To reduce MTU considerations, the egress could wait for less-than-full data packets.)

6.3. Measurement

When sending a block 1 acknowledgement, the LOOPS egress indicates the absolute time of reception of the packet. The LOOPS ingress can subtract the absolute time of transmission that it still has available, resulting in one high quality latency sample. (In an alternative design, the forward information could include the absolute time of transmission as well, and block1 information would echo it back. This trades memory management at the ingress for increased bandwidth and MTU reduction.)

When a data packet has been transmitted, it may not be clear which specific copy is acknowledged in a block 1 acknowledgement: the acknowledgement for the initial (or, more generally, an earlier) copy may have been delayed (ACK ambiguity)). The LOOPS ingress therefore SHOULD NOT base its measurements on acknowledgements for retransmitted data packets. One way to achieve this is by not setting the "ACK desired" bit on retransmissions in the first place.

The LOOPS ingress can also use the time of reception of the block 1 acknowledgement to obtain a segment RTT sample. Note that this will include any wait time the LOOPS egress incurs while waiting for a piggybacking opportunity -- this is appropriate, as all uses of an RTT will be for keeping a retransmission timeout.

To maintain quality of information during idle times, the LOOPS ingress may send keepalive packets, which are discarded at the LOOPS egress after sending acknowledgements. The indication that a packet is a keepalive packet is dependent on the encapsulation protocol.

6.3.1. Ingress-relative timestamps

As an optional procedure, the ingress node can attach a small blob of data to a forward packet that carries an ACK desired flag; this blob is then echoed by the egress in its block 1 acknowledgement. This is typically used to attach a timestamp on a time scale defined by the ingress; we speak of an ingress-relative timestamp. Alternatively, the ingress can keep a timestamp in its local storage, associated with the PSN of the packet that carries an ACK desired flag; it can then retrieve this timestamp when the block 1 acknowledgement arrives.

In either case, the LOOPS ingress keeps track of the local segment round trip time (LRTT) based on the (saved or received) timestamp and the arrival time of the block 1 acknowledgement, by setting the ACK Desired flag (D flag) occasionally (several times per RTT) and saving/including a sending timestamp for/in the packet.

As the egress will send block 1 acknowledgement information right away when it receives a packet with the D flag set, the measurement of LRTT is more accurate for such packets. A smoothed local segment round trip time S_LRTT can be computed in a similar way as defined by [RFC0793]. A recent minimum value of LRTT is also kept as min_LRTT. S_LRTT is used as a basis for the overall timing of retransmission and state management.

Retransmitted packets MUST NOT be used for local segment round trip time (LRTT) calculation.

6.3.2. ACK generation

A block 1 acknowledgement is generated based on receiving a forward packet with a D flag.

The way block 2 acknowledgement information is sent is more subject to control by the egress. Generally, the egress will aggregate ACK bits for at least K packets before sending a block 2; this can be used to amortize the overhead to close to a couple of bits per ACK. In order to counter loss of reverse information packets, an egress will also want to send an ACK bit more than once -- a saturation value of 3 or more may be chosen based on setup information. Typically, ACK bits already sent will be prepended to ACK bits that are new in this block 2 information set. If K packets do not accumulate for a while, the egress will send one or more packets with block 2 information that covers the unsent ACK bits it has so far.

(Discussion: This works best if the egress has information both about the S_RTT and min_RTT that the ingress uses and the reverse packet loss rate.)

6.4. Loss detection and Recovery

There are two ways for LOOPS local recovery, retransmission and FEC.

6.4.1. Local Retransmission

When retransmission is used as recovery mechanism, the LOOPS ingress detects a packet loss by not receiving an ACK for the packet within the time expected based on an RTO value (which might be calculated as in [RFC6298]). Packet retransmission should then not be performed more than once within an LRTT.

When a retransmission is desired (see Section 2.4 for why it might not be), the LOOPS ingress performs the local in-network recovery by retransmitting the packet. Further retransmissions may be desirable if the lack of ACK is persistent beyond an RTO, as long as the maximum latency increase is not reached.

6.4.2. FEC

FEC is another way to perform local recovery. When FEC is in use, a FEC header is sent with data packets as well as with special repair packets added to the flow. The specific FEC scheme used could be defined in the Setup Information, using a mechanism like [RFC5052]. The FEC rate (amount of redundancy added) and possibly the FEC scheme could be unilaterally adjusted by the LOOPS ingress in an adaptive mechanism based on the measurement information.

