Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: July 25, 2013

G. Fairhurst
University of Aberdeen
M. Westerlund
Ericsson

January 21, 2013

Applicability Statement for the use of IPv6 UDP Datagrams with Zero Checksums
draft-ietf-6man-udpzero-10

Abstract

This document provides an applicability statement for the use of UDP transport checksums with IPv6. It defines recommendations and requirements for the use of IPv6 UDP datagrams with a zero UDP checksum. It describes the issues and design principles that need to be considered when UDP is used with IPv6 to support tunnel encapsulations and examines the role of the IPv6 UDP transport checksum. An appendix presents a summary of the trade-offs that were considered in evaluating the safety of the update to RFC 2460 that updates use of the UDP checksum with IPv6.

Status of this Memo

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

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

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

This Internet-Draft will expire on July 25, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

The User Datagram Protocol (UDP) [RFC0768] transport is defined for the Internet Protocol (IPv4) [RFC0791] and is defined in "Internet Protocol, Version 6 (IPv6) [RFC2460] for IPv6 hosts and routers. The UDP transport protocol has a minimal set of features. This limited set has enabled a wide range of applications to use UDP, but these application do need to provide many important transport functions on top of UDP. The UDP Usage Guidelines [RFC5405] provides overall guidance for application designers, including the use of UDP to support tunneling. The key difference between UDP usage with IPv4 and IPv6 is that RFC 2460 mandates use of a calculated UDP checksum, i.e. a non-zero value, due to the lack of an IPv6 header checksum. Algorithms for checksum computation are described in [RFC1071].

The lack of a possibility to use an IPv6 datagram with a zero UDP checksum has been observed as a real problem for certain classes of application, primarily tunnel applications. This class of application has been deployed with a zero UDP checksum using IPv4. The design of IPv6 raises different issues when considering the safety of using a UDP checksum with IPv6. These issues can significantly affect applications, both when an endpoint is the intended user and when an innocent bystander (when a packet is received by a different endpoint to that intended).

This document examines the issues and an appendix compares the strengths and weaknesses of a number of proposed solutions. This identifies a set of issues that must be considered and mitigated to be able to safely deploy IPv6 applications that use a zero UDP checksum. The provided comparison of methods is expected to also be useful when considering applications that have different goals from the ones that initiated the writing of this document, especially the use of already standardized methods. The analysis concludes that using a zero UDP checksum is the best method of the proposed alternatives to meet the goals for certain tunnel applications.

This document defines recommendations and requirements for use of IPv6 datagrams with a zero UDP checksum. This usage is expected to have initial deployment issues related to middleboxes, limiting the usability more than desired in the currently deployed Internet. However, this limitation will be largest initially and will reduce as updates are provided in middleboxes that support the zero UDP checksum for IPv6. The document therefore derives a set of constraints required to ensure safe deployment of a zero UDP checksum.

Finally, the document also identifies some issues that require future consideration and possibly additional research.

1.1. Document Structure

Section 1 provides a background to key issues, and introduces the use of UDP as a tunnel transport protocol.

Section 2 describes a set of standards-track datagram transport protocols that may be used to support tunnels.

Section 3 discusses issues with a zero UDP checksum for IPv6. It considers the impact of corruption, the need for validation of the path and when it is suitable to use a zero UDP checksum.

Section 4 is an applicability statement that defines requirements and recommendations on the implementation of IPv6 nodes that support the use of a zero UDP checksum.

Section 5 provides an applicability statement that defines requirements and recommendations for protocols and tunnel encapsulations that are transported over an IPv6 transport that does not perform a UDP checksum calculation to verify the integrity at the transport endpoints.

Section 6 provides the recommendations for standardization of zero UDP checksum with a summary of the findings and notes remaining issues needing future work.

Appendix A evaluates the set of proposals to update the UDP transport behaviour and other alternatives intended to improve support for tunnel protocols. It concludes by assessing the trade-offs of the various methods, identifying advantages and disadvantages for each method.

1.2. Terminology

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

1.3. Use of UDP Tunnels

One increasingly popular use of UDP is as a tunneling protocol, where a tunnel endpoint encapsulates the packets of another protocol inside UDP datagrams and transmits them to another tunnel endpoint. Using UDP as a tunneling protocol is attractive when the payload protocol is not supported by the middleboxes that may exist along the path, because many middleboxes support transmission using UDP. In this use, the receiving endpoint decapsulates the UDP datagrams and forwards the original packets contained in the payload [RFC5405].

Tunnels establish virtual links that appear to directly connect locations that are distant in the physical Internet topology and can be used to create virtual (private) networks.

1.3.1. Motivation for new approaches

A number of tunnel encapsulations deployed over IPv4 have used the UDP transport with a zero checksum. Users of these protocols expect a similar solution for IPv6.

A number of tunnel protocols are also currently being defined (e.g. Automated Multicast Tunnels, AMT [I-D.ietf-mboned-auto-multicast], and the Locator/Identifier Separation Protocol, LISP [LISP]). These protocols motivated an update to IPv6 UDP checksum processing to benefit from simpler checksum processing for various reasons:

- Reducing forwarding costs, motivated by redundancy present in the encapsulated packet header, since in tunnel encapsulations, payload integrity and length verification may be provided by higher layer encapsulations (often using the IPv4, UDP, UDP-Lite, or TCP checksums).

- Eliminating a need to access the entire packet when forwarding the packet by a tunnel endpoint.

- Enhancing ability to traverse middleboxes, especially Network Address Translators, NATs.

- A desire to use the port number space to enable load-sharing.

1.3.2. Reducing forwarding cost

It is a common requirement to terminate a large number of tunnels on a single router/host. The processing cost per tunnel includes both state (memory requirements) and per-packet processing.

Automatic IP Multicast Tunneling, known as AMT [I-D.ietf-mboned-auto-multicast] currently specifies UDP as the transport protocol for packets carrying tunneled IP multicast packets. The current specification for AMT requires that the UDP checksum in the outer packet header should be zero (see Section 6.6 of [I-D.ietf-mboned-auto-multicast]). This argues that the computation of an additional checksum is an unwarranted burden on nodes implementing lightweight tunneling protocols when an inner packet is already adequately protected, . The AMT protocol needs to replicate a multicast packet to each gateway tunnel. In this case, the outer IP addresses are different for each tunnel and therefore require a different pseudo header to be built for each UDP replicated

encapsulation.

The argument concerning redundant processing costs is valid regarding the integrity of a tunneled packet. In some architectures (e.g. PC-based routers), other mechanisms may also significantly reduce checksum processing costs: There are implementations that have optimised checksum processing algorithms, including the use of checksum-offloading. This processing is readily available for IPv4 packets at high line rates. Such processing may be anticipated for IPv6 endpoints, allowing receivers to reject corrupted packets without further processing. However, there are certain classes of tunnel end-points where this off-loading is not available and unlikely to become available in the near future.

1.3.3. Need to inspect the entire packet

The currently-deployed hardware in many routers uses a fast-path processing that only provides the first n bytes of a packet to the forwarding engine, where typically n <= 128. This prevents fast processing of a transport checksum over an entire (large) packet. Hence the currently defined IPv6 UDP checksum is poorly suited to use within a router that is unable to access the entire packet and does not provide checksum-offloading. Thus enabling checksum calculation over the complete packet can impact router design, performance improvement, energy consumption and/or cost.

