RTGWG   E. Nordmark (ed)
Internet-Draft   Arista Networks
Intended status: Informational   A. Tian
Expires: September 22, 2016   Ericsson Inc.
                                                       J. Gross
                                                         VMware
                                                      J. Hudson
                         Brocade Communications Systems, Inc.
                                                     L. Kreeger
                                            Cisco Systems, Inc.
                                                        P. Garg
                                                      Microsoft
                                                      P. Thaler
                                            Broadcom Corporation
                                                     T. Herbert
                                                       Facebook
                                                 March 21, 2016

                      Encapsulation Considerations
                      draft-ietf-rtgwg-dt-encap-01

Abstract

   The IETF Routing Area director has chartered a design team to look at
   common issues for the different data plane encapsulations being
   discussed in the NVO3 and SFC working groups and also in the BIER
   BoF, and also to look at the relationship between such encapsulations
   in the case that they might be used at the same time. The purpose of
   this design team is to discover, discuss and document considerations
   across the different encapsulations in the different WGs/BoFs so that
   we can reduce the number of wheels that need to be reinvented in the
   future.

Status of This Memo

   This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Table of Contents

1. Design Team Charter

   There have been multiple efforts over the years that have resulted in
   new or modified data plane behaviors involving encapsulations. That
   includes IETF efforts like MPLS, LISP, and TRILL but also industry
   efforts like VXLAN and NVGRE. These collectively can be seen as a
   source of insight into the properties that data planes need to meet.
   The IETF is currently working on potentially new encapsulations in
   NVO3 and SFC and considering working on BIER. In addition there is
   work on tunneling in the INT area.

   This is a short term design team chartered to collect and construct
   useful advice to parties working on new or modified data plane
   behaviors that include additional encapsulations. The goal is for
   the group to document useful advice gathered from interacting with
   ongoing efforts. An Internet Draft will be produced for IETF92 to
   capture that advice, which will be discussed in RTGWG.

   Data plane encapsulations face a set of common issues such as:

   o How to provide entropy for ECMP
   o Issues around packet size and fragmentation/reassembly
   o OAM - what support is needed in an encapsulation format?
   o Security and privacy.
   o QoS
   o Congestion Considerations
   o IPv6 header protection (zero UDP checksum over IPv6 issue)
   o Extensibility - e.g., for evolving OAM, security, and/or
      congestion control
   o Layering of multiple encapsulations e.g., SFC over NVO3 over BIER

   The design team will provide advice on those issues. The intention
   is that even where we have different encapsulations for different
   purposes carrying different information, each such encapsulation
   doesn't have to reinvent the wheel for the above common issues.

   The design team will look across the routing area in particular at
   SFC, NVO3 and BIER. It will not be involved in comparing or
   analyzing any particular encapsulation formats proposed in those WGs
   and BoFs but instead focus on common advice.

2. Overview

   The references provide background information on NVO3, SFC, and BIER.
   In particular, NVO3 is introduced in [RFC7364], [RFC7365], and
   [I-D.ietf-nvo3-arch]. SFC is introduced in
   [I-D.ietf-sfc-architecture] and [I-D.ietf-sfc-problem-statement].
   Finally, the information on BIER is in
   [I-D.shepherd-bier-problem-statement],
   [I-D.wijnands-bier-architecture], and
   [I-D.wijnands-mpls-bier-encapsulation]. We assume the reader has
   some basic familiarity with those proposed encapsulations. The
   Related Work section points at some prior work that relates to the
   encapsulation considerations in this document.

   Encapsulation protocols typically have some unique information that
   they need to carry. In some cases that information might be modified
   along the path and in other cases it is constant. The in-flight
   modifications has impacts on what it means to provide security for
   the encapsulation headers.

   o NVO3 carries a VNI Identifier edge to edge which is not modified.
      There has been OAM discussions in the WG and it isn't clear
      whether some of the OAM information might be modified in flight.
   o SFC carries Service Function Path identification and service meta-
      data. The meta-data might be modified as the packets follow the
      service path. SFC talks of some loop avoidance mechanism which is
      likely to result in modifications for for each hop in the service
      chain even if the meta-data is unmodified.
   o BIER carries a bitmap of egress ports to which a packet should be
      delivered, and as the packet is forwarded down different paths
      different bits are cleared in that bitmap.

   Even if information isn't modified in flight there might be devices
   that wish to inspect that information. For instance, one can
   envision future NVO3 security devices which filter based on the
   virtual network identifier.

   The need for extensibility is different across the protocols

   o NVO3 might need some extensions for OAM and security.
   o SFC consists of Service Function Path identification plus carrying
      service meta-data along a path, and different services might need
      different types and amount of meta-data.
   o BIER might need variable number of bits in their bitmaps, or other
      future schemes to scale up to larger network.

   The extensibility needs and constraints might be different when
   considering hardware vs. software implementations of the

encapsulation headers. NIC hardware might have different constraints
than switch hardware.

As the IETF designs these encapsulations the different WGs solve the
issues for their own encapsulation. But there are likely to be
future cases when the different encapsulations are combined in the
same header. For instance, NVO3 might be a "transport" used to carry
SFC between the different hops in the service chain.

Most of the issues discussed in this document are not new. The IETF
and industry as specified and deployed many different encapsulation
or tunneling protocols over time, ranging from simple IP-in-IP and
GRE encapsulation, IPsec, pseudo-wires, session-based approached like
L2TP, and the use of MPLS control and data planes. IEEE 802 has also
defined layered encapsulation for Provider Backbone Bridges (PBB) and
IEEE 802.1Qbp (ECMP). This document tries to leverage what we
collectively have learned from that experience and summarize what
would be relevant for new encapsulations like NVO3, SFC, and BIER.

3. Common Issues

[This section is mostly a repeat of the charter but with a few
modifications and additions.]

Any new encapsulation protocol would need to address a large set of
issues that are not central to the new information that this protocol
intends to carry. The common issues explored in this document are:

o How to provide entropy for Equal Cost MultiPath (ECMP) routing
o Issues around packet size and fragmentation/reassembly
o Next header indication - each encapsulation might be able to carry
   different payloads
o OAM - what support is needed in an encapsulation format?
o Security and privacy
o QoS
o Congestion Considerations
o Header protection
o Extensibility - e.g., for evolving OAM, security, and/or
   congestion control
o Layering of multiple encapsulations e.g., SFC over NVO3 over BIER
o Importance of being friendly to hardware and software
   implementations

The degree to which these common issues apply to a particular
encapsulation can differ based on the intended purpose of the
encapsulation. But it is useful to understand all of them before
determining which ones apply.

4. Scope

It is important to keep in mind what we are trying to cover and not
cover in this document and effort. This is

o A look across the three new encapsulations, while taking lots of
   previous work into account
o Focus on the class of encapsulations that would run over IP/UDP.
   That was done to avoid being distracted by the data-plane and
   control-plane interaction, which is more significant for protocols
   that are designed to run over "transports" that maintain session
   or path state.
o We later expanded the scope somewhat to consider how the
   encapsulations would play with MPLS "transport", which is
   important because SFC and BIER seem to target being independent of
   the underlying "transport"

However, this document and effort is NOT intended to:

o Design some new encapsulation header to rule them all
o Design yet another new NVO3 encapsulation header
o Try to select the best encapsulation header
o Evaluate any existing and proposed encapsulations

While the origin and focus of this document is the routing area and
in particular NVO3, SFC, and BIER, the considerations apply to other
encapsulations that are being defined in the IETF and elsewhere.
There seems to be an increase in the number of encapsulations being
defined to run over UDP, where there might already exist an
encapsulation over IP or Ethernet. Feedback on how these
considerations apply in those contexts is welcome.

5. Assumptions

The design center for the new encapsulations is a well-managed
network. That network can be a datacenter network (plus datacenter
interconnect) or a service provider network. Based on the existing
and proposed encapsulations in those environment it is reasonable to
make these assumptions:

o The MTU is carefully managed and configured. Hence an
   encapsulation protocol can make the packets bigger without
   resulting in a requirement for fragmentation and reassembly
   between ingress and egress. (However, it might be useful to
   detecting MTU misconfigurations.)
o In general an encapsulation needs some approach for congestion
   management. But the assumptions are different than for arbitrary
   Internet paths in that the underlay might be well-provisioned and

better policed at the edge, and due to multi-tenancy, the
congestion control in the endpoints might be even less trusted
than on the Internet at large.

The goal is to implement these encapsulations in hardware and
software hence we can't assume that the needs of either
implementation approach can trump the needs of the other. In
particular, around extensibility the needs and constraints might be
quite different.

6. Terminology

The capitalized keyword MUST is used as defined in
http://en.wikipedia.org/wiki/Julmust

TBD: Refer to existing documents for at least NVO3 and SFC
terminology. We use at least the VNI ID in this document.

7. Entropy

In many cases the encapsulation format needs to enable ECMP in
unmodified routers. Those routers might use different fields in TCP/
UDP packets to do ECMP without a risk of reordering a flow. Note
that the same entropy might also be used at layer 2 e.g. for Link
Aggregation (LAG).

The common way to do ECMP-enabled encapsulation over IP today is to
add a UDP header and to use UDP with the UDP source port carrying
entropy from the inner/original packet headers as in LISP [RFC6830].
The total entropy consists of 14 bits in the UDP source port (using
the ephemeral port range) plus the outer IP addresses which seems to
be sufficient for entropy; using outer IPv6 headers would give the
option for more entropy should it be needed in the future.

In some environments it might be fine to use all 16 bits of the port
range. However, middleboxes might make assumptions about the system
ports or user ports. But they should not make any assumptions about
the ports in the Dynamic and/or Private Port range, which have the
two MSBs set to 11b.