6.5. Discussion

Without progress in the way that end-host transport protocols handle reordering, LOOPS will be unable to prevent end-to-end retransmissions that duplicate effort that is spent in local retransmissions. It depends on parameters of the path segment whether this wasted effort is significant or not.

One remedy against this waste could be the introduction of resequencing at the LOOPS Egress node. This increases overall mean packet latency, but does not always increase actual end-to-end data stream latency if a head-of-line blocking transport such as TCP is in use. For applications with a large percentage of legacy TCP end-hosts and sufficient processing capabilities at the LOOPS Egress node, resequencing may be a viable choice. Note that resequencing

could be switched off and on depending on some measurement information.

The packet numbering scheme chosen by LOOPS already provides the necessary information for the LOOPS Egress to reconstruct the sequence of data packets at the LOOPS ingress.

7. Sketches of Bindings to Tunnel Protocols

The LOOPS information defined above in a generic way can be mapped to specific tunnel encapsulation protocols. Sketches for two tunnel protocols are given below: Geneve (Section 7.1), and GUE (Section 7.2). The actual encapsulation can be designed in a "native" way by putting each of the various elements into the TLV format of the encapsulation protocol, or it can be achieved by providing single TLVs for forward and reverse information and using some generic encoding of both kinds of information as shown in Appendix B.3.

7.1. Embedding LOOPS in Geneve

Geneve [I-D.ietf-nvo3-geneve] is an extensible overlay protocol which can embed LOOPS functions. Geneve uses TLVs to carry optional information between NVEs. NVE is logically the same entity as the LOOPS node.

The Geneve header has a mandatory Virtual Network Identifier (VNI) field. The specific VNI value to be used is part of the setup information for the LOOPS tunnel.

More details for a Geneve binding for LOOPS can be found in [I-D.bormann-loops-geneve-binding].

7.2. Embedding LOOPS in GUE

GUE [I-D.ietf-intarea-gue] is an extensible overlay protocol which can embed LOOPS functions. GUE uses flags to indicate the presence of fixed length header extensions. It also allows variable length extensions to be put in "Private data" field. A new LOOPS data block in the "private data" field needs to be defined based on the LOOPS generic information in Section 5.

In the reverse direction, when no data packets are available for piggybacking, LOOPS reverse information is carried in a control message with the C-bit set in the GUE header. The Proto/ctype field contains a control message type when C bit is set. Hence a new control message type should be defined for such LOOPS reverse information.

7.3. Embedding LOOPS in SRv6

An SRv6 binding for LOOPS is proposed in [I-D.wang-loops-srv6-binding].

8. IANA Considerations

No IANA action is required at this stage. When a LOOPS representation is designed for a specific tunneling protocol, new codepoints will be required in the registries that pertain to that protocol.

9. Security Considerations

The security of a specific LOOPS segment will depend both on the properties of the generic information set described here and those of the encapsulation protocol employed. The security considerations of the encapsulation protocol will apply, as will the protection afforded by any security measures provided by the encapsulation protocol. Any LOOPS encapsulation specification is expected to provide information about preferred configurations of the encapsulation protocol employed, including security mechanisms, and to provide a security considerations section discussing the combination. The following discussion aims at discussing security considerations that will be common between different encapsulations.

9.1. Threat model

Attackers might attempt to perturb the operation of a LOOPS segment for a number of purposes:

o Denial of Service: Damaging the ability of LOOPS to recover packets, or damaging packet forwarding through the LOOPS segment in general.

o Attacks on Confidentiality or Integrity: Obtaining the content of data packets, modifying them, injecting new or suppressing specific data packets.

For the purposes of these security considerations, we can distinguish three classes of attackers:

1. on-path read-write: The attacker sees packets under way on the segment and can modify, inject, or suppress them.

   In this case there is really nothing LOOPS can do, except for acting as a full security protocol on its own, which would be the task of the encapsulation protocol. Without that, attackers

already can manipulate the packet stream as they wish. This class of attackers is considered out of scope for these security considerations.

2. on-path read + inject: The attacker sees packets under way on the segment and can inject new packets.

   For this case, LOOPS itself similarly cannot add to the confidentiality of the data stream. However, LOOPS could protect against denial of service against its own protocol operation and, in a limited fashion, against attacks on integrity that wouldn't already have been possible by packet injection without LOOPS.

