Internet Engineering Task Force	G. Fairhurst	
Internet-Draft	University of Aberdeen	
Intended status: Informational	M. Westerlund	
Expires: April 27, 2011	Ericsson	
	October 24, 2010	

TOC

IPv6 UDP Checksum Considerations
draft-ietf-6man-udpzero-02

Abstract

This document examines the role of the UDP transport checksum when used with IPv6, as defined in RFC2460. It presents a summary of the trade-offs for evaluating the safety of updating RFC 2460 to permit an IPv6 UDP endpoint to use a zero value in the checksum field as an indication that no checksum is present. This method is compared with some other possibilities. The document also describes the issues and design principles that need to be considered when UDP is used with IPv6 to support tunnel encapsulations. It concludes that UDP with a zero checksum in IPv6 can safely be used for this purpose, provided that this usage is governed by a set of constraints.

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 April 27, 2011.

Copyright Notice

Table of Contents

---

1. Introduction

TOC

The User Datagram Protocol (UDP) transport was defined by RFC768 (Postel, J., “User Datagram Protocol,” August 1980.) [RFC0768] for IPv4 RFC791 (Postel, J., “Internet Protocol,” September 1981.) [RFC0791] and is defined in RFC2460 (Deering, S. and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification,” December 1998.) [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 (Eggert, L. and G. Fairhurst, “Unicast UDP Usage Guidelines for Application Designers,” November 2008.) [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 IPv6 mandates use of the UDP checksum, i.e. a non-zero value, due to the lack of an IPv6 header checksum.

The lack of a possibility to use UDP with a zero-checksum in IPv6 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 checksum using IPv4. The design of IPv6 raises different issues when considering the safety of using a zero checksum for UDP with IPv6. These issues can significantly affect applications, both when an endpoint is the intended user and when an innocent bystander (received by a different endpoint to that intended). The document examines these issues and compares the strengths and weaknesses of a number of proposed solutions. This analysis presents a set of issues that must be considered and mitigated to be able to safely deploy UDP with a zero checksum over IPv6. 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 UDP with a zero checksum is the best method of the proposed alternatives to meet the goals for certain

tunnel applications. Unfortunately, this usage is expected to have some 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 for support of UDP zero checksum for IPv6 are provided to middleboxes. The document therefore derives a set of constraints required to ensure safe deployment of zero checksum in UDP. It also identifies some issues that require future consideration and possibly additional research.

1.1. Document Structure

TOC

Section 1 (Introduction) provides a background to key issues, and introduces the use of UDP as a tunnel transport protocol.
Section 2 (Standards-Track Transports) describes a set of standards-track datagram transport protocols that may be used to support tunnels.
Section 3 (Issues Requiring Consideration) discusses issues with a zero checksum in UDP for IPv6. It considers the impact of corruption, the need for validation of the path and when it is suitable to use a zero checksum.
Section 4 (Evaluation of proposal to update RFC 2460 to support zero checksum) evaluates a set of proposals to update the UDP transport behaviour and other alternatives intended to improve support for tunnel protocols. It focuses on a proposal to allow a zero checksum for this use-case with IPv6 and assess the trade-offs that would arise.
Section 5.1 (Constraints required on usage of a zero checksum) lists the constraints perceived for safe deployment of zero-checksum.
Section 6 (Summary) provides the recommendations for standardization of zero-checksum with a summary of the findings and notes remaining issues needing future work .

1.2. Background

TOC

This section provides a background on topics relevant to the following discussion.

1.2.1. The Role of a Transport Endpoint

TOC

An Internet transport endpoint should concern itself with the following issues:

*Protection of the endpoint transport state from unnecessary extra state (e.g. Invalid state from rogue packets).

*Protection of the endpoint transport state from corruption of internal state.

*Pre-filtering by the endpoint of erroneous data, to protect the transport from unnecessary processing and from corruption that it can not itself reject.

*Pre-filtering of incorrectly addressed destination packets, before responding to a source address.