The UDP source port might change over the lifetime of an encapsulated
flow, for instance for DoS mitigation or re-balancing load across
ECMP. Such changes need to consider reordering if there are packets
in flight for the flow.

There is some interaction between entropy and OAM and extensibility
mechanism. It is desirable to be able to send OAM packets to follow
the same path as network packets. Hence OAM packets should use the

same entropy mechanism as data packets. While routers might use
information in addition the entropy field and outer IP header, they
can not use arbitrary parts of the encapsulation header since that
might result in OAM frames taking a different path. Likewise if
routers look past the encapsulation header they need to be aware of
the extensibility mechanism(s) in the encapsulation format to be able
to find the inner headers in the presence of extensions; OAM frames
might use some extensions e.g. for timestamps.

Architecturally the entropy and the next header field are really part
of enclosing delivery header. UDP with entropy goes hand-in-hand
with the outer IP header. Thus the UDP entropy is present for the
underlay IP routers the same way that an MPLS entropy label is
present for LSRs. The entropy above is all about providing entropy
for the outer delivery of the encapsulated packets.

It has been suggested that when IPv6 is used it would not be
necessary to add a UDP header for entropy, since the IPv6 flow label
can be used for entropy. (This assumes that there is an IP protocol
number for the encapsulation in addition to a UDP destination port
number since UDP would be used with IPv4 underlay. And any use of
UDP checksums would need to be replaced by an encaps-specific
checksum or secure hash.) While such an approach would save 8 bytes
of headers when the underlay is IPv6, it does assume that the
underlay routers use the flow label for ECMP, and it also would make
the IPv6 approach different than the IPv4 approach. Currently the
leaning is towards recommending using the UDP encapsulation for both
IPv4 and IPv6 underlay. The IPv6 flow label can be used for
additional entropy if need be. There is more detailed discussion for
using the IPv6 flow label for tunnels in [RFC6438].

Note that in the proposed BIER encapsulation
[I-D.wijnands-mpls-bier-encapsulation], there is an an 8-bit field
which specifies an entropy value that can be used for load balancing
purposes. This entropy is for the BIER forwarding decisions, which
is independent of any outer delivery ECMP between BIER routers. Thus
it is not part of the delivery ECMP discussed in this section.

   [Note: For any given bit in BIER (that identifies an exit from the
   BIER domain) there might be multiple immediate next hops. The
   BIER entropy field is used to select that next hop as part of BIER
   processing. The BIER forwarding process may do equal cost load
   balancing, but the load balancing procedure MUST choose the same
   path for any two packets that have the same entropy value.]

In summary:

o The entropy is associated with the transport, that is an outer IP
   header or MPLS.
o In the case of IP transport use 14 or 16 bits of UDP source port,
   plus outer IPv6 flowid for entropy.

8. Next-protocol indication

   Next-protocol indications appear in three different contexts for
   encapsulations.

   Firstly, the transport delivery mechanism for the encapsulations we
   discuss in this document need some way to indicate which
   encapsulation header (or other payload) comes next in the packet.
   Some encapsulations might be identified by a UDP port; others might
   be identified by an Ethernet type or IP protocol number. Which
   approach is used is a function of the preceding header the same way
   as IPv4 is identified by both an Ethernet type and an IP protocol
   number (for IP-in-IP). In some cases the header type is implicit in
   some session (L2TP) or path (MPLS) setup. But this is largely beyond
   the control of the encapsulation protocol. For instance, if there is
   a requirement to carry the encapsulation after an Ethernet header,
   then an Ethernet type is needed. If required to be carried after an
   IP/UDP header, then a UDP port number is needed. For UDP port
   numbers there are considerations for port number conservation
   described in [I-D.ietf-tsvwg-port-use].

   It is worth mentioning that in the MPLS case of no implicit protocol
   type many forwarding devices peek at the first nibble of the payload
   to determine whether to apply IPv4 or IPv6 L3/L4 hashes for load
   balancing [RFC7325]. That behavior places some constraints on other
   payloads carried over MPLS and some protocol define an initial
   control word in the payload with a value of zero in its first nibble
   [RFC4385] to avoid confusion with IPv4 and IPv6 payload headers.

   Secondly, the encapsulation needs to indicate the type of its
   payload, which is in scope for the design of the encapsulation. We
   have existing protocols which use Ethernet types (such as GRE). Here
   each encapsulation header can potentially makes its own choices
   between:

   o Use the Ethernet type space - makes it easy to carry existing L2
      and L3 protocols including IPv4, IPv6, and Ethernet.
      Disadvantages are that it is a 16 bit number and we probably need
      far less than 100 values, and the number space is controlled by
      the IEEE 802 RAC with its own allocation policies.
   o Use the IP protocol number space - makes it easy to carry e.g.,
      ESP in addition to IP and Ethernet but brings in all existing
      protocol numbers many of which would never be used directly on top

of the encapsulation protocol. IANA managed eight bit values,
presumably more difficult to get an assigned number than to get a
transport port assignment.

o Define their own next-protocol number space, which can use fewer
   bits than an Ethernet type and give more flexibility, but at the
   cost of administering that numbering space (presumably by the
   IANA).

Thirdly, if the IETF ends up defining multiple encapsulations at
about the same time, and there is some chance that multiple such
encapsulations can be combined in the same packet, there is a
question whether it makes sense to use a common approach and
numbering space for the encapsulation across the different protocols.
A common approach might not be beneficial as long as there is only
one way to indicate e.g., SFC inside NVO3.

Many Internet protocols use fixed values (typically managed by the
IANA function) for their next-protocol field. That facilitates
interpretation of packets by middleboxes and e.g., for debugging
purposes, but might make the protocol evolution inflexible. Our
collective experience with MPLS shows an alternative where the label
can be viewed as an index to a table containing processing
instructions and the table content can be managed in different ways.
Encapsulations might want to consider the tradeoffs between such more
flexible versus more fixed approaches.

In summary:

o Would it be useful for the IETF come up with a common scheme for
   encapsulation protocols? If not each encapsulation can define its
   own scheme.

9. MTU and Fragmentation

A common approach today is to assume that the underlay have
sufficient MTU to carry the encapsulated packets without any
fragmentation and reassembly at the tunnel endpoints. That is
sufficient when the operator of the ingress and egress have full
control of the paths between those endpoints. And it makes for
simpler (hardware) implementations if fragmentation and reassembly
can be avoided.

However, even under that assumption it would be beneficial to be able
to detect when there is some misconfiguration causing packets to be
dropped due to MTU issues. One way to do this is to have the
encapsulator set the don't-fragment (DF) flag in the outer IPv4
header and receive and log any received ICMP "packet too big" (PTB)

errors. Note that no flag needs to be set in an outer IPv6 header
[RFC2460].

Encapsulations could also define an optional tunnel fragmentation and
reassembly mechanism which would be useful in the case when the
operator doesn't have full control of the path, or when the protocol
gets deployed outside of its original intended context. Such a
mechanism would be required if the underlay might have a path MTU
which makes it impossible to carry at least 1518 bytes (if offering
Ethernet service), or at least 1280 (if offering IPv6 service). The
use of such a protocol mechanism could be triggered by receiving a
PTB. But such a mechanism might not be implemented by all
encapsulators and decapsulators. [Aerolink is one example of such a
protocol.]

Depending on the payload carried by the encapsulation there are some
additional possibilities:

o If payload is IPv4/6 then the underlay path MTU could be used to
   report end-to-end path MTU.
o If the payload service is Ethernet/L2, then there is no such per
   destination reporting mechanism. However, there is a LLDP TLV for
   reporting max frame size; might be useful to report minimum to end
   stations, but unmodified end stations would do nothing with that
   TLV since they assume that the MTU is at least 1518.

In summary:

o In some deployments an encapsulation can assume well-managed MTU
   hence no need for fragmentation and reassembly related to the
   encapsulation.
o Even so, it makes sense for ingress to track any ICMP packet too
   big addressed to ingress to be able to log any MTU
   misconfigurations.
o Should an encapsulation protocol be deployed outside of the
   original context it might very well need support for fragmentation
   and reassembly.

10. OAM

The OAM area is seeing active development in the IETF with
discussions (at least) in NVO3 and SFC working groups, plus the new
LIME WG looking at architecture and YANG models.

The design team has take a narrow view of OAM to explore the
potential OAM implications on the encapsulation format.

In terms of what we have heard from the various working groups there
seem to be needs to:

o Be able to send out-of-band OAM messages - that potentially should
   follow the same path through the network as some flow of data
   packets.

   * Such OAM messages should not accidentally be decapsulated and
      forwarded to the end stations.
o Be able to add OAM information to data packets that are
   encapsulated. Discussions have been around:

   * Using a bit in the OAM to synchronize sampling of counters
      between the encapsulator and decapsulator.
   * Optional timestamps, sequence numbers, etc for more detailed
      measurements between encapsulator and decapsulator.
o Usable for both proactive monitoring (akin to BFD) and reactive
   checks (akin to traceroute to pin-point a failure)

To ensure that the OAM messages can follow the same path the OAM
messages need to get the same ECMP (and LAG hashing) results as a
given data flow. An encapsulator can choose between one of:

o Limit ECMP hashing to not look past the UDP header i.e. the
   entropy needs to be in the source/destination IP and UDP ports
o Make OAM packets look the same as data packets i.e. the initial
   part of the OAM payload has the inner Ethernet, IP, TCP/UDP
   headers as a payload. (This approach was taken in TRILL out of
   necessity since there is no UDP header.) Any OAM bit in the
   encapsulation header must in any case be excluded from the
   entropy.