3. off-path inject: The attacker can inject new packets, but cannot see existing packets under way on the segment.

   Similar considerations apply as for class 2, except that the "blind" class 3 attacker might need to guess information it could have extracted from the packet stream in class 2.

9.2. Discussion

Class 2 attackers can see e.g. sequence numbers and can inject, but not modify traffic. Attacks might include injecting false ACKs, initial PSN flags, ... (TBD)

Class 3 ("blind") attackers might still be able to fake initial PSN bits + false ACKs, but will have a harder time otherwise as it would need to guess the PSN range in which it can wreak havoc. Even random guesses will sometimes hit, though, so the protocol needs to be robust to such injection attacks. ... (TBD)

10. References

10.1. Normative References

[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>.

[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>.

10.2. Informative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC1982] Elz, R. and R. Bush, "Serial Number Arithmetic", RFC 1982, DOI 10.17487/RFC1982, August 1996, <https://www.rfc-editor.org/info/rfc1982>.

[RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, DOI 10.17487/RFC3708, February 2004, <https://www.rfc-editor.org/info/rfc3708>.

[RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, DOI 10.17487/RFC5052, August 2007, <https://www.rfc-editor.org/info/rfc5052>.

[RFC5862] Yasukawa, S. and A. Farrel, "Path Computation Clients (PCC) - Path Computation Element (PCE) Requirements for Point-to-Multipoint MPLS-TE", RFC 5862, DOI 10.17487/RFC5862, June 2010, <https://www.rfc-editor.org/info/rfc5862>.

[RFC6040] Briscoe, B., "Tunnelling of Explicit Congestion Notification", RFC 6040, DOI 10.17487/RFC6040, November 2010, <https://www.rfc-editor.org/info/rfc6040>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC6330] Luby, M., Shokrollahi, A., Watson, M., Stockhammer, T., and L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery", RFC 6330, DOI 10.17487/RFC6330, August 2011, <https://www.rfc-editor.org/info/rfc6330>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[I-D.ietf-tcpm-rack]
Cheng, Y., Cardwell, N., Dukkipati, N., and P. Jha, "RACK: a time-based fast loss detection algorithm for TCP", draft-ietf-tcpm-rack-07 (work in progress), January 2020.

[I-D.ietf-nvo3-geneve]
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-16 (work in progress), March 2020.

[I-D.ietf-intarea-gue]
Herbert, T., Yong, L., and O. Zia, "Generic UDP Encapsulation", draft-ietf-intarea-gue-09 (work in progress), October 2019.

[I-D.li-tsvwg-loops-problem-opportunities]
Yizhou, L., Zhou, X., Boucadair, M., and J. Wang, "LOOPS (Localized Optimizations on Path Segments) Problem Statement and Opportunities for Network-Assisted Performance Enhancement", draft-li-tsvwg-loops-problem-opportunities-04 (work in progress), January 2020.

[I-D.ietf-tsvwg-tunnel-congestion-feedback]
Wei, X., Yizhou, L., Boutros, S., and L. Geng, "Tunnel Congestion Feedback", draft-ietf-tsvwg-tunnel-congestion-feedback-07 (work in progress), May 2019.

[I-D.bormann-loops-geneve-binding]
Bormann, C., "Embedding LOOPS in Geneve", draft-bormann-loops-geneve-binding-00 (work in progress), November 2019.

[I-D.wang-loops-srv6-binding]
Wang, J., Nie, S., Lei, B., and C. Bormann, "Embedding LOOPS in SRv6", draft-wang-loops-srv6-binding-00 (work in progress), March 2020.

[IEN74] Plummer, W., "Sequence Number Arithmetic", Internet Experiment Note 74, September 1978.

Appendix A. Protocol used in Prototype Implementation

This appendix describes, in a somewhat abstracted form, the protocol as used in a prototype implementation, as described by Yizhou Li, and Xingwang Zhou.

The prototype protocol can be run in one of two modes (defined by preconfiguration):

- o Retransmission mode

- o Forward Error Correction (FEC) mode

Forward information is piggybacked in data packets.

Reverse information can be carried in a pure acknowledgement packet or piggybacked when carrying packets for the inverse direction.

The forward information includes:

- o Packet Sequence Number (PSN) (32 bits): This identifies a packet over a specific overlay segment from a specific LOOPS Ingress. If a packet is retransmitted by LOOPS, the retransmission uses the original PSN.