1.3.4. Interactions with middleboxes

In IPv4, UDP-encapsulation may be desirable for NAT traversal, since UDP support is commonly provided. It is also necessary due to the almost ubiquitous deployment of IPv4 NATs. There has also been discussion of NAT for IPv6, although not for the same reason as in IPv4. If IPv6 NAT becomes a reality they hopefully do not present the same protocol issues as for IPv4. If NAT is defined for IPv6, it should take into consideration the use of a zero UDP checksum.

The requirements for IPv6 firewall traversal are likely be to be similar to those for IPv4. In addition, it can be reasonably expected that a firewall conforming to RFC 2460 will not regard datagrams with a zero UDP checksum as valid. Use of a zero UDP checksum with IPv6 requires firewalls to be updated before the full utility of the change is available.

It can be expected that datagrams with zero UDP checksum will initially not have the same middlebox traversal characteristics as regular UDP (RFC 2460). However when implementations follow the requirements specified in this document, we expect the traversal capabilities to improve over time. We also note that deployment of

IPv6-capable middleboxes is still in its initial phases. Thus, it might be that the number of non-updated boxes quickly become a very small percentage of the deployed middleboxes.

1.3.5. Support for load balancing

The UDP port number fields have been used as a basis to design load-balancing solutions for IPv4. This approach has also been leveraged for IPv6. An alternate method would be to utilise the IPv6 Flow Label as a basis for entropy for load balancing. This would have the desirable effect of releasing IPv6 load-balancing devices from the need to assume semantics for the use of the transport port field and also works for all type of transport protocols.

This use of the flow-label is consistent with the intended use, although further clarity may be needed to ensure the field can be consistently used for this purpose, (e.g. the updated IPv6 Flow Label [RFC6438] and Equal-Cost Multi-Path routing, ECMP [RFC6437]). Router vendors could be encouraged to start using the IPv6 Flow Label as a part of the flow hash, providing support for ECMP without requiring use of UDP.

However, the method for populating the outer IPv6 header with a value for the flow label is not trivial: If the inner packet uses IPv6, then the flow label value could be copied to the outer packet header. However, many current end-points set the flow label to a zero value (thus no entropy). The ingress of a tunnel seeking to provide good entropy in the flow label field would therefore need to create a random flow label value and keep corresponding state, so that all packets that were associated with a flow would be consistently given the same flow label. Although possible, this complexity may not be desirable in a tunnel ingress.

The end-to-end use of flow labels for load balancing is a long-term solution. Even if the usage of the flow label is clarified, there would be a transition time before a significant proportion of end-points start to assign a good quality flow label to the flows that they originate, with continued use of load balancing using the transport header fields until any widespread deployment is finally achieved.

2. Standards-Track Transports

The IETF has defined a set of transport protocols that may be applicable for tunnels with IPv6. There are also a set of network layer encapsulation tunnels such as IP-in-IP and GRE. These already standardized solutions are discussed here prior to the issues, as

background for the issue description and some comparison of where the issue may already occur.

2.1. UDP with Standard Checksum

UDP [RFC0768] with standard checksum behaviour, as defined in RFC 2460, has already been discussed. UDP usage guidelines are provided in [RFC5405].

2.2. UDP-Lite

UDP-Lite [RFC3828] offers an alternate transport to UDP, specified as a proposed standard, RFC 3828. A MIB is defined in RFC 5097 and unicast usage guidelines in [RFC5405]. There is at least one open source implementation as a part of the Linux kernel since version 2.6.20.

UDP-Lite provides a checksum with optional partial coverage. When using this option, a datagram is divided into a sensitive part (covered by the checksum) and an insensitive part (not covered by the checksum). When the checksum covers the entire packet, UDP-Lite is fully equivalent with UDP, with the exception that it uses a different value in the Next Header field in the IPv6 header. Errors/ corruption in the insensitive part will not cause the datagram to be discarded by the transport layer at the receiving endpoint. A minor side-effect of using UDP-Lite is that this was specified for damage-tolerant payloads and some link-layers may employ different link encapsulations when forwarding UDP-Lite segments (e.g. radio access bearers). Most link-layers will cover the insensitive part with the same strong layer 2 frame CRC that covers the sensitive part.

2.2.1. Using UDP-Lite as a Tunnel Encapsulation

Tunnel encapsulations can use UDP-Lite (e.g. Control And Provisioning of Wireless Access Points, CAPWAP [RFC5415]), since UDP-Lite provides a transport-layer checksum, including an IP pseudo header checksum, in IPv6, without the need for a router/middlebox to traverse the entire packet payload. This provides most of the verification required for delivery and still keeps a low complexity for the checksumming operation. UDP-Lite may set the length of checksum coverage on a per packet basis. This feature could be used if a tunnel protocol is designed to only verify delivery of the tunneled payload and uses a calculated checksum for control information.

There is currently poor support for middlebox traversal using UDP-Lite, because UDP-Lite uses a different IPv6 network-layer Next Header value to that of UDP, and few middleboxes are able to

interpret UDP-Lite and take appropriate actions when forwarding the packet. This makes UDP-Lite less suited to protocols needing general Internet support, until such time that UDP-Lite has achieved better support in middleboxes and end-points.

2.3. General Tunnel Encapsulations

The IETF has defined a set of tunneling protocols or network layer encapsulations, e.g., IP-in-IP and GRE. These either do not include a checksum or use a checksum that is optional, since tunnel encapsulations are typically layered directly over the Internet layer (identified by the upper layer type in the IPv6 Next Header field) and are also not used as endpoint transport protocols. There is little chance of confusing a tunnel-encapsulated packet with other application data that could result in corruption of application state or data.

From the end-to-end perspective, the principal difference is that the network-layer Next Header field identifies a separate transport, which reduces the probability that corruption could result in the packet being delivered to the wrong endpoint or application. Specifically, packets are only delivered to protocol modules that process a specific Next Header value. The Next Header field therefore provides a first-level check of correct demultiplexing. In contrast, the UDP port space is shared by many diverse applications and therefore UDP demultiplexing relies solely on the port numbers.

3. Issues Requiring Consideration

This informative section evaluates issues around the proposal to update IPv6 [RFC2460], to enable the UDP transport checksum to be set to zero. Some of the identified issues are shared with other protocols already in use. The section also provides background to the requirements and recommendations that follow.

The decision by RFC 2460 to omit an integrity check at the network level meant that the IPv6 transport checksum was overloaded with many functions, including validating:

- the endpoint address was not corrupted within a router, i.e., a packet was intended to be received by this destination and validate that the packet does not consist of a wrong header spliced to a different payload;

- that extension header processing is correctly delimited - i.e., the start of data has not been corrupted. In this case, reception of a valid Next Header value provides some protection;

- reassembly processing, when used;
- the length of the payload;
- the port values - i.e., the correct application receives the payload (applications should also check the expected use of source ports/addresses);
- the payload integrity.