There can be several ways to prevent OAM packets from accidentally
being forwarded to the end station using:

o A bit in the frame (as in TRILL) indicating OAM
o A next-protocol indication with a designated value for "none" or
   "oam".

This assumes that the bit or next protocol, respectively, would not
affect entropy/ECMP in the underlay. However, the next-protocol
field might be used to provide differentiated treatment of packets
based on their payload; for instance a TCP vs. IPsec ESP payload
might be handled differently. Based on that observation it might be
undesirable to overload the next protocol with the OAM drop behavior,
resulting in a preference for having a bit to indicate that the
packet should be forwarded to the end station after decapsulation.

There has been suggestions that one (or more) marker bits in the
encaps header would be useful in order to delineate measurement
epochs on the encapsulator and decapsulator and use that to compare
counters to determine packet loss.

A result of the above is that OAM is likely to evolve and needs some
degree of extensibility from the encapsulation format; a bit or two
plus the ability to define additional larger extensions.

An open question is how to handle error messages or other reports
relating to OAM. One can think if such reporting as being associated
with the encapsulation the same way ICMP is associated with IP.
Would it make sense for the IETF to develop a common Encapsulation
Error Reporting Protocol as part of OAM, which can be used for
different encapsulations? And if so, what are the technical
challenges. For instance, how to avoid it being filtered as ICMP
often is?

A potential additional consideration for OAM is the possible future
existence of gateways that "stitch" together different dataplane
encapsulations and might want to carry OAM end-to-end across the
different encapsulations.

In summary:

o It makes sense to reserve a bit for "drop after decapsulation" for
   OAM out-of-band.
o An encapsulation needs sufficient extensibility for OAM (such as
   bits, timestamps, sequence numbers). That might be motivated by
   in-band OAM but it would make sense to leverage the same
   extensions for out-of band OAM.
o OAM places some constraints on use of entropy in forwarding
   devices.
o Should IETF look into error reporting that is independent of the
   specific encapsulation?

11. Security Considerations

Different encapsulation use cases will have different requirements
around security. For instance, when encapsulation is used to build
overlay networks for network virtualization, isolation between
virtual networks may be paramount. BIER support of multicast may
entail different security requirements than encapsulation for
unicast.

In real deployment, the security of the underlying network may be
considered for determining the level of security needed in the
encapsulation layer. However for the purposes of this discussion, we

assume that network security is out of scope and that the underlying
network does not itself provide adequate or as least uniform security
mechanisms for encapsulation.

There are at least three considerations for security:

o Anti-spoofing/virtual network isolation
o Interaction with packet level security such as IPsec or DTLS
o Privacy (e.g., VNI ID confidentially for NVO3)

This section uses a VNI ID in NVO3 as an example. A SFC or BIER
encapsulation is likely to have fields with similar security and
privacy requirements.

11.1. Encapsulation-specific considerations

Some of these considerations appear for a new encapsulation, and
others are more specific to network virtualization in datacenters.

o New attack vectors:

   * DDOS on specific queued/paths by attempting to reproduce the
      5-tuple hash for targeted connections.
   * Entropy in outer 5-tuple may be too little or predictable.
   * Leakage of identifying information in the encapsulation header
      for an encrypted payload.
   * Vulnerabilities of using global values in fields like VNI ID.
o Trusted versus untrusted tenants in network virtualization:

   * The criticality of virtual network isolation depends on whether
      tenants are trusted or untrusted. In the most extreme cases,
      tenants might not only be untrusted but may be considered
      hostile.
   * For a trusted set of users (e.g. a private cloud) it may be
      sufficient to have just a virtual network identifier to provide
      isolation. Packets inadvertently crossing virtual networks
      should be dropped similar to a TCP packet with a corrupted port
      being received on the wrong connection.
   * In the presence of untrusted users (e.g. a public cloud) the
      virtual network identifier must be adequately protected against
      corruption and verified for integrity. This case may warrant
      keyed integrity.
o Different forms of isolation:

   * Isolation could be blocking all traffic between tenants (or
      except as allowed by some firewall)
   * Could also be about performance isolation i.e. one tenant can
      overload the network in a way that affects other tenants

        * Physical isolation of traffic for different tenants in network
           may be required, as well as required restrictions that tenants
           may have on where their packets may be routed.
   o New attack vectors from untrusted tenants:

        * Third party VMs with untrusted tenants allows internally borne
           attacks within data centers
        * Hostile VMs inside the system may exist (e.g. public cloud)
        * Internally launched DDOS
        * Passive snooping for mis-delivered packets
        * Mitigate damage and detection in event that a VM is able to
           circumvent isolation mechanisms
   o Tenant-provider relationship:

        * Tenant might not trust provider, hypervisors, network
        * Provider likely will need to provide SLA or a least a statement
           on security
        * Tenant may implement their own additional layers of security
        * Regulation and certification considerations
   o Trend towards tighter security:

        * Tenants' data in network increases in volume and value, attacks
           become more sophisticated
        * Large DCs already encrypt everything on disk
        * DCs likely to encrypt inter-DC traffic at this point, use TLS
           to Internet.
        * Encryption within DC is becoming more commonplace, becomes
           ubiquitous when cost is low enough.
        * Cost/performance considerations. Cost of support for strong
           security has made strong network security in DCs prohibitive.
        * Are there lessons from MacSec?

11.2. Virtual network isolation

   The first requirement is isolation between virtual networks. Packets
   sent in one virtual network should never be illegitimately received
   by a node in another virtual network. Isolation should be protected
   in the presence of malicious attacks or inadvertent packet
   corruption.

   The second requirement is sender authentication. Sender identity is
   authenticated to prevent anti-spoofing. Even if an attacker has
   access to the packets in the network, they cannot send packets into a
   virtual network. This may have two possibilities:

   o Pairwise sender authentication. Any two communicating hosts
      negotiate a shared key.

o Group authentication. A group of hosts share a key (this may be
   more appropriate for multicast of encapsulation).

Possible security solutions:

o Security cookie: This is similar to L2TP cookie mechanism
   [RFC3931]. A shared plain text cookie is shared between
   encapsulator and decapsulator. A receiver validates a packet by
   evaluating if the cookie is correct for the virtual network and
   address of a sender. Validation function is F(cookie, VNI ID,
   source address). If cookie matches, accept packet, else drop.
   Since cookie is plain text this method does not protect against an
   eavesdropping. Cookies are set and may be rotated out of band.
o Secure hash: This is a stronger mechanism than simple cookies that
   borrows from IPsec and PPP authentication methods. In this model
   security field contains a secure hash of some fields in the packet
   using a shared key. Hash function may be something like H(key,
   VNI ID, address, salt). The salt ensures the hash is not the same
   for every packet, and if it includes a sequence number may also
   protect against replay attacks.

In any use of a shared key, periodic re-keying should be allowed.
This could include use of techniques like generation numbers, key
windows, etc. See [I-D.farrelll-mpls-opportunistic-encrypt] for an
example application.

We might see firewalls that are aware of the encapsulation and can
provide some defense in depth combined with the above example anti-
spoofing approaches. An example would be an NVO3-aware firewall
being able to check the VNI ID.

Separately and in addition to such filtering, there might be a desire
to completely block an encapsulation protocol at certain places in
the network, e.g., at the edge of a datacenter. Using a fixed
standard UDP destination port number for each encapsulation protocol
would facilitate such blocking.

11.3. Packet level security

An encapsulated packet may itself be encapsulated in IPsec (e.g.
ESP). This should be straightforward and in fact is what would
happen today in security gateways. In this case, there is no special
consideration for the fact that packet is encapsulated, however since
the encapsulation layer headers are included (part of encrypted data
for instance) we lose visibility in the network of the encapsulation.

The more interesting case is when security is applied to the
encapsulation payload. This will keep the encapsulation headers in

the outer header visible to the network (for instance in nvo3 we may
way to firewall based on VNI ID even if the payload is encrypted).
One possibility is to apply DTLS to the encapsulation payload. In
this model the protocol stack may be something like
IP|UDP|Encap|DTLS|encrypted_payload. The encapsulation and security
should be done together at an encapsulator and resolved at the
decapsulator. Since the encapsulation header is outside of the
security coverage, this may itself require security (like described
above).

In both of the above the security associations (SAs) may be between
physical hosts, so for instance in nvo3 we can have packets of
different virtual networks using the same SA-- this should not be an
issue since it is the VNI ID that ensures isolation (which needs to
be secured also).

11.4. In summary:

o Encapsulations need extensibility mechanisms to be able to add
   security features like cookies and secure hashes protecting the
   encapsulation header.
o NVO3 probably has specific higher requirements relating to
   isolation for network virtualization, which is in scope for the
   NVO3 WG.
o Our collective IETF experience is that successful protocols get
   deployed outside of the original intended context, hence the
   initial assumptions about the threat model might become invalid.
   That needs to be considered in the standardization of new
   encapsulations.

12. QoS

In the Internet architecture we support QoS using the Differentiated
Services Code Points (DSCP) in the formerly named Type-of-Service
field in the IPv4 header, and in the Traffic-Class field in the IPv6
header. The ToS and TC fields also contain the two ECN bits, which
are discussed in Section 13.

We have existing specifications how to process those bits. See
[RFC2983] for diffserv handling, which specifies how the received
DSCP value is used to set the DSCP value in an outer IP header when
encapsulating. (There are also existing specifications how DSCP can
be mapped to layer2 priorities.)

Those specifications apply whether or not there is some intervening
headers (e.g., for NVO3 or SFC) between the inner and outer IP
headers. Thus the encapsulation considerations in this area are
mainly about applying the framework in [RFC2983].