1.2.2. The UDP Checksum

TOC

UDP, as defined in [RFC0768] (Postel, J., "User Datagram Protocol," August 1980.), supports two checksum behaviours when used with IPv4. The normal behaviour is for the sender to calculate a checksum over a block of data that includes a pseudo header and the UDP datagram payload. The UDP header includes a 16-bit one's complement checksum that provides a statistical guarantee that the payload was not corrupted in transit. This also allows a receiver to verify that the endpoint was the intended destination of the datagram, because the transport pseudo header covers the IP addresses, port numbers, transport payload length, and Next Header/Protocol value corresponding to the UDP transport protocol [RFC1071] (Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum," September 1988.). The length field verifies that the datagram is not truncated or padded. The checksum therefore protects an application against receiving corrupted payload data in place of, or in addition to, the data that was sent. Although the IPv4 UDP (Postel, J., "User Datagram Protocol," August 1980.) [RFC0768] checksum may be disabled, applications are recommended to enable UDP checksums [RFC5405] (Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers," November 2008.).
The network-layer fields that are validated by a transport checksum are:

*Endpoint IP source address (always included in the pseudo header of the checksum)

*Endpoint IP destination address (always included in the pseudo header of the checksum)

*Upper layer payload type (always included in the pseudo header of the checksum)

*IP length of payload (always included in the pseudo header of the checksum)

*Length of the network layer extension headers (i.e. by correct position of the checksum bytes)

The transport-layer fields that are validated by a transport checksum are:

*Transport demultiplexing, i.e. ports (always included in the checksum)

*Transport payload size (always included in the checksum)

Transport endpoints also need to verify the correctness of reassembly of any fragmented datagram. For UDP, this is normally provided as a part of the integrity check. Disabling the IPv4 checksum prevents this check. A lack of the UDP header and checksum in fragments can lead to issues in a translator or middlebox. For example, many IPv4 Network Address Translators, NATs, rely on port numbers to find the mappings, packet fragments do not carry port numbers, so fragments get dropped. RFC2765 (Nordmark, E., “Stateless IP/ICMP Translation Algorithm (SIIT),” February 2000.) [RFC2765] provides some guidance on the processing of fragmented IPv4 UDP datagrams that do not carry a UDP checksum.
IPv4 UDP checksum control is often a kernel-wide configuration control (e.g. In Linux and BSD), rather than a per socket call. There are also Networking Interface Cards (NICs) that automatically calculate TCP (Postel, J., “Transmission Control Protocol,” September 1981.) [RFC0793] and UDP checksums on transmission when a checksum of zero is sent to the NIC, using a method known as checksum offloading.

1.2.3. Differences between IPv6 and IPv4

TOC

IPv6 does not provide a network-layer integrity check. The removal of the header checksum from the IPv6 specification released routers from a need to update a network-layer checksum for each router hop as the IPv6 Hop Count is changed (in contrast to the checksum update needed when an IPv4 router modifies the Time-To-Live (TTL)).
The IP header checksum calculation was seen as redundant for most traffic (with UDP or TCP checksums enabled), and people wanted to avoid this extra processing. However, there was concern that the removal of the IP header checksum in IPv6 combined with a UDP checksum set to zero would lessen the protection of the source/destination IP addresses and

result in a significant (a multiplier of ~32,000) increase in the number of times that a UDP packet was accidentally delivered to the wrong destination address and/or apparently sourced from the wrong source address. This would have had implications on the detectability of mis-delivery of a packet to an incorrect endpoint/socket, and the robustness of the Internet infrastructure. The use of the UDP checksum is therefore required [RFC2460] (Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," December 1998.) when endpoint application s transmit UDP datagrams over IPv6.