In IPv4, the first four checks are performed using the IPv4 header checksum.

In IPv6, these checks occur within the endpoint stack using the UDP checksum information. An IPv6 node also relies on the header information to determine whether to send an ICMPv6 error message [RFC4443] and to determine the node to which this is sent. Corrupted information may lead to misdelivery to an unintended application socket on an unexpected host.

3.1. Effect of packet modification in the network

IP packets may be corrupted as they traverse an Internet path. Evidence has been presented [Sigcomm2000] to show that this was once an issue with IPv4 routers, and occasional corruption could result from bad internal router processing in routers or hosts. These errors are not detected by the strong frame checksums employed at the link-layer [RFC3819]. There is no current evidence that such cases are rare in the modern Internet, nor that they may not be applicable to IPv6. It therefore seems prudent not to relax this constraint. The emergence of low-end IPv6 routers and the proposed use of NAT with IPv6 further motivate the need to protect from this type of error.

Corruption in the network may result in:

- A datagram being misdelivered to the wrong host/router or the wrong transport entity within an endpoint. Such a datagram needs to be discarded;
- A datagram payload being corrupted, but still delivered to the intended host/router transport entity. Such a datagram needs to be either discarded or correctly processed by an application that provides its own integrity checks;
- A datagram payload being truncated by corruption of the length field. Such a datagram needs to be discarded.

When a checksum is used, this significantly reduces the impact of errors, reducing the probability of undetected corruption of state (and data) on both the host stack and the applications using the transport service.

The following sections examine the impact of modifying each of these header fields.

3.1.1. Corruption of the destination IP address

An IPv6 endpoint destination address could be modified in the network (e.g. corrupted by an error). This is not a concern for IPv4, because the IP header checksum will result in this packet being discarded by the receiving IP stack. Such modification in the network can not be detected at the network layer when using IPv6.

There are two possible outcomes:

- Delivery to a destination address that is not in use (the packet will not be delivered, but could result in an error report);

- Delivery to a different destination address. This modification will normally be detected by the transport checksum, resulting in silent discard. Without a computed checksum, the packet would be passed to the endpoint port demultiplexing function. If an application is bound to the associated ports, the packet payload will be passed to the application (see the subsequent section on port processing).

3.1.2. Corruption of the source IP address

This section examines what happens when the source address is corrupted in transit. This is not a concern in IPv4, because the IP header checksum will normally result in this packet being discarded by the receiving IP stack.

Corruption of an IPv6 source address does not result in the IP packet being delivered to a different endpoint protocol or destination address. If only the source address is corrupted, the datagram will likely be processed in the intended context, although with erroneous origin information. When using Unicast Reverse Path Forwarding [RFC2827], a change in address may result in the router discarding the packet when the route to the modified source address is different to that of the source address of the original packet.

The result will depend on the application or protocol that processes the packet. Some examples are:

- An application that requires a per-established context may disregard the datagram as invalid, or could map this to another context (if a context for the modified source address was already activated).

- A stateless application will process the datagram outside of any context, a simple example is the ECHO server, which will respond with a datagram directed to the modified source address. This would create unwanted additional processing load, and generate traffic to the modified endpoint address.

- Some datagram applications build state using the information from packet headers. A previously unused source address would result in receiver processing and the creation of unnecessary transport-layer state at the receiver. For example, Real Time Protocol (RTP) [RFC3550] sessions commonly employ a source independent receiver port. State is created for each received flow. Reception of a datagram with a corrupted source address will therefore result in accumulation of unnecessary state in the RTP state machine, including collision detection and response (since the same synchronization source, SSRC, value will appear to arrive from multiple source IP addresses).

- ICMP messages relating to a corrupted packet can be misdirected to the wrong source node.

In general, the effect of corrupting the source address will depend upon the protocol that processes the packet and its robustness to this error. For the case where the packet is received by a tunnel endpoint, the tunnel application is expected to correctly handle a corrupted source address.

The impact of source address modification is more difficult to quantify when the receiving application is not that originally intended and several fields have been modified in transit.

3.1.3. Corruption of Port Information

This section describes what happens if one or both of the UDP port values are corrupted in transit. This can also happen with IPv4 is used with a zero UDP checksum, but not when UDP checksums are calculated or when UDP-Lite is used. If the ports carried in the transport header of an IPv6 packet were corrupted in transit, packets may be delivered to the wrong application process (on the intended machine) and/or responses or errors sent to the wrong application process (on the intended machine).

3.1.4. Delivery to an unexpected port

If one combines the corruption effects, such as destination address and ports, there is a number of potential outcomes when traffic arrives at an unexpected port. This section discusses these possibilities and their outcomes for a packet that does not use the UDP checksum validation:

- Delivery to a port that is not in use. The packet is discarded, but could generate an ICMPv6 message (e.g. port unreachable).

- It could be delivered to a different node that implements the same application, where the packet may be accepted, generating side-effects or accumulated state.

- It could be delivered to an application that does not implement the tunnel protocol, where the packet may be incorrectly parsed, and may be misinterpreted, generating side-effects or accumulated state.

The probability of each outcome depends on the statistical probability that the address or the port information for the source or destination becomes corrupt in the datagram such that they match those of an existing flow or server port. Unfortunately, such a match may be more likely for UDP than for connection-oriented transports, because:

1. There is no handshake prior to communication and no sequence numbers (as in TCP, DCCP, or SCTP). Together, this makes it hard to verify that an application process is given only the application data associated with a specific transport session.

2. Applications writers often bind to wild-card values in endpoint identifiers and do not always validate correctness of datagrams they receive (guidance on this topic is provided in [RFC5405]).

While these rules could, in principle, be revised to declare naive applications as "Historic". This remedy is not realistic: the transport owes it to the stack to do its best to reject bogus datagrams.

If checksum coverage is suppressed, the application therefore needs to provide a method to detect and discard the unwanted data. A tunnel protocol would need to perform its own integrity checks on any control information if transported in datagrams with a zero UDP checksum. If the tunnel payload is another IP packet, the packets requiring checksums can be assumed to have their own checksums provided that the rate of corrupted packets is not significantly

larger due to the tunnel encapsulation. If a tunnel transports other inner payloads that do not use IP, the assumptions of corruption detection for that particular protocol must be fulfilled, this may require an additional checksum/CRC and/or integrity protection of the payload and tunnel headers.

A protocol that uses a zero UDP checksum can not assume that it is the only protocol using a zero UDP checksum. Therefore, it needs to gracefully handle misdelivery. It must be robust to reception of malformed packets received on a listening port and expect that these packets may contain corrupted data or data associated with a completely different protocol.

3.1.5. Corruption of Fragmentation Information

The fragmentation information in IPv6 employs a 32-bit identity field, compared to only a 16-bit field in IPv4, a 13-bit fragment offset and a 1-bit flag, indicating if there are more fragments. Corruption of any of these field may result in one of two outcomes:

Reassembly failure: An error in the "More Fragments" field for the last fragment will for example result in the packet never being considered complete and will eventually be timed out and discarded. A corruption in the ID field will result in the fragment not being delivered to the intended context thus leaving the rest incomplete, unless that packet has been duplicated prior to corruption. The incomplete packet will eventually be timed out and discarded.