Note that the DSCP and ECN bits are not the only part of an inner
packet that might potentially affect the outer packet. For example,
[RFC2473] specifies handling of inner IPv6 hop-by-hop options that
effectively result in copying some options to the outer header. It
is simpler to not have future encapsulations depend on such copying
behavior.

There are some other considerations specific to doing OAM for
encapsulations. If OAM messages are used to measure latency, it
would make sense to treat them the same as data payloads. Thus they
need to have the same outer DSCP value as the data packets which they
wish to measure.

Due to OAM there are constraints on middleboxes in general. If
middleboxes inspect the packet past the outer IP+UDP and
encapsulation header and look for inner IP and TCP/UDP headers, that
might violate the assumption that OAM packets will be handled the
same as regular data packets. That issue is broader than just QoS -
applies to firewall filters etc.

In summary:

o Leverage the existing approach in [RFC2983] for DSCP handling.

13. Congestion Considerations

Additional encapsulation headers does not introduce anything new for
Explicit Congestion Notification. It is just like IP-in-IP and IPsec
tunnels which is specified in [RFC6040] in terms of how the ECN bits
in the inner and outer header are handled when encapsulating and
decapsulating packets. Thus new encapsulations can more or less
include that by reference.

There are additional considerations around carrying non-congestion
controlled traffic. These details have been worked out in
[I-D.ietf-mpls-in-udp]. As specified in [RFC5405]: "IP-based traffic
is generally assumed to be congestion-controlled, i.e., it is assumed
that the transport protocols generating IP-based traffic at the
sender already employ mechanisms that are sufficient to address
congestion on the path. Consequently, a tunnel carrying IP-based
traffic should already interact appropriately with other traffic
sharing the path, and specific congestion control mechanisms for the
tunnel are not necessary". Those considerations are being captured
in [I-D.ietf-tsvwg-rfc5405bis].

For this reason, where an encapsulation method is used to carry IP
traffic that is known to be congestion controlled, the UDP tunnels
does not create an additional need for congestion control. Internet

IP traffic is generally assumed to be congestion-controlled.
Similarly, in general Layer 3 VPNs are carrying IP traffic that is
similarly assumed to be congestion controlled.

However, some of the encapsulations (at least NVO3) will be able to
carry arbitrary Layer 2 packets to provide an L2 service, in which
case one can not assume that the traffic is congestion controlled.

One could handle this by adding some congestion control support to
the encapsulation header (one instance of which would end up looking
like DCCP). However, if the underlay is well-provisioned and managed
as opposed to being arbitrary Internet path, it might be sufficient
to have a slower reaction to congestion induced by that traffic.
There is work underway on a notion of "circuit breakers" for this
purpose. See See [I-D.ietf-tsvwg-circuit-breaker]. Encapsulations
which carry arbitrary Layer 2 packets want to consider that ongoing
work.

If the underlay is provisioned in such a way that it can guarantee
sufficient capacity for non-congestion controlled Layer 2 traffic,
then such circuit breakers might not be needed.

Two other considerations appear in the context of these
encapsulations as applied to overlay networks:

o Protect against malicious end stations
o Ensure fairness and/or measure resource usage across multiple
   tenants

Those issues are really orthogonal to the encapsulation, in that they
are present even when no new encapsulation header is in use.
However, the application of the new encapsulations are likely to be
in environments where those issues are becoming more important.
Hence it makes sense to consider them.

One could make the encapsulation header be extensible to that it can
carry sufficient information to be able to measure resource usage,
delays, and congestion. The suggestions in the OAM section about a
single bit for counter synchronization, and optional timestamps and/
or sequence numbers, could be part of such an approach. There might
also be additional congestion-control extensions to be carried in the
encapsulation. Overall this results in a consideration to support
sufficient extensibility in the encapsulation to handle potential
future developments in this space.

Coarse measurements are likely to suffice, at least for circuit-
breaker-like purposes, see [I-D.wei-tsvwg-tunnel-congestion-feedback]
and [I-D.briscoe-conex-data-centre] for examples on active work in

this area via use of ECN. [RFC6040] Appendix C is also relevant.
The outer ECN bits seem sufficient (at least when everything uses
ECN) to do this course measurements. Needs some more study for the
case when there are also drops; might need to exchange counters
between ingress and egress to handle drops.

Circuit breakers are not sufficient to make a network with different
congestion control when the goal is to provide a predictable service
to different tenants. The fallback would be to rate limit different
traffic.

In summary:

o Leverage the existing approach in [RFC6040] for ECN handling.
o If the encapsulation can carry non-IP, hence non-congestion
   controlled traffic, then leverage the approach in
   [I-D.ietf-mpls-in-udp].
o "Watch this space" for circuit breakers.

14. Header Protection

Many UDP based encapsulations such as VXLAN [RFC7348] either
discourage or explicitly disallow the use of UDP checksums. The
reason is that the UDP checksum covers the entire payload of the
packet and switching ASICs are typically optimized to look at only a
small set of headers as the packet passes through the switch. In
these case, computing a checksum over the packet is very expensive.
(Software endpoints and the NICs used with them generally do not have
the same issue as they need to look at the entire packet anyways.)

The lack a header checksum creates the possibility that bit errors
can be introduced into any information carried by the new headers.
Specifically, in the case of IPv6, the assumption is that a transport
layer checksum - UDP in this case - will protect the IP addresses
through the inclusion of a pseudo-header in the calculation. This is
different from IPv4 on which many of these encapsulation protocols
are initially deployed which contains its own header checksum. In
addition to IP addresses, the encapsulation header often contains its
own information which is used for addressing packets or other high
value network functions. Without a checksum, this information is
potentially vulnerable - an issue regardless of whether the packet is
carried over IPv4 or IPv6.

Several protocols cite [RFC6935] and [RFC6936] as an exemption to the
IPv6 checksum requirements. However, these are intended to be
tailored to a fairly narrow set of circumstances - primarily relying
on sparseness of the address space to detect invalid values and well
managed networks - and are not a one size fits all solution. In

these cases, an analysis should be performed of the intended
environment, including the probability of errors being introduced and
the use of ECC memory in routing equipment.

Conceptually, the ideal solution to this problem is a checksum that
covers only the newly added headers of interest. There is little
value in the portion of the UDP checksum that covers the encapsulated
packet because that would generally be protected by other checksums
and this is the expensive portion to compute. In fact, this solution
already exists in the form of UDP-Lite and UDP based encapsulations
could be easily ported to run on top of it. Unfortunately, the main
value in using UDP as part of the encapsulation header is that it is
recognized by already deployed equipment for the purposes of ECMP,
RSS, and middlebox operations. As UDP-Lite uses a different protocol
number than UDP and it is not widely implemented in middleboxes, this
value is lost. A possible solution is to incorporate the same
partial-checksum concept as UDP-Lite or other header checksum
protection into the encapsulation header and continue using UDP as
the outer protocol. One potential challenge with this approach is
the use of NAT or other form of translation on the outer header will
result in an invalid checksum as the translator will not know to
update the encapsulation header.

The method chosen to protect headers is often related to the security
needs of the encapsulation mechanism. On one hand, the impact of a
poorly protected header is not limited to only data corruption but
can also introduce a security vulnerability in the form of
misdirected packets to an unauthorized recipient. Conversely, high
security protocols that already include a secure hash over the
valuable portion of the header (such as by encrypting the entire IP
packet using IPsec, or some secure hash of the encap header) do not
require additional checksum protection as the hash provides stronger
assurance than a simple checksum.

If the sender has included a checksum, then the receiver should
verify that checksum or, if incapable, drop the packet. The
assumption is that configuration and/or control-plane capability
exchanges can be used when different receiver have different checksum
validation capabilities.

In summary:

o Encapsulations need extensibility to be able to add checksum/CRC
   for the encapsulation header itself.
o When the encapsulation has a checksum/CRC, include the IPv6
   pseudo-header in it.
o The checksum/CRC can potentially be avoided when cryptographic
   protection is applied to the encapsulation.

15. Extensibility Considerations

   Protocol extensibility is the concept that a networking protocol may
   be extended to include new use cases or functionality that were not
   part of the original protocol specification. Extensibility may be
   used to add security, control, management, or performance features to
   a protocol. A solution may allow private extensions for
   customization or experimentation.

   Extending a protocol often implies that a protocol header must carry
   new information. There are two usual methods to accomplish this:

   1. Define or redefine the meaning of existing fields in a protocol
       header.
   2. Add new (optional) fields to the protocol header.

   It is also possible to create a new protocol version, but this is
   more associated with defining a protocol than extending it (IPv6
   being a successor to IPv4 is an example of protocol versioning).

   In some cases it might be more appropriate to define a new inner
   protocol which can carry the new functionality instead of extending
   the outer protocol. Examples where this works well is in the IP/
   transport split, where the earlier architecture had a single NCP
   [RFC0033] protocol which carried both the hop-by-hop semantics which
   are now in IP, and the end-to-end semantics which are now in TCP.
   Such a split is effective when different nodes need to act upon the
   different information. Applying this for general protocol
   extensibility through nesting is not well understood, and does result
   in longer header chains. Furthermore, our experience with IPv6
   extension headers [RFC2460] in middleboxes indicates that the header
   chaining approach does not help with middlebox traversal.