- o Timestamp (32 bits): Information, in a format local to the LOOPS ingress, that provides the time when the packet was sent. In the current implementation, a 32-bit unsigned value specifying the time delta in some granularity from the epoch time to the sending time of the packet carrying this timestamp. The granularity can be from 1 ms to 1 second. The epoch time follows the current TCP practice which is 1 January 1970 00:00:00 UTC. Note that a retransmitted packet uses its own Timestamp.

- o FEC Info for Block Code (56 bits): This header is used in FEC mode. It currently only provides for a block code FEC scheme. It includes the Source Block Number (SBN), Encoding Symbol ID (ESI), number of symbols in a single source block and symbol size. Appendix A.1 gives more details on FEC.

The reverse information includes:

- o ACK Number (32 bits): The largest (in sequence number arithmetic [RFC1982]) PSN received so far.

- o ACK List (variable): This indicates an array of PSN numbers to describe the PSN "holes" preceding the ACK number. It conceptually lists the PSNs of every packet perceived as lost by the LOOPS egress. In actual use, it is truncated.

- o Echoed Timestamp (32 bits): The timestamp received with the packet being acknowledged.

A.1. Block Code FEC

The prototype currently uses a block code FEC scheme (RaptorQ [RFC6330]). The fields in the FEC Info forward information are:

- Source Block Number (SBN): 16 bits. An integer identifier for the source block that the encoding symbols within the packet relate to.

- Encoding Symbol ID (ESI): 16 bits. An integer identifier for the encoding symbols within the packet.

- K: 8 bits. Number of symbols in a single source block.

- T: 16 bits. Symbol size in bytes.

The LOOPS Ingress uses the data packet in Figure 1 to generate the encoding packet. Both source packets and repair packets carry the FEC header information; the LOOPS Egress reconstructs the data packets from both kinds of packets. The LOOPS Egress currently resequences the forwarded and reconstructed packets, so they are passed on in-order when the lost packets are recoverable within the source block.

The LOOPS Nodes need to agree on the use of FEC block mode and on the specific FEC Encoding ID to use; this is currently done by configuration.

Appendix B. Transparent mode

This appendix defines a very different way to provide the LOOPS services, "transparent mode". (We call the protocol described in the main body of the document "encapsulated mode".)

In transparent mode, the idea is that LOOPS does not meddle with the forward transmission of data packets, but runs on the side exchanging additional information.

An implementation could be based on conventional forwarding switches that just provide a copy of the ingress and egress packet stream to the LOOPS implementations. The LOOPS process would occasionally inject recovered packets back into the LOOPS egress node's forwarding switch, see Figure 3.

```
              |
      +-------+-------------------------------------------+
      |       |                                           |
      |  +----+--------+   +-------------------+          |
      |  |    | copy   |   |                   |          |
      |  |    |----------------> LOOPS ingress |          |
      |  |    |        |   |     |     ^       |          |
      |  +----+--------+   +-----|-----|-------+          |
      |   data|packets    forward|     |reverse           |
      |       |              info|     |info              |
      +-------+------------------|-----|------------------+
              |                  |     |
      +-------+------------------|-----|------------------+
      |       |                  |     |                  |
      |  +----+---------+   +----|-----|----------+       |
      |  |    | copy    |   |    v     |          |       |
      |  |    |---------|---|---> LOOPS egress    |       |
      |  |    |         |   |                     |       |
      |  |    |<--------|---|---- inject          |       |
      |  +----+---------+   +---------------------+       |
      |       |                                           |
      +-------+-------------------------------------------+
              |
              v
```

Figure 3: LOOPS Transparent Mode

The obvious advantage of transparent mode is that no encapsulation is needed, reducing processing requirements and keeping the MTU unchanged. The obvious disadvantage is that no forward information can be provided with each data packet, so a replacement needs to be found for the PSN (packet sequence number) employed in encapsulated mode. Any forward information beyond the data packets is sent in separate packets exchanged directly between the LOOPS nodes.

B.1. Packet identification

Retransmission mode and FEC mode differ in their needs for packet identification. For retransmission mode, a somewhat probabilistic accuracy of the packet identification is sufficient, for FEC mode, packet identification should not make mistakes (as these would lead to faultily reconstructed packets).