Erroneous reassembly: The re-assembled packet did not match the original packet. This can occur when the ID field of a fragment is corrupted, resulting in a fragment becoming associated with another packet and taking the place of another fragment. Corruption in the offset information can cause the fragment to be misaligned in the reassembly buffer, resulting in incorrect reassembly. Corruption can cause the packet to become shorter or longer, however completion of reassembly is much less probable, since this would require consistent corruption of the IPv6 headers payload length field and the offset field. The possibility of mis-assembly requires the reassembling stack to provide strong checks that detect overlap or missing data, note however that this is not guaranteed and has been clarified in "Handling of Overlapping IPv6 Fragments" [RFC5722].

The erroneous reassembly of packets is a general concern and such packets should be discarded instead of being passed to higher layer processes. The primary detector of packet length changes is the IP payload length field, with a secondary check by the transport

checksum. The Upper-Layer Packet length field included in the pseudo header assists in verifying correct reassembly, since the Internet checksum has a low probability of detecting insertion of data or overlap errors (due to misplacement of data). The checksum is also incapable of detecting insertion or removal of all zero-data that occurs in a multiple of a 16-bit chunk.

The most significant risk of corruption results following mis-association of a fragment with a different packet. This risk can be significant, since the size of fragments is often the same (e.g. fragments resulting when the path MTU results in fragmentation of a larger packet, common when addition of a tunnel encapsulation header expands the size of a packet). Detection of this type of error requires a checksum or other integrity check of the headers and the payload. Such protection is anyway desirable for tunnel encapsulations using IPv4, since the small fragmentation ID can easily result in wrap-around [RFC4963], this is especially the case for tunnels that perform flow aggregation [I-D.ietf-intarea-tunnels].

Tunnel fragmentation behavior matters. There can be outer or inner fragmentation "Tunnels in the Internet Architecture" [I-D.ietf-intarea-tunnels]. If there is inner fragmentation by the tunnel, the outer headers will never be fragmented and thus a zero UDP checksum in the outer header will not affect the reassembly process. When a tunnel performs outer header fragmentation, the tunnel egress needs to perform reassembly of the outer fragments into an inner packet. The inner packet is either a complete packet or a fragment. If it is a fragment, the destination endpoint of the fragment will perform reassembly of the received fragments. The complete packet or the reassembled fragments will then be processed according to the packet Next Header field. The receiver may only detect reassembly anomalies when it uses a protocol with a checksum. The larger the number of reassembly processes to which a packet has been subjected, the greater the probability of an error.

- An IP-in-IP tunnel that performs inner fragmentation has similar properties to a UDP tunnel with a zero UDP checksum that also performs inner fragmentation.

- An IP-in-IP tunnel that performs outer fragmentation has similar properties to a UDP tunnel with a zero UDP checksum that performs outer fragmentation.

- A tunnel that performs outer fragmentation can result in a higher level of corruption due to both inner and outer fragmentation, enabling more chances for reassembly errors to occur.

- Recursive tunneling can result in fragmentation at more than one header level, even for inner fragmentation unless it goes to the inner-most IP header.

- Unless there is verification at each reassembly, the probability for undetected error will increase with the number of times fragmentation is recursively applied, making IP-in-IP and UDP with zero UDP checksum both vulnerable to undetected errors.

In conclusion, fragmentation of datagrams with a zero UDP checksum does not worsen the performance compared to some other commonly used tunnel encapsulations. However, caution is needed for recursive tunneling without any additional verification at the different tunnel layers.

3.2. Where Packet Corruption Occurs

Corruption of IP packets can occur at any point along a network path, during packet generation, during transmission over the link, in the process of routing and switching, etc. Some transmission steps include a checksum or Cyclic Redundancy Check (CRC) that reduces the probability for corrupted packets being forwarded, but there still exists a probability that errors may propagate undetected. Unfortunately the community lacks reliable information to identify the most common functions or equipment that result in packet corruption. However, there are indications that the place where corruption occurs can vary significantly from one path to another. There is therefore a risk in applying evidence from one domain of usage to infer characteristics for another. Methods intended for general Internet usage must therefore assume that corruption can occur and deploy mechanisms to mitigate the effect of corruption and/or resulting misdelivery.

3.3. Validating the network path

IP transports designed for use in the general Internet should not assume specific path characteristics. Network protocols may reroute packets that change the set of routers and middleboxes along a path. Therefore transports such as TCP, SCTP and DCCP have been designed to negotiate protocol parameters, adapt to different network path characteristics, and receive feedback to verify that the current path is suited to the intended application. Applications using UDP and UDP-Lite need to provide their own mechanisms to confirm the validity of the current network path.

A zero value in the UDP checksum field is explicitly disallowed in RFC2460. Thus it may be expected that any device on the path that has a reason to look beyond the IP header will consider such a packet

as erroneous or illegal and may discard it, unless the device is updated to support the new behavior. A pair of end-points intending to use a new behavior will therefore not only need to ensure support at each end-point, but also that the path between them will deliver packets with the new behavior. This may require using negotiation or an explicit mandate to use the new behavior by all nodes that support the new protocol.

Enabling the use of a zero checksum places new requirements on equipment deployed within the network, such as middleboxes. A middlebox (e.g. Firewalls, Network Address Translators) may enable zero checksum usage for a particular range of ports. Note that checksum off-loading and operating system design may result in all IPv6 UDP traffic being sent with a calculated checksum. This requires middleboxes that are configured to enable a zero UDP checksum to continue to work with bidirectional UDP flows that use a zero UDP checksum in only one direction, and therefore they must not maintain separate state for a UDP flow based on its checksum usage.

Support along the path between end points can be guaranteed in limited deployments by appropriate configuration. In general, it can be expected to take time for deployment of any updated behaviour to become ubiquitous.

A sender will need to probe the path to verify the expected behavior. Path characteristics may change, and usage therefore should be robust and able to detect a failure of the path under normal usage and re-negotiate. Note that a bidirectional path does not necessarily support the same checksum usage in both the forward and return directions: Receipt of a datagram with a zero UDP checksum, does not imply that the remote endpoint can also receive a datagram with a zero UDP checksum. This will require periodic validation of the path, adding complexity to any solution using the new behavior.

3.4. Applicability of method

The update to the IPv6 specification defined in [I-D.ietf-6man-udpchecksums] only modifies IPv6 nodes that implement specific protocols designed to permit omission of a UDP checksum. This document therefore provides an applicability statement for the updated method indicating when the mechanism can (and can not) be used. Enabling this, and ensuring correct interactions with the stack, implies much more than simply disabling the checksum algorithm for specific packets at the transport interface.

When the method is widely available, it may be expected to be used by applications that are perceived to gain benefit. Any solution that uses an end-to-end transport protocol, rather than an IP-in-IP

encapsulation, needs to minimise the possibility that application processes could confuse a corrupted or wrongly delivered UDP datagram with that of data addressed to the application running on their endpoint.