   Many protocol definitions include some number of reserved fields or
   bits which can be used for future extension. VXLAN is an example of
   a protocol that includes reserved bits which are subsequently being
   allocated for new purposes. Another technique employed is to re-
   purpose existing header fields with new meanings. A classic example
   of this is the definition of DSCP code point which redefines the ToS
   field originally specified in IPv4. When a field is redefined, some
   mechanism may be needed to ensure that all interested parties agree
   on the meaning of the field. The techniques of defining meaning for
   reserved bits or redefining existing fields have the advantage that a
   protocol header can be kept a fixed length. The disadvantage is that
   the extensibility is limited. For instance, the number reserved bits
   in a fixed protocol header is limited. For standard protocols the
   decision to commit to a definition for a field can be wrenching since
   it is difficult to retract later. Also, it is difficult to predict a

priori how many reserved fields or bits to put into a protocol header
to satisfy the extensions create over the lifetime of the protocol.

Extending a protocol header with new fields can be done in several
ways.

o TLVs are a very popular method used in such protocols as IP and
   TCP. Depending on the type field size and structure, TLVs can
   offer a virtually unlimited range of extensions. A disadvantage
   of TLVs is that processing them can be verbose, quite complicated,
   several validations must often be done for each TLV, and there is
   no deterministic ordering for a list of TLVs. TCP serves as an
   example of a protocol where TLVs have been successfully used (i.e.
   required for protocol operation). IP is an example of a protocol
   that allows TLVs but are rarely used in practice (router fast
   paths usually that assume no IP options). Note that TCP TLVs are
   implemented in software as well as (NIC) hardware handling various
   forms of TCP offload. Additional discussions about hardware
   implications for extensibility is captured in Section 18.
o Extension headers are closely related to TLVs. These also carry
   type/value information, but instead of being a list of TLVs within
   a single protocol header, each one is in its own protocol header.
   IPv6 extension headers and SFC NSH are examples of this technique.
   Similar to TLVs these offer a wide range of extensibility, but
   have similarly complex processing. Another difference with TLVs
   is that each extension header is idempotent. This is beneficial
   in cases where a protocol implements a push/pop model for header
   elements like service chaining, but makes it more difficult group
   correlated information within one protocol header.
o A particular form of extension headers are the tags used by IEEE
   802 protocols. Those are similar to e.g., IPv6 extension headers
   but with the key difference that each tag is a fixed length header
   where the length is implicit in the tag value. Thus as long as a
   receiver can be programmed with a tag value to length map, it can
   skip those new tags.
o Flag-fields are a non-TLV like method of extending a protocol
   header. The basic idea is that the header contains a set of
   flags, where each set flags corresponds to optional field that is
   present in the header. GRE is an example of a protocol that
   employs this mechanism. The fields are present in the header in
   the order of the flags, and the length of each field is fixed.
   Flag-fields are simpler to process compared to TLVs, having fewer
   validations and the order of the optional fields is deterministic.
   A disadvantage is that range of possible extensions with flag-
   fields is smaller than TLVs.

The requirements for receiving unknown or unimplemented extensible
elements in an encapsulation protocol (flags, TLVs, optional fields)

need to be specified. There are two parties to consider, middle
boxes and terminal endpoints of encapsulation (at the decapsulator).

A protocol may allow or expect nodes in a path to modify fields in an
encapsulation (example use of this is BIER). In this case, the
middleboxes should follow the same requirements as nodes terminating
the encapsulation. In the case that middle boxes do not modify the
encapsulation, we can assume that they may still inspect any fields
of the encapsulation. Missing or unknown fields should be accepted
per protocol specification, however it is permissible for a site to
implement a local policy otherwise (e.g. a firewall may drop packets
with unknown options).

For handling unknown options at terminal nodes, there are two
possibilities: drop packet or accept while ignoring the unknown
options. Many Internet protocols specify that reserved flags must be
set to zero on transmission and ignored on reception. L2TP is
example data protocol that has such flags. GRE is a notable
exception to this rule, reserved flag bits 1-5 cannot be ignored
[RFC2890]. For TCP and IPv4, implementations must ignore optional
TLVs with unknown type; however in IPv6 if a packet contains an
unknown extension header (unrecognized next header type) the packet
must be dropped with an ICMP error message returned. The IPv6
options themselves (encoded inside the destinations options or hop-
by-hop options extension header) have more flexibility. There are
bits in the option code are used to instruct the receiver whether to
ignore, silently drop, or drop and send error if the option is
unknown. Some protocols define a "mandatory bit" that can is set
with TLVs to indicate that an option must not be ignored.
Conceptually, optional data elements can only be ignored if they are
idempotent and do not alter how the rest of the packet is parsed or
processed.

Depending on what type of protocol evolution one can predict, it
might make sense to have a way for a sender to express that the
packet should be dropped by a terminal node which does not understand
the new information. In other cases it would make sense to have the
receiver silently ignore the new info. The former can be expressed
by having a version field in the encapsulation, or a notion of
"mandatory bit" as discussed above.

A security mechanism which use some form secure hash over the
encapsulation header would need to be able to know which extensions
can be changed in flight.

In summary:

   o Encapsulations need the ability to be extended to handle e.g., the
      OAM or security aspects discussed in this document.
   o Practical experience seems to tell us that extensibility
      mechanisms which are not in use on day one might result in
      immediate ossification by lack of implementation support. In some
      cases that has occurred in routers and in other cases in
      middleboxes. Hence devising ways where the extensibility
      mechanisms are in use seems important.

16. Layering Considerations

   One can envision that SFC might use NVO3 as a delivery/transport
   mechanism. With more imagination that in turn might be delivered
   using BIER. Thus it is useful to think about what things look like
   when we have BIER+NVO3+SFC+payload. Also, if NVO3 is widely deployed
   there might be cases of NVO3 nesting where a customer uses NVO3 to
   provide network virtualization e.g., across departments. That
   customer uses a service provider which happens to use NVO3 to provide
   transport for their customers.Thus NVO3 in NVO3 might happen.

   A key question we set out to answer is what the packets might look
   like in such a case, and in particular whether we would end up with
   multiple UDP headers for entropy.

   Based on the discussion in the Entropy section, the entropy is
   associated with the outer delivery IP header. Thus if there are
   multiple IP headers there would be a UDP header for each one of the
   IP headers. But SFC does not require its own IP header. So a case
   of NVO3+SFC would be IP+UDP+NVO3+SFC. A nested NVO3 encapsulation
   would have independent IP+UDP headers.

   The layering also has some implications for middleboxes.

   o A device on the path between the ingress and egress is allowed to
      transparently inspect all layers of the protocol stack and drop or
      forward, but not transparently modify anything but the layer in
      which they operate. What this means is that an IP router is
      allowed modify the outer IP ttl and ECN bits, but not the
      encapsulation header or inner headers and payload. And a BIER
      router is allowed to modify the BIER header.
   o Alternatively such a device can become visible at a higher layer.
      E.g., a middlebox could a middlebox could first decapsulate,
      perform some function then encapsulate; which means it will
      generate a new encapsulation header.

   The design team asked itself some additional questions:

   o Would it make sense to have a common encapsulation base header
      (for OAM, security?, etc) and then followed by the specific
      information for NVO3, SFC, BIER? Given that there are separate
      proposals and the set of information needing to be carried
      differs, and the extensibility needs might be different, it would
      be difficult and not that useful to have a common base header.

   o With a base header in place, one could view the different
      functions (NVO3, SFC, and BIER) as different extensions to that
      base header resulting in encodings which are more space optimal by
      not repeating the same base header. The base header would only be
      repeated when there is an additional IP (and hence UDP) header.
      That could mean a single length field (to skip to get to the
      payload after all the encapsulation headers). That might be
      technically feasible, but it would create a lot of dependencies
      between different WGs making it harder to make progress. Compare
      with the potential savings in packet size.

17. Service model

   The IP service is lossy and subject to reordering. In order to avoid
   a performance impact on transports like TCP the handling of packets
   is designed to avoid reordering packets that are in the same
   transport flow (which is typically identified by the 5-tuple). But
   across such flows the receiver can see different ordering for a given
   sender. That is the case for a unicast vs. a multicast flow from the
   same sender.

   There is a general tussle between the desire for high capacity
   utilization across a multipath network and the impact on packet
   ordering within the same flow (which results in lower transport
   protocol performance). That isn't affected by the introduction of an
   encapsulation. However, the encapsulation comes with some entropy,
   and there might be cases where folks want to change that in response
   to overload or failures. For instance, one might want to change UDP
   source port to try different ECMP route. Such changes can result in
   packet reordering within a flow, hence would need to be done
   infrequently and with care e.g., by identifying packet trains.

   There might be some applications/services which are not able to
   handle reordering across flows. The IETF has defined pseudo-wires
   [RFC3985] which provides the ability to ensure ordering (implemented
   using sequence numbers and/or timestamps).

   Architectural such services would make sense, but as a separate layer
   on top of an encapsulation protocol. They could be deployed between
   ingress and egress of a tunnel which uses some encaps. Potentially
   the tunnel control points at the ingress and egress could become a
   platform for fixing suboptimal behavior elsewhere in the network.

That would clearly be undesirable in the general case. However,
handling encapsulation of non-IP traffic hence non-congestion-
controlled traffic is likely to be required, which implies some
fairness and/or QoS policing on the ingress and egress devices.

But the tunnels could potentially do more like increase reliability
(retransmissions, FEC) or load spreading using e.g. MP-TCP between
ingress and egress.

18. Hardware Friendly

Hosts, switches and routers often leverage capabilities in the
hardware to accelerate packet encapsulation, decapsulation and
forwarding.

Some design considerations in encapsulation that leverage these
hardware capabilities may result in more efficiently packet
processing and higher overall protocol throughput.

While "hardware friendliness" can be viewed as unnecessary
considerations for a design, part of the motivation for considering
this is ease of deployment; being able to leverage existing NIC and
switch chips for at least a useful subset of the functionality that
the new encapsulation provides. The other part is the ease of
implementing new NICs and switch/router chips that support the
encapsulation at ever increasing line rates.