1.3. Use of UDP Tunnels

TOC

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] (Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers," November 2008.). 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

TOC

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 (Internet draft, draft-ietf-mboned-auto-multicast-10, "Automatic IP Multicast Without Explicit Tunnels (AMT)," March 2010.) [AMT], and the Locator/Identifier Separation Protocol, LISP (Internet draft, draft-farinacci-lisp-12.txt, "Locator/ID Separation Protocol (LISP)," March 2009.) [LISP]). These protocols have proposed an update to IPv6 UDP checksum processing. These tunnel protocols could 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

TOC

It is a common requirement to terminate a large number of tunnels on a single router/host. Processing per tunnel concerns both state (memory requirements) and per-packet processing costs.
Automatic IP Multicast Without Explicit Tunnels, known as AMT (Internet draft, draft-ietf-mboned-auto-multicast-10, “Automatic IP Multicast Without Explicit Tunnels (AMT),” March 2010.) [AMT] 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 0 (see Section 6.6). It argues that the computation of an additional checksum, when an inner packet is already adequately protected, is an unwarranted burden on nodes implementing lightweight tunneling protocols. 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.

TOC

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

TOC

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 UDP zero checksum into consideration.

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 UDP datagrams with a zero checksum as valid packets. If an zero-checksum for UDP were to be allowed for IPv6, this would need firewalls to be updated before full utility of the change is available.

It can be expected that UDP with zero-checksum will initially not have the same middlebox traversal characteristics as regular UDP. However, if standardized we can expect an improvement over time of the traversal capabilities. 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

TOC

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 basis for entropy for the 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. Equal-Cost Multi-Path routing, ECMP [ECMP] (, “Using the IPv6 flow label for equal cost multipath routing in tunnels (draft-carpenter-flow-ecmp),” .)).
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

TOC

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

TOC

UDP (Postel, J., “User Datagram Protocol,” August 1980.) [RFC0768] with standard checksum behaviour is defined in RFC 2460 has already been discussed. UDP usage guidelines are provided in [RFC5405] (Eggert, L. and G. Fairhurst, “Unicast UDP Usage Guidelines for Application Designers,” November 2008.).

TOC

2.2. UDP-Lite

UDP-Lite [RFC3828] (Larzon, L-A., Degermark, M., Pink, S., Jonsson, L-E., and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)," July 2004.) 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] (Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers," November 2008.). 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. 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 also cover the insensitive part by a strong layer 2 frame CRC.

2.2.1. Using UDP-Lite as a Tunnel Encapsulation

TOC

Tunnel encapsulations can use UDP-Lite (e.g. Control And Provisioning of Wireless Access Points, CAPWAP [RFC5415] (Calhoun, P., Montemurro, M., and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification," March 2009.)), since UDP-Lite provides a transport-layer checksum, including an IP pseudo header checksum, in IPv6, without the need for a router/middelbox to traverse the entire packet payload. This provides most of the delivery verifications and still keep the complexity of the checksumming operation low. 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 full checksumming 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.

TOC

2.3. General Tunnel Encapsulations

The IETF has defined a set of tunneling protocols or network layer encapsulations, like 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

TOC

This section evaluates issues around the proposal to update IPv6 [RFC2460], to provide the option of using a UDP transport checksum set to zero. Some of the identified issues are shared with other protocols already in use.
The decision by IPv6 to omit an integrity check at the network level has meant that the transport check 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 a wrong header has not been 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] (Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification," March 2006.) 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

TOC

IP packets may be corrupted as they traverse an Internet path. Evidence has been presented [Sigcomm2000] (http://conferences.sigcomm.org/sigcomm/2000/conf/abstract/9-1.htm, "When the CRC and TCP Checksum Disagree," 2000.) 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] (Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers," July 2004.). 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 mis-delivered 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

TOC

An IP 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 this 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

TOC

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. 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] (Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," July 2003.) flows 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).

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

TOC

This section describes what happens if one or both of the UDP port values are corrupted in transit. This can also happen with IPv4 in the zero checksum case, but not when UDP checksums are enabled or with UDP-Lite. If the ports carried in the transport header of an IPv6 packet were corrupted in transit, packets may be delivered to the wrong 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

TOC

If one combines the corruption effects 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 source address and the destination port of the datagram (the source port is not always used in UDP) match those of an existing connection. 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 is given only the data associated with a 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] (Eggert, L. and G. Fairhurst, “Unicast UDP Usage Guidelines for Application Designers,” November 2008.)).