The protocol or application that uses the zero checksum method must ensure that the lack of checksum does not affect the protocol operation. This includes being robust to receiving a unintended packet from another protocol or context following corruption of a destination or source address and/or port value. It also includes considering the need for additional implicit protection mechanisms required when using the payload of a UDP packet received with a zero checksum.

3.5. Impact on non-supporting devices or applications

It is important to consider the potential impact of using a zero UDP checksum on end-point devices or applications that are not modified to support the new behavior or by default or preference, use the regular behavior. These applications must not be significantly impacted by the update.

To illustrate why this necessary, consider the implications of a node that enables use of a zero UDP checksum at the interface level: This would result in all applications that listen to a UDP socket receiving datagrams where the checksum was not verified. This could have a significant impact on an application that was not designed with the additional robustness needed to handle received packets with corruption, creating state or destroying existing state in the application.

A zero UDP checksum therefore needs to be enabled only for individual ports using an explicit request by the application. In this case, applications using other ports would maintain the current IPv6 behavior, discarding incoming datagrams with a zero UDP checksum. These other applications would not be affected by this changed behavior. An application that allows the changed behavior should be aware of the risk of corruption and the increased level of misdirected traffic, and can be designed robustly to handle this risk.

4. Constraints on implementation of IPv6 nodes supporting zero checksum

This section is an applicability statement that defines requirements and recommendations on the implementation of IPv6 nodes that support use of a zero value in the checksum field of a UDP datagram.

All implementations that support this zero UDP checksum method MUST conform to the requirements defined below.

1. An IPv6 sending node MAY use a calculated RFC 2460 checksum for all datagrams that it sends. This explicitly permits an interface that supports checksum offloading to insert an updated UDP checksum value in all UDP datagrams that it forwards, however note that sending a calculated checksum requires the receiver to also perform the checksum calculation. Checksum offloading can normally be switched off for a particular interface to ensure that datagrams are sent with a zero UDP checksum.

2. IPv6 nodes SHOULD by default NOT allow the zero UDP checksum method for transmission.

3. IPv6 nodes MUST provide a way for the application/protocol to indicate the set of ports that will be enabled to send datagrams with a zero UDP checksum. This may be implemented by enabling a transport mode using a socket API call when the socket is established, or a similar mechanism. It may also be implemented by enabling the method for a pre-assigned static port used by a specific tunnel protocol.

4. IPv6 nodes MUST provide a method to allow an application/ protocol to indicate that a particular UDP datagram is required to be sent with a UDP checksum. This needs to be allowed by the operating system at any time (e.g. to send keep-alive datagrams), not just when a socket is established in the zero checksum mode.

5. The default IPv6 node receiver behaviour MUST discard all IPv6 packets carrying datagrams with a zero UDP checksum.

6. IPv6 nodes MUST provide a way for the application/protocol to indicate the set of ports that will be enabled to receive datagrams with a zero UDP checksum. This may be implemented via a socket API call, or similar mechanism. It may also be implemented by enabling the method for a pre-assigned static port used by a specific tunnel protocol.

7. IPv6 nodes supporting usage of zero UDP checksums MUST also allow reception using a calculated UDP checksum on all ports configured to allow zero UDP checksum usage. (The sending endpoint, e.g. encapsulating ingress, may choose to compute the UDP checksum, or may calculate this by default.) The receiving endpoint MUST use the reception method specified in RFC2460 when the checksum field is not zero.

8. RFC 2460 specifies that IPv6 nodes SHOULD log received datagrams with a zero UDP checksum. This remains the case for any datagram received on a port that does not explicitly enable processing of a zero UDP checksum. A port for which the zero UDP checksum has been enabled MUST NOT log the datagram solely because the checksum value is zero.

9. IPv6 nodes MAY separately identify received UDP datagrams that are discarded with a zero UDP checksum. It SHOULD NOT add these to the standard log, since the endpoint has not been verified. This may be used to support other functions (such as a security policy).

10. IPv6 nodes that receive ICMPv6 messages that refer to packets with a zero UDP checksum MUST provide appropriate checks concerning the consistency of the reported packet to verify that the reported packet actually originated from the node, before acting upon the information (e.g. validating the address and port numbers in the ICMPv6 message body).

5. Requirements on usage of the zero UDP checksum

This section is an applicability statement that identifies requirements and recommendations for protocols and tunnel encapsulations that are transported over an IPv6 transport flow (e.g. tunnel) that does not perform a UDP checksum calculation to verify the integrity at the transport endpoints.

1. Transported protocols that enable the use of zero UDP checksum MUST only enable this for a specific port or port-range. This needs to be enabled at the sending and receiving endpoints for a UDP flow.

2. An integrity mechanism is always RECOMMENDED at the transported protocol layer to ensure that corruption rates of the delivered payload is not increased (e.g. the inner-most packet of a UDP tunnel). A mechanism that isolates the causes of corruption (e.g. identifying misdelivery, IPv6 header corruption, tunnel header corruption) is expected to also provide additional information about the status of the tunnel (e.g. to suggest a security attack).

3. A transported protocol that encapsulates Internet Protocol (IPv4 or IPv6) packets MAY rely on the inner packet integrity checks, provided that the tunnel protocol will not significantly increase the rate of corruption of the inner IP packet. If a significantly increased corruption rate can occur, then the

tunnel protocol MUST provide an additional integrity verification mechanism. Early detection is desirable to avoid wasting unnecessary computation, transmission capacity or storage for packets that will subsequently be discarded.

4. A transported protocol that supports use of a zero UDP checksum, MUST be designed so that corruption of this information does not result in accumulated state for the protocol.

5. A transported protocol with a non-tunnel payload or one that encapsulates non-IP packets MUST have a CRC or other mechanism for checking packet integrity, unless the non-IP packet is specifically designed for transmission over a lower layer that does not provide a packet integrity guarantee.

6. A transported protocol with control feedback SHOULD be robust to changes in the network path, since the set of middleboxes on a path may vary during the life of an association. The UDP endpoints need to discover paths with middleboxes that drop packets with a zero UDP checksum. Therefore, transported protocols SHOULD send keep-alive messages with a zero UDP checksum. An endpoint that discovers an appreciable loss rate for keep-alive packets MAY terminate the UDP flow (e.g. tunnel). Section 3.1.3 of RFC 5405 describes requirements for congestion control when using a UDP-based transport.

7. A protocol with control feedback that can fall-back to using UDP with a calculated RFC 2460 checksum is expected to be more robust to changes in the network path. Therefore, keep-alive messages SHOULD include both UDP datagrams with a checksum and datagrams with a zero UDP checksum. This will enable the remote endpoint to distinguish between a path failure and dropping of datagrams with a zero UDP checksum.

8. A middlebox implementation MUST allow forwarding of an IPv6 UDP datagram with both a zero and standard UDP checksum using the same UDP port.

9. A middlebox MAY configure a restricted set of specific port ranges that forward UDP datagrams with a zero UDP checksum. The middlebox MAY drop IPv6 datagrams with a zero UDP checksum that are outside a configured range.

10. When a middlebox forwards an IPv6 UDP flow containing datagrams with both a zero and standard UDP checksum, the middlebox MUST NOT maintain separate state for flows depending on the value of their UDP checksum field. (This requirement is necessary to enable a sender that always calculates a checksum to communicate

via a middlebox with a remote endpoint that uses a zero UDP checksum.)