[disclaimer] There are many different types of hardware in any given
network, each maybe better at some tasks while worse at others. We
would still recommend protocol designers to examine the specific
hardware that are likely to be used in their networks and make
decisions on a case by case basis.

Some considerations are:

o Keep the encap header small. Switches and routers usually only
   read the first small number of bytes into the fast memory for
   quick processing and easy manipulation. The bulk of the packets
   are usually stored in slow memory. A big encap header may not fit
   and additional read from the slow memory will hurt the overall
   performance and throughput.
o Put important information at the beginning of the encapsulation
   header. The reasoning is similar as explained in the previous
   point. If important information are located at the beginning of
   the encapsulation header, the packet may be processed with smaller
   number of bytes to be read into the fast memory and improve
   performance.

o Avoid full packet checksums in the encapsulation if possible.
   Encapsulations should instead consider adding their own checksum
   which covers the encapsulation header and any IPv6 pseudo-header.
   The motivation is that most of the switch/router hardware make
   switching/forwarding decisions by reading and examining only the
   first certain number of bytes in the packet. Most of the body of
   the packet do not need to be processed normally. If we are
   concerned of preventing packet to be misdelivered due to memory
   errors, consider only perform header checksums. Note that NIC
   chips can typically already do full packet checksums for TCP/UDP,
   while adding a header checksum might require adding some hardware
   support.

o Place important information at fixed offset in the encapsulation
   header. Packet processing hardware may be capable of parallel
   processing. If important information can be found at fixed
   offset, different part of the encapsulation header may be
   processed by different hardware units in parallel (for example
   multiple table lookups may be launched in parallel). It is easier
   for hardware to handle optional information when the information,
   if present, can be found in ideally one place, but in general, in
   as few places as possible. That facilitates parallel processing.
   TLV encoding with unconstrained order typically does not have that
   property.

o Limit the number of header combinations. In many cases the
   hardware can explore different combinations of headers in
   parallel, however there is some added cost for this.

18.1. Considerations for NIC offload

This section provides guidelines to provide support of common
offloads for encapsulation in Network Interface Cards (NICs).
Offload mechanisms are techniques that are implemented separately
from the normal protocol implementation of a host networking stack
and are intended to optimize or speed up protocol processing.
Hardware offload is performed within a NIC device on behalf of a
host.

There are three basic offload techniques of interest:

o Receive multi queue
o Checksum offload
o Segmentation offload

18.1.1. Receive multi-queue

Contemporary NICs support multiple receive descriptor queues (multi-
queue). Multi-queue enables load balancing of network processing for
a NIC across multiple CPUs. On packet reception, a NIC must select

the appropriate queue for host processing. Receive Side Scaling
(RSS) is a common method which uses the flow hash for a packet to
index an indirection table where each entry stores a queue number.

UDP encapsulation, where the source port is used for entropy, should
be compatible with multi-queue NICs that support five-tuple hash
calculation for UDP/IP packets as input to RSS. The source port
ensures classification of the encapsulated flow even in the case that
the outer source and destination addresses are the same for all flows
(e.g. all flows are going over a single tunnel).

18.1.2. Checksum offload

Many NICs provide capabilities to calculate standard ones complement
payload checksum for packets in transmit or receive. When using
encapsulation over UDP there are at least two checksums that may be
of interest: the encapsulated packet's transport checksum, and the
UDP checksum in the outer header.

18.1.2.1. Transmit checksum offload

NICs may provide a protocol agnostic method to offload transmit
checksum (NETIF_F_HW_CSUM in Linux parlance) that can be used with
UDP encapsulation. In this method the host provides checksum related
parameters in a transmit descriptor for a packet. These parameters
include the starting offset of data to checksum, the length of data
to checksum, and the offset in the packet where the computed checksum
is to be written. The host initializes the checksum field to pseudo
header checksum. In the case of encapsulated packet, the checksum
for an encapsulated transport layer packet, a TCP packet for
instance, can be offloaded by setting the appropriate checksum
parameters.

NICs typically can offload only one transmit checksum per packet, so
simultaneously offloading both an inner transport packet's checksum
and the outer UDP checksum is likely not possible. In this case
setting UDP checksum to zero (per above discussion) and offloading
the inner transport packet checksum might be acceptable.

There is a proposal in [I-D.herbert-remotecsumoffload] to leverage
NIC checksum offload when an encapsulator is co-resident with a host.

18.1.2.2. Receive checksum offload

Protocol encapsulation is compatible with NICs that perform a
protocol agnostic receive checksum (CHECKSUM_COMPLETE in Linux
parlance). In this technique, a NIC computes a ones complement
checksum over all (or some predefined portion) of a packet. The

computed value is provided to the host stack in the packet's receive
descriptor. The host driver can use this checksum to "patch up" and
validate any inner packet transport checksum, as well as the outer
UDP checksum if it is non-zero.

Many legacy NICs don't provide checksum-complete but instead provide
an indication that a checksum has been verified (CHECKSUM_UNNECESSARY
in Linux). Usually, such validation is only done for simple TCP/IP
or UDP/IP packets. If a NIC indicates that a UDP checksum is valid,
the checksum-complete value for the UDP packet is the "not" of the
pseudo header checksum. In this way, checksum-unnecessary can be
converted to checksum-complete. So if the NIC provides checksum-
unnecessary for the outer UDP header in an encapsulation, checksum
conversion can be done so that the checksum-complete value is derived
and can be used by the stack to validate an checksums in the
encapsulated packet.

18.1.3. Segmentation offload

Segmentation offload refers to techniques that attempt to reduce CPU
utilization on hosts by having the transport layers of the stack
operate on large packets. In transmit segmentation offload, a
transport layer creates large packets greater than MTU size (Maximum
Transmission Unit). It is only at much lower point in the stack, or
possibly the NIC, that these large packets are broken up into MTU
sized packet for transmission on the wire. Similarly, in receive
segmentation offload, small packets are coalesced into large, greater
than MTU size packets at a point low in the stack receive path or
possibly in a device. The effect of segmentation offload is that the
number of packets that need to be processed in various layers of the
stack is reduced, and hence CPU utilization is reduced.

18.1.3.1. Transmit Segmentation Offload

Transmit Segmentation Offload (TSO) is a NIC feature where a host
provides a large (larger than MTU size) TCP packet to the NIC, which
in turn splits the packet into separate segments and transmits each
one. This is useful to reduce CPU load on the host.

The process of TSO can be generalized as:

o Split the TCP payload into segments which allow packets with size
   less than or equal to MTU.
o For each created segment:

   1. Replicate the TCP header and all preceding headers of the
       original packet.

2. Set payload length fields in any headers to reflect the length
    of the segment.
3. Set TCP sequence number to correctly reflect the offset of the
    TCP data in the stream.
4. Recompute and set any checksums that either cover the payload
    of the packet or cover header which was changed by setting a
    payload length.

Following this general process, TSO can be extended to support TCP
encapsulation UDP. For each segment the Ethernet, outer IP, UDP
header, encapsulation header, inner IP header if tunneling, and TCP
headers are replicated. Any packet length header fields need to be
set properly (including the length in the outer UDP header), and
checksums need to be set correctly (including the outer UDP checksum
if being used).

To facilitate TSO with encapsulation it is recommended that optional
fields should not contain values that must be updated on a per
segment basis-- for example an encapsulation header should not
include checksums, lengths, or sequence numbers that refer to the
payload. If the encapsulation header does not contain such fields
then the TSO engine only needs to copy the bits in the encapsulation
header when creating each segment and does not need to parse the
encapsulation header.

18.1.3.2. Large Receive Offload

Large Receive Offload (LRO) is a NIC feature where packets of a TCP
connection are reassembled, or coalesced, in the NIC and delivered to
the host as one large packet. This feature can reduce CPU
utilization in the host.

LRO requires significant protocol awareness to be implemented
correctly and is difficult to generalize. Packets in the same flow
need to be unambiguously identified. In the presence of tunnels or
network virtualization, this may require more than a five-tuple match
(for instance packets for flows in two different virtual networks may
have identical five-tuples). Additionally, a NIC needs to perform
validation over packets that are being coalesced, and needs to
fabricate a single meaningful header from all the coalesced packets.

The conservative approach to supporting LRO for encapsulation would
be to assign packets to the same flow only if they have identical
five-tuple and were encapsulated the same way. That is the outer IP
addresses, the outer UDP ports, encapsulated protocol, encapsulation
headers, and inner five tuple are all identical.

18.1.3.3. In summary:

In summary, for NIC offload:

o The considerations for using full UDP checksums are different for
   NIC offload than for implementations in forwarding devices like
   routers and switches.
o Be judicious about encapsulations that change fields on a per-
   packet basis, since such behavior might make it hard to use TSO.

19. Middlebox Considerations

This document has touched upon middleboxes in different section. The
reason for this is as encapsulations get widely deployed one would
expect different forms of middleboxes might become aware of the
encapsulation protocol just as middleboxes have been made aware of
other protocols where there are business and deployment
opportunities. Such middleboxes are likely to do more than just drop
packets based on the UDP port number used by an encapsulation
protocol.

We note that various forms of encapsulation gateways that stitch one
encapsulation protocol together with another form of protocol could
have similar effects.

An example of a middlebox that could see some use would be an
NVO3-aware firewall that would filter on the VNI IDs to provide some
defense in depth inside or across NVO3 datacenters.

A question for the IETF is whether we should document what to do or
what not to do in such middleboxes. This document touches on areas
of OAM and ECMP as it relates to middleboxes and it might make sense
to document how encapsulation-aware middleboxes should avoid
unintended consequences in those (and perhaps other) areas.