In Retransmission mode, misidentification of a packet could lead to measurement errors as well as missed retransmission opportunities. The latter will be fixed end-to-end. The tolerance for measurement errors would influence the degree of accuracy that is aimed for.

Packet identification can be based on a cryptographic hash of the packet, computed in LOOPS ingress and egress using the same algorithm (excluding fields that can change in transit, such as TTL/hop limit). The hash can directly be used as a packet number, or it can be sent in the forward information together with a packet sequence number, establishing a mapping.

For probabilistic packet identification, it is almost always sufficient to hash the first few (say, 64) bytes of the packet; all known transport protocols keep sufficient identifying information in that part (and, for encrypted protocols, the entropy will be sufficient). Any collisions of the hash could be used to disqualify the packet for measurement purposes, minimizing the measurement errors; this could allow rather short packet identifiers in retransmission mode.

For FEC mode, the packet identification together with the per-packet FEC information needs to be sent in the (separate) forward information, so that a systematic code can be reconstructed. For retransmission mode, there is no need to send any forward information for most packets, or a mapping from packet identifiers to packet sequence numbers could be sent in the forward information (probably in some aggregated form). The latter would allow keeping the acknowledgement form described in the main body (with aggregate acknowledgement); otherwise, packet identifiers need to be acknowledged. With this change, the LOOPS egress will send reverse information as in the encapsulating LOOPS protocol.

B.2. Generic information and protocol operation

With the changes outlined above, transparent mode operates just as encapsulated mode. If packet sequence numbers are not used, there is no use for block2 reverse information; if they are used, a new block3 needs to be defined that provides the mapping from packet identifiers to packet sequence numbers in the forward information. To avoid MTU reduction, some mechanism will be needed to encapsulate the actual FEC information (additional packets) in the forward information.

B.3. A hybrid mode

Figure 3 can be modified by including a GRE encapsulator into the top left corner and a GRE decapsulator in the bottom left corner. This provides more defined ingress and egress points, but it also provides an opportunity to add a packet sequence number at the ingress. The copies to the top right and bottom right corners are the encapsulated form, i.e., include the sequence number.

The GRE packet header then has the form:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0|0|0|1|    000000000    | 000 |         Protocol Type         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         Sequence Number                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The forward and reverse information can be designed closer to the approach in the main body of the document, to be exchanged using UDP packets between top right ingress and bottom right egress using a port number allocated for this purpose.

Rough ideas for both directions are given below in CDDL [RFC8610]. This information set could be encoded in CBOR or in a bespoke encoding; details such as this can be defined later.

```
forward-information = [
  [rel-psn, ack-desired, ? fec-info] /
  fec-repair-data
]

rel-psn = uint; relative packet sequence number
; always given as a delta from the previous one in the array
; starting out with a "previous value" of 0

ack-desired = bool

fec-info = [
    sbn: uint, ; Source Block Number
    esi: uint, ; Encoding Symbol ID
    ? (
      nsssb: uint; number of symbols in a single source block
      ss: uint; symbol size
    )
]

fec-repair-data = [
    repair-data: bytes
    ? (
      sbn: uint, ; Source Block Number
      esi: uint, ; Encoding Symbol ID
    )
]
```

If left out for a sequence number, the fec-info block is constructed by adding one to the previous one. fec-repair-data contain repair

symbols for the sbn/esi given (which, again, are reconstructed from context if not given).

```
   reverse-information = [
       block1 / block2
   ]

   block1 = [rel-psn, timestamp]
   block2 = [end-psn-delta: uint, acked-bits: bytes]
```

The acked-bits in a block2 is a bitmap that gives acknowledgments for received data packets. The bitmap always comes as a multiple of 8 bits (all bytes are filled in with 8 bits, each identifying a PSN). The end PSN of the bitmap (actually the first PSN that would be beyond it) is computed from the current PSN as set by rel-psn, rounded down to a multiple of 8, and adding 8*(end-psn-delta+1) to that value.

Acknowledgements

Sami Boutros helped with sketching the use of Geneve (Section 7.1), and Tom Herbert helped with sketching the use of GUE (Section 7.2).

Michael Welzl has been supported by the Research Council of Norway under its "Toppforsk" programme through the "OCARINA" project.

Authors' Addresses

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo N-0316
Norway

Phone: +47 22 85 24 20
Email: michawe@ifi.uio.no

Carsten Bormann (editor)
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org