6. Summary

This document examines the role of the UDP transport checksum when used with IPv6. It presents a summary of the trade-offs in evaluating the safety of updating RFC 2460 to permit an IPv6 endpoint to use a zero UDP checksum field to indicate that no checksum is present.

The use of UDP with a zero UDP checksum has merits for some applications, such as tunnel encapsulation, and is widely used in IPv4. However, there are different dangers for IPv6: There is an increased risk of corruption and misdelivery when using zero UDP checksum in IPv6 compared to using IPv4 due to the lack of an IPv6 header checksum. Thus, applications need to re-evaluate the risks of enabling use of a zero UDP checksum and consider a solution that at least provides the same delivery protection as for IPv4, for example by utilizing UDP-Lite, or by enabling the UDP checksum. The use of checksum off-loading may help alleviate the checksum processing cost and permit use of a checksum using method defined in RFC 2460.

Tunnel applications using UDP for encapsulation can in many cases use a zero UDP checksum without significant impact on the corruption rate. A well-designed tunnel application should include consistency checks to validate the header information encapsulated with a received packet. In most cases, tunnels encapsulating IP packets can rely on the integrity protection provided by the transported protocol (or tunneled inner packet). When correctly implemented, such an endpoint will not be negatively impacted by omission of the transport-layer checksum. Recursive tunneling and fragmentation is a potential issue that can raise corruption rates significantly, and requires careful consideration.

Other UDP applications at the intended destination node or another node can be impacted if they are allowed to receive datagrams that have a zero UDP checksum. It is important that already deployed applications are not impacted by a change at the transport layer. If these applications execute on nodes that implement RFC 2460, they will discard (and log) all datagrams with a zero UDP checksum. This is not an issue.

In general, UDP-based applications need to employ a mechanism that allows a large percentage of the corrupted packets to be removed before they reach an application, both to protect the data stream of the application and the control plane of higher layer protocols.

These checks are currently performed by the UDP checksum for IPv6, or the reduced checksum for UDP-Lite when used with IPv6.

The transport of recursive tunneling and the use of fragmentation pose difficult issues that need to be considered in the design of tunnel protocols. There is an increased risk of an error in the inner-most packet when fragmentation when several layers of tunneling and several different reassembly processes are run without verification of correctness. This requires extra thought and careful consideration in the design of transported tunnels.

The use of the updated method must consider the implications on firewalls, NATs and other middleboxes. It is not expected that IPv6 NATs handle IPv6 UDP datagrams in the same way that they handle IPv4 UDP datagrams. This possibly reduces the need to update the checksum. Firewalls are intended to be configured, and therefore may need to be explicitly updated to allow new services or protocols. IPv6 middlebox deployment is not yet as prolific as it is in IPv4, and therefore new devices are expected to follow the methods specified in this document.

Each application should consider the implications of choosing an IPv6 transport that uses a zero UDP checksum, and consider whether other standard methods may be more appropriate, and may simplify application design.

7. Acknowledgements

Brian Haberman, Brian Carpenter, Margaret Wasserman, Lars Eggert, others in the TSV directorate. Barry Leiba, Ronald Bonica and Stewart Bryant are thanked for resulting in a document with much greater applicability. Thanks to P.F. Chimento for careful review and editorial corrections.

Thanks also to: Remi Denis-Courmont, Pekka Savola, Glen Turner, and many others who contributed comments and ideas via the 6man, behave, lisp and mboned lists.

8. IANA Considerations

This document does not require any actions by IANA.

9. Security Considerations

Transport checksums provide the first stage of protection for the

stack, although they can not be considered authentication mechanisms. These checks are also desirable to ensure packet counters correctly log actual activity, and can be used to detect unusual behaviours.

Depending on the hardware design, the processing requirements may differ for tunnels that have a zero UDP checksum and those that calculate a checksum. This processing overhead may need to be considered when deciding whether to enable a tunnel and to determine an acceptable rate for transmission.

Transmission of IPv6 packets with a zero UDP checksum could reveal additional information to an on-path attacker to identify the operating system or configuration of a sending node. There is a need to probe the network path to determine whether the path supports using IPv6 packets with a zero UDP checksum. The details of the probing mechanism may differ for different tunnel encapsulations and if visible in the network (e.g. if not using IPsec in encryption mode) could reveal additional information to an on-path attacker to identify the type of tunnel being used.

IP-in-IP or GRE tunnels offer good traversal of middleboxes that have not been designed for security, e.g. firewalls. However, firewalls may be expected to be configured to block general tunnels as they present a large attack surface. This applicability statement therefore permits this method to be enabled only for specific ranges of ports.

When enabled, nodes and middleboxes must forward received UDP datagrams that have either a calculated checksum or a zero checksum.

10. References

10.1. Normative References

[I-D.ietf-6man-udpchecksums]
Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", draft-ietf-6man-udpchecksums-07 (work in progress), January 2013.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

10.2. Informative References

[I-D.ietf-intarea-tunnels]
Touch, J. and M. Townsley, "Tunnels in the Internet Architecture", draft-ietf-intarea-tunnels-00 (work in progress), March 2010.

[I-D.ietf-mboned-auto-multicast]
Bumgardner, G., "Automatic Multicast Tunneling", draft-ietf-mboned-auto-multicast-14 (work in progress), June 2012.

[LISP] D. Farinacci et al, "Locator/ID Separation Protocol (LISP)", November 2012.

[RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.

[RFC1141] Mallory, T. and A. Kullberg, "Incremental updating of the Internet checksum", RFC 1141, January 1990.

[RFC1624] Rijsinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.

[RFC2827] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, May 2000.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3819] Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, July 2004.

[RFC3828] Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)", RFC 3828, July 2004.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control

Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

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

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

[RFC5415] Calhoun, P., Montemurro, M., and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification", RFC 5415, March 2009.

[RFC5722] Krishnan, S., "Handling of Overlapping IPv6 Fragments", RFC 5722, December 2009.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, November 2011.

[RFC6438] Carpenter, B. and S. Amante, "Using the IPv6 Flow Label for Equal Cost Multipath Routing and Link Aggregation in Tunnels", RFC 6438, November 2011.

[Sigcomm2000] Jonathan Stone and Craig Partridge , "When the CRC and TCP Checksum Disagree", 2000.

[UDPTT] G Fairhurst, "The UDP Tunnel Transport mode", Feb 2010.

Appendix A. Evaluation of proposal to update RFC 2460 to support zero checksum

This informative appendix documents the evaluation of the proposal to update IPv6 [RFC2460], to provide the option that some nodes may suppress generation and checking of the UDP transport checksum. It also compares the proposal with other alternatives, and notes that for a particular application some standard methods may be more appropriate than using IPv6 with a zero UDP checksum.