In summary:

o We are likely to see middleboxes that at least parse the headers
   for successful new encapsulations.
o Should the IETF document considerations for what not to do in such
   middleboxes?

20. Related Work

The IETF and industry has defined encapsulations for a long time,
with examples like GRE [RFC2890], VXLAN [RFC7348], and NVGRE
[I-D.sridharan-virtualization-nvgre] being able to carry arbitrary
Ethernet payloads, and various forms of IP-in-IP and IPsec

encapsulations that can carry IP packets. As part of NVO3 there has
been additional proposals like Geneve [I-D.gross-geneve] and GUE
[I-D.herbert-gue] which look at more extensibility. NSH
[I-D.quinn-sfc-nsh] is an example of an encapsulation that tries to
provide extensibility mechanisms which target both hardware and
software implementations.

There is also a large body of work around MPLS encapsulations
[RFC3032]. The MPLS-in-UDP work [I-D.ietf-mpls-in-udp] and GRE over
UDP [I-D.ietf-tsvwg-gre-in-udp-encap] have worked on some of the
common issues around checksum and congestion control. MPLS also
introduced a entropy label [RFC6790]. There is also a proposal for
MPLS encryption [I-D.farrelll-mpls-opportunistic-encrypt].

The idea to use a UDP encapsulation with a UDP source port for
entropy for the underlay routers' ECMP dates back to LISP [RFC6830].

The pseudo-wire work [RFC3985] is interesting in the notion of
layering additional services/characteristics such as ordered delivery
or timely deliver on top of an encapsulation. That layering approach
might be useful for the new encapsulations as well. For instance,
the control word [RFC4385]. There is also material on congestion
control for pseudo-wires in [I-D.ietf-pwe3-congcons].

Both MPLS and L2TP [RFC3931] rely on some control or signaling to
establish state (for the path/labels in the case of MPLS, and for the
session in the case of L2TP). The NVO3, SFC, and BIER encapsulations
will also have some separation between the data plane and control
plane, but the type of separation appears to be different.

IEEE 802.1 has defined encapsulations for L2 over L2, in the form of
Provider backbone Bridging (PBB) [IEEE802.1Q-2014] and Equal Cost
Multipath (ECMP) [IEEE802.1Q-2014]. The latter includes something
very similar to the way the UDP source port is used as entropy: "The
flow hash, carried in an F-TAG, serves to distinguish frames
belonging to different flows and can be used in the forwarding
process to distribute frames over equal cost paths"

TRILL, which is also a L2 over L2 encapsulation, took a different
approach to entropy but preserved the ability for OAM frames
[RFC7174] to use the same entropy hence ECMP path as data frames. In
[I-D.ietf-trill-oam-fm] there 96 bytes of headers for entropy in the
OAM frames, followed by the actual OAM content. This ensures that
any headers, which fit in those 96 bytes except the OAM bit in the
TRILL header, can be used for ECMP hashing.

As encapsulations evolve there might be a desire to fit multiple
inner packets into one outer packet. The work in
[I-D.saldana-tsvwg-simplemux] might be interesting for that purpose.

21. Acknowledgements

The authors acknowledge the comments from Alia Atlas, Fred Baker,
David Black, Bob Briscoe, Stewart Bryant, Mike Cox, Andy Malis, Radia
Perlman, Carlos Pignataro, Jamal Hadi Salim, Michael Smith, and Lucy
Yong.

22. Open Issues

o Middleboxes:

   * Due to OAM there are constraints on middleboxes in general. If
      middleboxes inspect the packet past the outer IP+UDP and
      encapsulation header and look for inner IP and TCP/UDP headers,
      that might violate the assumption that OAM packets will be
      handled the same as regular data packets. That issue is
      broader than just QoS - applies to firewall filters etc.
   * Firewalls looking at inner payload? How does that work for OAM
      frames? Even if it only drops ... TRILL approach might be an
      option? Would that encourage more middleboxes making the
      network more fragile?
   * Editorially perhaps we should pull the above two into a
      separate section about middlebox considerations?
o Next-protocol indication - should it be common across different
   encapsulation headers? We will have different ways to indicate
   the presence of the first encapsulation header in a packet (could
   be a UDP destination port, an Ethernet type, etc depending on the
   outer delivery header). But for the next protocol past an
   encapsulation header one could envision creating or adoption a
   common scheme. Such a would also need to be able to identify
   following headers like Ethernet, IPv4/IPv6, ESP, etc.
o Common OAM error reporting protocol?
o There is discussion about timestamps, sequence numbers, etc in
   three different parts of the document. OAM, Congestion
   Considerations, and Service Model, where the latter argues that a
   pseudo-wire service should really be layered on top of the
   encapsulation using its own header. Those recommendations seem to
   be at odds with each other. Do we envision sequence numbers,
   timestamps, etc as potential extensions for OAM and CC? If so,
   those extensions could be used to provide a service which doesn't
   reorder packets.

23. Change Log

The changes from draft-rtg-dt-encap-01 based on feedback at the
Dallas IETF meeting:

o Setting the context that not all common issues might apply to all
   encapsulations, but that they should all be understood before
   being dismissed.
o Clarified that IPv6 flow label is useful for entropy in
   combination with a UDP source port.
o Editorially added a "summary" set of bullets to most sections.
o Editorial clarifications in the next protocol section to more
   clearly state the three areas.
o Folded the two next protocol sections into one.
o Mention the MPLS first nibble issue in the next protocol section.
o Mention that viewing the next protocol as an index to a table with
   processing instructions can provide additional flexibility in the
   protocol evolution.
o For the OAM "don't forward to end stations" added that defining a
   bit seems better than using a special next-protocol value.
o Added mention of DTLS in addition to IPsec for security.
o Added some mention of IPv6 hob-by-hop options of other headers
   than potentially can be copied from inner to outer header.
o Added text on architectural considerations when it might make
   sense to define an additional header/protocol as opposed to using
   the extensibility mechanism in the existing encapsulation
   protocol.
o Clarified the "unconstrained TLVs" in the hardware friendly
   section.
o Clarified the text around checksum verification and full vs.
   header checksums.
o Added wording that the considerations might apply for encaps
   outside of the routing area.
o Added references to draft-ietf-pwe3-congcons, draft-ietf-tsvwg-
   rfc5405bis, RFC2473, and RFC7325
o Removed reference to RFC3948.
o Updated the acknowledgements section.
o Added this change log section.

24. References

24.1. Normative References

[I-D.ietf-tsvwg-rfc5405bis]
          Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage
          Guidelines", draft-ietf-tsvwg-rfc5405bis-10 (work in
          progress), March 2016.

   [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6
              (IPv6) Specification", RFC 2460, DOI 10.17487/RFC2460,
              December 1998, <http://www.rfc-editor.org/info/rfc2460>.

   [RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in
              IPv6 Specification", RFC 2473, DOI 10.17487/RFC2473,
              December 1998, <http://www.rfc-editor.org/info/rfc2473>.

   [RFC2890] Dommety, G., "Key and Sequence Number Extensions to GRE",
              RFC 2890, DOI 10.17487/RFC2890, September 2000,
              <http://www.rfc-editor.org/info/rfc2890>.

   [RFC2983] Black, D., "Differentiated Services and Tunnels",
              RFC 2983, DOI 10.17487/RFC2983, October 2000,
              <http://www.rfc-editor.org/info/rfc2983>.

   [RFC3032] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y.,
              Farinacci, D., Li, T., and A. Conta, "MPLS Label Stack
              Encoding", RFC 3032, DOI 10.17487/RFC3032, January 2001,
              <http://www.rfc-editor.org/info/rfc3032>.

   [RFC3931] Lau, J., Ed., Townsley, M., Ed., and I. Goyret, Ed.,
              "Layer Two Tunneling Protocol - Version 3 (L2TPv3)",
              RFC 3931, DOI 10.17487/RFC3931, March 2005,
              <http://www.rfc-editor.org/info/rfc3931>.

   [RFC3985] Bryant, S., Ed. and P. Pate, Ed., "Pseudo Wire Emulation
              Edge-to-Edge (PWE3) Architecture", RFC 3985,
              DOI 10.17487/RFC3985, March 2005,
              <http://www.rfc-editor.org/info/rfc3985>.

   [RFC4385] Bryant, S., Swallow, G., Martini, L., and D. McPherson,
              "Pseudowire Emulation Edge-to-Edge (PWE3) Control Word for
              Use over an MPLS PSN", RFC 4385, DOI 10.17487/RFC4385,
              February 2006, <http://www.rfc-editor.org/info/rfc4385>.

   [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines
              for Application Designers", BCP 145, RFC 5405,
              DOI 10.17487/RFC5405, November 2008,
              <http://www.rfc-editor.org/info/rfc5405>.

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

   [RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label
              for Equal Cost Multipath Routing and Link Aggregation in
              Tunnels", RFC 6438, DOI 10.17487/RFC6438, November 2011,
              <http://www.rfc-editor.org/info/rfc6438>.

   [RFC6790] Kompella, K., Drake, J., Amante, S., Henderickx, W., and
              L. Yong, "The Use of Entropy Labels in MPLS Forwarding",
              RFC 6790, DOI 10.17487/RFC6790, November 2012,
              <http://www.rfc-editor.org/info/rfc6790>.

   [RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The
              Locator/ID Separation Protocol (LISP)", RFC 6830,
              DOI 10.17487/RFC6830, January 2013,
              <http://www.rfc-editor.org/info/rfc6830>.

   [RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and
              UDP Checksums for Tunneled Packets", RFC 6935,
              DOI 10.17487/RFC6935, April 2013,
              <http://www.rfc-editor.org/info/rfc6935>.