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 UDP with zero-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 using UDP zero-checksum can never assume that it is the only protocol using a zero 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

TOC

The fragmentation information in IPv6 employs a 32-bit identity field, compared to only a 16-bit filed 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-assemblied 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 requires 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 recently been clarified in "Handling of Overlapping IPv6 Fragments" (Krishnan, S., “Handling of Overlapping IPv6 Fragments,” December 2009.) [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] (Heffner, J., Mathis, M., and B. Chandler, “IPv4 Reassembly Errors at High Data Rates,” July 2007.), this is especially the case for tunnels that perform flow aggregation [I-D.ietf-intarea-tunnels] (Touch, J. and M. Townsley, “Tunnels in the Internet Architecture,” March 2010.).
Tunnel fragmentation behavior matters. There can be outer or inner fragmentation "Tunnels in the Internet Architecture" (Touch, J. and M. Townsley, “Tunnels in the Internet Architecture,” March 2010.) [I-D.ietf-intarea-tunnels]. If there is inner fragmentation by the tunnel, the outer headers will never be fragmented and thus a zero-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-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 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 checksum equal subject to this effect.

In conclusion fragmentation of packets with a zero-checksum does not worsen the situation 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. Validating the network path

TOC

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.
The zero-checksum in UDP 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 likely discard it, unless the device is updated to support a 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 negotiation or an explicit mandate to use the new behavior by all nodes intended to use a new protocol. Support along the path between end points may 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. This will require periodic validation of the path, adding complexity to any solution using the new behavior.

3.3. Applicability of method

TOC

The expectation of the present proposal defined in [UDPZ] (, “UDP Checksums for Tunneled Packets,” (Oct 2009.) is that this change would only apply to IPv6 router nodes that implement specific protocols that permit omission of UDP checksums. However, 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 also no way to identify the role of a receiver from a received packet.

Any new method would therefore need a specific applicability statement 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.
The IETF should carefully consider constraints on sanctioning the use of any new transport mode. If this is specified and 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 end-hosts could confuse a corrupted or wrongly delivered packet with that of data addressed to an application running on their endpoint unless they accept that behavior.

3.4. Impact on non-supporting devices or applications

TOC

It is important to consider what potential impact the zero-checksum behavior may have on end-points, 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 changes.
To illustrate a potential issue, consider the implications of a node that were to enable use of a zero-checksum at the interface level: This would result in all applications that listen to a UDP socket receiving datagram 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.
In contrast, the use of a zero-checksum could 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 UDP datagrams with a zero-checksum. These other applications would not be effected by this changed behavior. An application that allows the changed behavior should be aware of the risk for corruption and the increased level of misdirected traffic, and can be designed robustly to handle this risk.

---

4. Evaluation of proposal to update RFC 2460 to support zero checksum

TOC

This section evaluates 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.

4.1. Alternatives to the Standard Checksum

TOC

There are several alternatives to the normal method for calculating the UDP Checksum 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] (Rijsinghani, A., “Computation of the Internet Checksum via Incremental Update,” May 1994.), 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] (Mallory, T. and A. Kullberg, “Incremental updating of the Internet checksum,” January 1990.). 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 [UDPTT] (, “The UDP Tunnel Transport mode,” Feb 2010.) 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 packet (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[UDPZ] (, “UDP Checksums for Tunneled Packets,” (Oct 2009.). This requires no 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.

4.2. Comparison

TOC

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

4.2.1. Middlebox Traversal

TOC

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 may also 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. The methods that ignores the

checksum has an additional downside in that middlebox traversal can not be improved, because there is no way to identify which packets 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.
Packets using UDP with a zero 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 the zero checksum packets.
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.

4.2.2. Load Balancing

TOC

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 likely to not be supported by deployed hashing mechanisms, which may 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.

4.2.3. Ingress and Egress Performance Implications

TOC

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 zero-checksum method, ignoring the 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 packet. Regular UDP is clearly the most costly to process, always requiring checksum calculation over the entire packet.
It is important to note that the zero-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.