A.1. Alternatives to the Standard Checksum

There are several alternatives to the normal method for calculating the UDP Checksum [RFC1071] that do not require a tunnel endpoint to inspect the entire packet when computing a checksum. These include (in decreasing order of complexity):

- Delta computation of the checksum from an encapsulated checksum field. Since the checksum is a cumulative sum [RFC1624], an encapsulating header checksum can be derived from the new pseudo header, the inner checksum and the sum of the other network-layer fields not included in the pseudo header of the encapsulated packet, in a manner resembling incremental checksum update [RFC1141]. This would not require access to the whole packet, but does require fields to be collected across the header, and arithmetic operations on each packet. The method would only work for packets that contain a 2's complement transport checksum (i.e., it would not be appropriate for SCTP or when IP fragmentation is used).

- UDP-Lite with the checksum coverage set to only the header portion of a packet. This requires a pseudo header checksum calculation only on the encapsulating packet header. The computed checksum value may be cached (before adding the Length field) for each flow/destination and subsequently combined with the Length of each packet to minimise per-packet processing. This value is combined with the UDP payload length for the pseudo header, however this length is expected to be known when performing packet forwarding.

- The proposed UDP Tunnel Transport [UDPTT] suggested a method where UDP would be modified to derive the checksum only from the encapsulating packet protocol header. This value does not change between packets in a single flow. The value may be cached per flow/destination to minimise per-packet processing.

- There has been a proposal to simply ignore the UDP checksum value on reception at the tunnel egress, allowing a tunnel ingress to insert any value correct or false. For tunnel usage, a non standard checksum value may be used, forcing an RFC 2460 receiver to drop the packet. The main downside is that it would be impossible to identify a UDP datagram (in the network or an endpoint) that is treated in this way compared to a packet that has actually been corrupted.

- A method has been proposed that uses a new (to be defined) IPv6 Destination Options Header to provide an end-to-end validation check at the network layer. This would allow an endpoint to verify delivery to an appropriate end point, but would also require IPv6 nodes to correctly handle the additional header, and would require changes to middlebox behavior (e.g. when used with a NAT that always adjusts the checksum value).

- UDP modified to disable checksum processing [I-D.ietf-6man-udpchecksums]. This eliminates the need for a checksum calculation, but would require constraints on appropriate

usage and updates to end-points and middleboxes.

- IP-in-IP tunneling. As this method completely dispenses with a transport protocol in the outer-layer it has reduced overhead and complexity, but also reduced functionality. There is no outer checksum over the packet and also no ports to perform demultiplexing between different tunnel types. This reduces the information available upon which a load balancer may act.

These options are compared and discussed further in the following sections.

A.2. Comparison

This section compares the above listed methods to support datagram tunneling. It includes proposals for updating the behaviour of UDP.

While this comparison focuses on applications that are expected to execute on routers, the distinction between a router and a host is not always clear, especially at the transport level. Systems (such as unix-based operating systems) routinely provide both functions. There is no way to identify the role of the receiving node from a received packet.

A.2.1. Middlebox Traversal

Regular UDP with a standard checksum or the delta encoded optimization for creating correct checksums have the best possibilities for successful traversal of a middlebox. No new support is required.

A method that ignores the UDP checksum on reception is expected to have a good probability of traversal, because most middleboxes perform an incremental checksum update. UDPTT would also have been able to traverse a middlebox with this behaviour. However, a middlebox on the path that attempts to verify a standard checksum will not forward packets using either of these methods, preventing traversal. A method that ignores the checksum has an additional downside in that it prevents improvement of middlebox traversal, because there is no way to identify UDP datagrams that use the modified checksum behaviour.

IP-in-IP or GRE tunnels offer good traversal of middleboxes that have not been designed for security, e.g. firewalls. However, firewalls may be expected to be configured to block general tunnels as they present a large attack surface.

A new IPv6 Destination Options header will suffer traversal issues

with middleboxes, especially Firewalls and NATs, and will likely require them to be updated before the extension header is passed.

Datagrams with a zero UDP checksum will not be passed by any middlebox that validates the checksum using RFC 2460 or updates the checksum field, such as NAT or firewalls. This would require an update to correctly handle a datagram with a zero UDP checksum.

UDP-Lite will require an update of almost all type of middleboxes, because it requires support for a separate network-layer protocol number. Once enabled, the method to support incremental checksum update would be identical to that for UDP, but different for checksum validation.

A.2.2. Load Balancing

The usefulness of solutions for load balancers depends on the difference in entropy in the headers for different flows that can be included in a hash function. All the proposals that use the UDP protocol number have equal behavior. UDP-Lite has the potential for equally good behavior as for UDP. However, UDP-Lite is currently unlikely to be supported by deployed hashing mechanisms, which could cause a load balancer to not use the transport header in the computed hash. A load balancer that only uses the IP header will have low entropy, but could be improved by including the IPv6 the flow label, providing that the tunnel ingress ensures that different flow labels are assigned to different flows. However, a transition to the common use of good quality flow labels is likely to take time to deploy.

A.2.3. Ingress and Egress Performance Implications

IP-in-IP tunnels are often considered efficient, because they introduce very little processing and low data overhead. The other proposals introduce a UDP-like header incurring associated data overhead. Processing is minimised for the method that uses a zero UDP checksum, ignoring the UDP checksum on reception, and only slightly higher for UDPTT, the extension header and UDP-Lite. The delta-calculation scheme operates on a few more fields, but also introduces serious failure modes that can result in a need to calculate a checksum over the complete datagram. Regular UDP is clearly the most costly to process, always requiring checksum calculation over the entire datagram.

It is important to note that the zero UDP checksum method, ignoring checksum on reception, the Option Header, UDPTT and UDP-Lite will likely incur additional complexities in the application to incorporate a negotiation and validation mechanism.

A.2.4. Deployability

The major factors influencing deployability of these solutions are a need to update both end-points, a need for negotiation and the need to update middleboxes. These are summarised below:

- The solution with the best deployability is regular UDP. This requires no changes and has good middlebox traversal characteristics.

- The next easiest to deploy is the delta checksum solution. This does not modify the protocol on the wire and only needs changes in tunnel ingress.

- IP-in-IP tunnels should not require changes to the end-points, but raise issues when traversing firewalls and other security-type devices, which are expected to require updates.

- Ignoring the checksum on reception will require changes at both end-points. The never ceasing risk of path failure requires additional checks to ensure this solution is robust and will require changes or additions to the tunnel control protocol to negotiate support and validate the path.

- The remaining solutions offer similar deployability. UDP-Lite requires support at both end-points and in middleboxes. UDPTT and the zero UDP checksum method with or without an extension header require support at both end-points and in middleboxes. UDP-Lite, UDPTT, and the zero UDP checksum method and use of extension headers may additionally require changes or additions to the tunnel control protocol to negotiate support and path validation.

A.2.5. Corruption Detection Strength

The standard UDP checksum and the delta checksum can both provide some verification at the tunnel egress. This can significantly reduce the probability that a corrupted inner packet is forwarded. UDP-Lite, UDPTT and the extension header all provide some verification against corruption, but do not verify the inner packet. They only provide a strong indication that the delivered packet was intended for the tunnel egress and was correctly delimited. The methods using a zero UDP checksum, ignoring the UDP checksum on reception and IP-and-IP encapsulation all provide no verification that a received datagram was intended to be processed by a specific tunnel egress or that the inner encapsulated packet was correct.