   [RFC6936] Fairhurst, G. and M. Westerlund, "Applicability Statement
              for the Use of IPv6 UDP Datagrams with Zero Checksums",
              RFC 6936, DOI 10.17487/RFC6936, April 2013,
              <http://www.rfc-editor.org/info/rfc6936>.

   [RFC7174] Salam, S., Senevirathne, T., Aldrin, S., and D. Eastlake
              3rd, "Transparent Interconnection of Lots of Links (TRILL)
              Operations, Administration, and Maintenance (OAM)
              Framework", RFC 7174, DOI 10.17487/RFC7174, May 2014,
              <http://www.rfc-editor.org/info/rfc7174>.

   [RFC7325] Villamizar, C., Ed., Kompella, K., Amante, S., Malis, A.,
              and C. Pignataro, "MPLS Forwarding Compliance and
              Performance Requirements", RFC 7325, DOI 10.17487/RFC7325,
              August 2014, <http://www.rfc-editor.org/info/rfc7325>.

   [RFC7348] Mahalingam, M., Dutt, D., Duda, K., Agarwal, P., Kreeger,
              L., Sridhar, T., Bursell, M., and C. Wright, "Virtual
              eXtensible Local Area Network (VXLAN): A Framework for
              Overlaying Virtualized Layer 2 Networks over Layer 3
              Networks", RFC 7348, DOI 10.17487/RFC7348, August 2014,
              <http://www.rfc-editor.org/info/rfc7348>.

   [RFC7364] Narten, T., Ed., Gray, E., Ed., Black, D., Fang, L.,
              Kreeger, L., and M. Napierala, "Problem Statement:
              Overlays for Network Virtualization", RFC 7364,
              DOI 10.17487/RFC7364, October 2014,
              <http://www.rfc-editor.org/info/rfc7364>.

   [RFC7365] Lasserre, M., Balus, F., Morin, T., Bitar, N., and Y.
              Rekhter, "Framework for Data Center (DC) Network
              Virtualization", RFC 7365, DOI 10.17487/RFC7365, October
              2014, <http://www.rfc-editor.org/info/rfc7365>.

24.2. Informative References

   [I-D.briscoe-conex-data-centre]
              Briscoe, B. and M. Sridharan, "Network Performance
              Isolation in Data Centres using Congestion Policing",
              draft-briscoe-conex-data-centre-02 (work in progress),
              February 2014.

   [I-D.farrelll-mpls-opportunistic-encrypt]
              Farrel, A. and S. Farrell, "Opportunistic Security in MPLS
              Networks", draft-farrelll-mpls-opportunistic-encrypt-05
              (work in progress), June 2015.

   [I-D.gross-geneve]
              Gross, J., Sridhar, T., Garg, P., Wright, C., Ganga, I.,
              Agarwal, P., Duda, K., Dutt, D., and J. Hudson, "Geneve:
              Generic Network Virtualization Encapsulation", draft-
              gross-geneve-02 (work in progress), October 2014.

   [I-D.herbert-gue]
              Herbert, T., Yong, L., and O. Zia, "Generic UDP
              Encapsulation", draft-herbert-gue-03 (work in progress),
              March 2015.

   [I-D.herbert-remotecsumoffload]
              Herbert, T., "Remote checksum offload for encapsulation",
              draft-herbert-remotecsumoffload-02 (work in progress),
              March 2016.

   [I-D.ietf-mpls-in-udp]
              Xu, X., Sheth, N., Yong, L., Callon, R., and D. Black,
              "Encapsulating MPLS in UDP", draft-ietf-mpls-in-udp-11
              (work in progress), January 2015.

   [I-D.ietf-nvo3-arch]
              Black, D., Hudson, J., Kreeger, L., Lasserre, M., and T.
              Narten, "An Architecture for Overlay Networks (NVO3)",
              draft-ietf-nvo3-arch-04 (work in progress), October 2015.

   [I-D.ietf-pwe3-congcons]
              Stein, Y., Black, D., and B. Briscoe, "Pseudowire
              Congestion Considerations", draft-ietf-pwe3-congcons-02
              (work in progress), July 2014.

   [I-D.ietf-sfc-architecture]
             Halpern, J. and C. Pignataro, "Service Function Chaining
             (SFC) Architecture", draft-ietf-sfc-architecture-11 (work
             in progress), July 2015.

   [I-D.ietf-sfc-problem-statement]
             Quinn, P. and T. Nadeau, "Service Function Chaining
             Problem Statement", draft-ietf-sfc-problem-statement-13
             (work in progress), February 2015.

   [I-D.ietf-trill-oam-fm]
             Senevirathne, T., Finn, N., Salam, S., Kumar, D.,
             Eastlake, D., Aldrin, S., and L. Yizhou, "TRILL Fault
             Management", draft-ietf-trill-oam-fm-11 (work in
             progress), October 2014.

   [I-D.ietf-tsvwg-circuit-breaker]
             Fairhurst, G., "Network Transport Circuit Breakers",
             draft-ietf-tsvwg-circuit-breaker-13 (work in progress),
             February 2016.

   [I-D.ietf-tsvwg-gre-in-udp-encap]
             Yong, L., Crabbe, E., Xu, X., and T. Herbert, "GRE-in-UDP
             Encapsulation", draft-ietf-tsvwg-gre-in-udp-encap-11 (work
             in progress), March 2016.

   [I-D.ietf-tsvwg-port-use]
             Touch, J., "Recommendations on Using Assigned Transport
             Port Numbers", draft-ietf-tsvwg-port-use-11 (work in
             progress), April 2015.

   [I-D.quinn-sfc-nsh]
             Quinn, P., Guichard, J., Surendra, S., Smith, M.,
             Henderickx, W., Nadeau, T., Agarwal, P., Manur, R.,
             Chauhan, A., Halpern, J., Majee, S., Elzur, U., Melman,
             D., Garg, P., McConnell, B., Wright, C., and K. Kevin,
             "Network Service Header", draft-quinn-sfc-nsh-07 (work in
             progress), February 2015.

   [I-D.saldana-tsvwg-simplemux]
             Saldana, J., "Simplemux. A generic multiplexing protocol",
             draft-saldana-tsvwg-simplemux-04 (work in progress),
             January 2016.

   [I-D.shepherd-bier-problem-statement]
              Shepherd, G., Dolganow, A., and a.
              arkadiy.gulko@thomsonreuters.com, "Bit Indexed Explicit
              Replication (BIER) Problem Statement", draft-shepherd-
              bier-problem-statement-02 (work in progress), February
              2015.

   [I-D.sridharan-virtualization-nvgre]
              Garg, P. and Y. Wang, "NVGRE: Network Virtualization using
              Generic Routing Encapsulation", draft-sridharan-
              virtualization-nvgre-08 (work in progress), April 2015.

   [I-D.wei-tsvwg-tunnel-congestion-feedback]
              Wei, X., Zhu, L., Deng, L., and B. Briscoe, "Tunnel
              Congestion Feedback", draft-wei-tsvwg-tunnel-congestion-
              feedback-04 (work in progress), June 2015.

   [I-D.wijnands-bier-architecture]
              Wijnands, I., Rosen, E., Dolganow, A., Przygienda, T., and
              S. Aldrin, "Multicast using Bit Index Explicit
              Replication", draft-wijnands-bier-architecture-05 (work in
              progress), March 2015.

   [I-D.wijnands-mpls-bier-encapsulation]
              Wijnands, I., Rosen, E., Dolganow, A., Tantsura, J., and
              S. Aldrin, "Encapsulation for Bit Index Explicit
              Replication in MPLS Networks", draft-wijnands-mpls-bier-
              encapsulation-02 (work in progress), December 2014.

   [I-D.xu-bier-encapsulation]
              Xu, X., Somasundaram, S., Jacquenet, C., and R. Raszuk,
              "BIER Encapsulation", draft-xu-bier-encapsulation-03 (work
              in progress), October 2015.

   [IEEE802.1Q-2014]
              IEEE, "IEEE Standard for Local and metropolitan area
              networks--Bridges and Bridged Networks", IEEE Std 802.1Q-
              2014, 2014,
              <http://www.ieee802.org/1/pages/802.1Q-2014.html>.

              (Access Controlled link within page)

   [RFC0033] Crocker, S., "New Host-Host Protocol", RFC 33,
              DOI 10.17487/RFC0033, February 1970,
              <http://www.rfc-editor.org/info/rfc33>.

Authors' Addresses

   Erik Nordmark
   Arista Networks
   5453 Great America Parkway
   Santa Clara, CA 95054
   USA

   Email: nordmark@arista.com

   Albert Tian
   Ericsson Inc.
   300 Holger Way
   San Jose, California 95134
   USA

   Email: albert.tian@ericsson.com

   Jesse Gross
   VMware
   3401 Hillview Ave.
   Palo Alto, CA 94304
   USA

   Email: jgross@vmware.com

   Jon Hudson
   Brocade Communications Systems, Inc.
   130 Holger Way
   San Jose, CA 95134
   USA

   Email: jon.hudson@gmail.com

   Lawrence Kreeger
   Cisco Systems, Inc.
   170 W. Tasman Avenue
   San Jose, CA 95134
   USA

   Email: kreeger@cisco.com

Pankaj Garg
Microsoft
1 Microsoft Way
Redmond, WA 98052
USA

Email: pankajg@microsoft.com

Patricia Thaler
Broadcom Corporation
3151 Zanker Road
San Jose, CA 95134
USA

Email: pthaler@broadcom.com

Tom Herbert
Facebook
1 Hacker Way
Menlo Park, CA 94052
USA

Email: tom@herbertland.com