4.2.4. Deployability

TOC

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 tunneling 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 Zero-checksum with or without an Extension header require support at both end-points and in middleboxes. UDP-Lite, UDPTT, and Zero-checksum and Extension header may additionally require changes or additions to the tunneling control protocol to negotiate support and path validation.

TOC

4.2.5. 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 Zero-checksum, ignoring the checksum on reception and IP-and-IP encapsulation provide no verification that a received packet was intended to be processed by a specific tunnel egress or that the inner packet was correct.

4.2.6. Comparison Summary

TOC

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:** 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 and validation.

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.
>
> **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 and validation.

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 application level negotiation and validation.

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 application level negotiation and validation.

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 application level negotiation and validation.

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. UDP Zero-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 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 zero-checksum becomes standardized the fewer boxes will be non-compliant.
Thus, the question of whether to allow UDP with a zero-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 zero-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 authors do think the answer to the above questions are such that zero-checksum should be standardized for use by tunnel encapsulations.

---

5. Requirements on the specification of transported protocols

TOC

5.1. Constraints required on usage of a zero checksum

TOC

If a zero checksum approach were to be adopted by the IETF, the specification should consider adding the following constraints on usage:

1. IPv6 protocol stack implementations should not by default allow the new method. The default node receiver behaviour must discard all IPv6 packets carrying UDP packets with a zero checksum.

2. Implementations must provide a way to signal the set of ports that will be enabled to receive UDP datagrams with a zero checksum. An IPv6 node that enables reception of UDP packets with a zero-checksum, must enable this only for a specific port or port-range. This may be implemented via a socket API call, or similar mechanism.

3. RFC 2460 specifies that IPv6 nodes should log UDP datagrams with a zero-checksum. This should remain the case for any datagram received on a port that does not explicitly enable zero-checksum processing. A port for which zero-checksum has been enabled must not log the datagram.

4. A stack may separately identify UDP datagrams that are discarded with a zero checksum. It should not add these to the standard log, since the endpoint has not been verified.

5. Tunnels that encapsulate IP may rely on the inner packet integrity checks provided that the tunnel will not significantly increase the rate of corruption of the inner IP packet. If a significantly increased corruption rate can occur, then the tunnel must provide an additional integrity verification mechanism. An integrity mechanisms is always

recommended at the tunnel layer to ensure that corruption rates of the inner most packet are not increased.

6. Tunnels that encapsulate Non-IP packets must have a CRC or other mechanism for checking packet integrity, unless the Non-IP packet specifically is designed for transmission over lower layers that do not provide any packet integrity guarantee. In particular, the application must be designed so that corruption of this information does not result in accumulated state or incorrect processing of a tunneled payload.

7. UDP applications that support use of a zero-checksum, should not rely upon correct reception of the IP and UDP protocol information (including the length of the packet) when decoding and processing the packet payload. In particular, the application must be designed so that corruption of this information does not result in accumulated state or incorrect processing of a tunneled payload.

8. If a method proposes recursive tunnels, it needs to provide guidance that is appropriate for all use-cases. Restrictions may be needed to the use of a tunnel encapsulations and the use of recursive tunnels (e.g. Necessary when the endpoint is not verified).

9. 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).

Deployment of the new method needs to remain restricted to endpoints that explicitly enable this mode and adopt the above procedures. Any middlebox that examines or interact with the UDP header over IPv6 should support the new method.

---

6. Summary

TOC

This document examines the role of the transport checksum when used with IPv6, as defined in RFC2460.
It presents a summary of the trade-offs for evaluating the safety of updating RFC 2460 to permit an IPv6 UDP endpoint to use a zero value in the checksum field to indicate that no checksum is present. A decision not to include a UDP checksum in received IPv6 datagrams could impact a tunnel application that receives these packets. However, a well-designed tunnel application should include consistency checks to

validate any header information encapsulated with a packet. In most cases tunnels encapsulating IP packets can rely on the inner packets own integrity protection. When correctly implemented, such a tunnel endpoint will not be negatively impacted by omission of the transport-layer checksum. Recursive tunneling and fragmentation is a potential issues that can raise corruption rates significantly, and requires careful consideration.