A.2.6. Comparison Summary

The comparisons above may be summarised as "there is no silver bullet that will slay all the issues". One has to select which down side(s) can best be lived with. Focusing on the existing solutions, this can be summarized as:

Regular UDP: The method defined in RFC 2460 has good middlebox traversal and load balancing and multiplexing, requiring a checksum in the outer headers covering the whole packet.

IP in IP: A low complexity encapsulation, with limited middlebox traversal, no multiplexing support, and currently poor load balancing support that could improve over time.

UDP-Lite: A medium complexity encapsulation, with good multiplexing support, limited middlebox traversal, but possible to improve over time, currently poor load balancing support that could improve over time, in most cases requiring application level negotiation to select the protocol and validation to confirm the path forwards UDP-Lite.

The delta-checksum is an optimization in the processing of UDP, as such it exhibits some of the drawbacks of using regular UDP.

The remaining proposals may be described in similar terms:

Zero-Checksum: A low complexity encapsulation, with good multiplexing support, limited middlebox traversal that could improve over time, good load balancing support, in most cases requiring application level negotiation and validation to confirm the path forwards a zero UDP checksum.

UDPTT: A medium complexity encapsulation, with good multiplexing support, limited middlebox traversal, but possible to improve over time, good load balancing support, in most cases requiring application level negotiation to select the transport and validation to confirm the path forwards UDPTT datagrams.

IPv6 Destination Option IP in IP tunneling: A medium complexity, with no multiplexing support, limited middlebox traversal, currently poor load balancing support that could improve over time, in most cases requiring negotiation to confirm the option is supported and validation to confirm the path forwards the option.

IPv6 Destination Option combined with UDP Zero-checksuming: A medium complexity encapsulation, with good multiplexing support, limited load balancing support that could improve over time, in most cases requiring negotiation to confirm the option is supported and validation to confirm the path forwards the option.

Ignore the checksum on reception: A low complexity encapsulation, with good multiplexing support, medium middlebox traversal that never can improve, good load balancing support, in most cases requiring negotiation to confirm the option is supported by the remote endpoint and validation to confirm the path forwards a zero UDP checksum.

There is no clear single optimum solution. If the most important need is to traverse middleboxes, then the best choice is to stay with regular UDP and consider the optimizations that may be required to perform the checksumming. If one can live with limited middlebox traversal, low complexity is necessary and one does not require load balancing, then IP-in-IP tunneling is the simplest. If one wants strengthened error detection, but with currently limited middlebox traversal and load-balancing. UDP-Lite is appropriate. Zero UDP checksum addresses another set of constraints, low complexity and a need for load balancing from the current Internet, providing it can live with currently limited middlebox traversal.

Techniques for load balancing and middlebox traversal do continue to evolve. Over a long time, developments in load balancing have good potential to improve. This time horizon is long since it requires both load balancer and end-point updates to get full benefit. The challenges of middlebox traversal are also expected to change with time, as device capabilities evolve. Middleboxes are very prolific with a larger proportion of end-user ownership, and therefore may be expected to take long time cycles to evolve.

One potential advantage is that the deployment of IPv6-capable middleboxes are still in its initial phase and the quicker a new method becomes standardized, the fewer boxes will be non-compliant.

Thus, the question of whether to permit use of datagrams with a zero UDP checksum for IPv6 under reasonable constraints, is therefore best viewed as a trade-off between a number of more subjective questions:

- Is there sufficient interest in using a zero UDP checksum with the given constraints (summarised below)?

- Are there other avenues of change that will resolve the issue in a better way and sufficiently quickly ?

- Do we accept the complexity cost of having one more solution in the future?

The analysis concludes that the IETF should carefully consider constraints on sanctioning the use of any new transport mode. The 6man working group of the IETF has determined that the answer to the above questions are sufficient to update IPv6 to standardise use of a zero UDP checksum for use by tunnel encapsulations for specific applications.

Each application should consider the implications of choosing an IPv6 transport that uses a zero UDP checksum. In many cases, standard methods may be more appropriate, and may simplify application design. The use of checksum off-loading may help alleviate the checksum processing cost and permit use of a checksum using method defined in RFC 2460.

Appendix B. Document Change History

{RFC EDITOR NOTE: This section must be deleted prior to publication}

Individual Draft 00 This is the first DRAFT of this document - It contains a compilation of various discussions and contributions from a variety of IETF WGs, including: mboned, tsv, 6man, lisp, and behave. This includes contributions from Magnus with text on RTP, and various updates.

Individual Draft 01

* This version corrects some typos and editorial NiTs and adds discussion of the need to negotiate and verify operation of a new mechanism (3.3.4).

Individual Draft 02

* Version -02 corrects some typos and editorial NiTs.

* Added reference to ECMP for tunnels.

* Clarifies the recommendations at the end of the document.

Working Group Draft 00

* Working Group Version -00 corrects some typos and removes much of rationale for UDPTT. It also adds some discussion of IPv6 extension header.

Working Group Draft 01

* Working Group Version -01 updates the rules and incorporates off-list feedback. This version is intended for wider review within the 6man working group.

Working Group Draft 02

* This version is the result of a major rewrite and re-ordering of the document.

* A new section comparing the results have been added.

* The constraints list has been significantly altered by removing some and rewording other constraints.

* This contains other significant language updates to clarify the intent of this draft.

Working Group Draft 03

* Editorial updates

Working Group Draft 04

* Resubmission only updating the AMT and RFC2765 references.

Working Group Draft 05

* Resubmission to correct editorial NiTs - thanks to Bill Atwood for noting these.Group Draft 05.

Working Group Draft 06

* Resubmission to keep draft alive (spelling updated from 05).

Working Group Draft 07

* Interim Version

* Submission after IESG Feedback

* Updates to enable the document to become a PS Applicability Statement

Working Group Draft 08

* First Version written as a PS Applicability Statement

* Changes to reflect decision to update RFC 2460, rather than recommend decision

* Updates to requirements for middleboxes

* Inclusion of requirements for security, API, and tunnel

* Move of the rationale for the update to an Annex (former section 4)

Working Group Draft 09

* Submission after second WGLC (note mistake corrected in -09).

* Clarified role of API for supporting full checksum.

* Clarified that full checksum is required in security considerations, and therefore noting that full checksum should not be treated as an attack - consistent with remainder of document.

* Added mention that API can set a mode in transport stack - to link to similar statement in RFC 2460 update.

* Fixed typos.

Working Group Draft 08

* Submission to correct unwanted removal of text from section 5 bullets 5-7 by GF.

* Replaced section 5 text with the text from 08, and reapplied the editorial correction.

* Note to reviewers: Please compare this revision with -08 used in the IETF LC).

Authors' Addresses

Godred Fairhurst
University of Aberdeen
School of Engineering
Aberdeen, AB24 3UE
Scotland, UK

Email: gorry@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk/users/gorry

Magnus Westerlund
Ericsson
Farogatan 6
Stockholm, SE-164 80
Sweden

Phone: +46 8 719 0000
Email: magnus.westerlund@ericsson.com