Other applications at the intended destination node or another IPv6 node can be impacted if they are allowed to receive datagrams without a transport-layer checksum. It is particularly important that already deployed applications are not impacted by any change at the transport layer. If these applications execute on nodes that implement RFC 2460, they will reject all datagrams with a zero UDP checksum, thus this is not an issue. For nodes that implement support for zero-checksum it is important to ensure that only UDP applications that desire zero-checksum can receive and originate zero-checksum packets. Thus, the enabling of zero-checksum needs to be at a port level, not for the entire host or for all use of an interface.

The implications on firewalls, NATs and other middleboxes need to be considered. 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. Thus, relatively few current middleboxes may actually block IPv6 UDP with a zero checksum.

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 applications data stream 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 use of UDP with no checksum has merits for some applications, such as tunnel encapsulation, and is widely used in IPv4. However, there are dangers for IPv6: There is a bigger risk of corruption and miss-delivery when using zero-checksum in IPv6 compared to IPv4 due to the removed IP header checksum. Thus, applications needs to make a new evaluation of the risks of enabling a zero-checksum. Some applications will need to re-consider their usage of zero-checksum, and possibly 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. Tunnel applications using UDP for encapsulation can in many case use zero-checksum without significant impact on the corruption rate. In some cases, the use of checksum off-loading may help alleviate the checksum processing cost.

Recursive tunneling and fragmentation is a difficult issue relating to tunnels in general. 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 any

verification of correctness. This issue requires future thought and consideration.
The conclusion is that UDP zero checksum in IPv6 should be standardized, as it satisfies usage requirements that are currently difficult to address. We do note that a safe deployment of zero-checksum will need to follow a set of constraints listed in Section 5.1 (Constraints required on usage of a zero checksum).

---

7. Acknowledgements

TOC

Brian Haberman, Brian Carpenter, Magaret Wasserman, Lars Eggert, others in the TSV directorate.
Thanks also to: Rémi Denis-Courmont, Pekka Savola and many others who contributed comments and ideas via the 6man, behave, lisp and mboned lists.

---

8. IANA Considerations

TOC

This document does not require IANA considerations.

---

9. Security Considerations

TOC

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.

---

10. References

TOC

10.1. Normative References

TOC

[RFC0791]	Postel, J., “Internet Protocol,” STD 5, RFC 791, September 1981 (TXT).
[RFC0793]	Postel, J., “Transmission Control Protocol,” STD 7, RFC 793, September 1981 (TXT).
[RFC1071]	

	Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum," RFC 1071, September 1988 (TXT).
[RFC2460]	Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification," RFC 2460, December 1998 (TXT, HTML, XML).

10.2. Informative References

TOC

[AMT]	Internet draft, draft-ietf-mboned-auto-multicast-10, "Automatic IP Multicast Without Explicit Tunnels (AMT)," March 2010.
[ECMP]	"Using the IPv6 flow label for equal cost multipath routing in tunnels (draft-carpenter-flow-ecmp)."
[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 (TXT).
[LISP]	Internet draft, draft-farinacci-lisp-12.txt, "Locator/ID Separation Protocol (LISP)," March 2009.
[RFC0768]	Postel, J., "User Datagram Protocol," STD 6, RFC 768, August 1980 (TXT).
[RFC1141]	Mallory, T. and A. Kullberg, "Incremental updating of the Internet checksum," RFC 1141, January 1990 (TXT).
[RFC1624]	Rijsinghani, A., "Computation of the Internet Checksum via Incremental Update," RFC 1624, May 1994 (TXT).
[RFC2765]	Nordmark, E., "Stateless IP/ICMP Translation Algorithm (SIIT)," RFC 2765, February 2000 (TXT).
[RFC3550]	Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," STD 64, RFC 3550, July 2003 (TXT, PS, PDF).
[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 (TXT).
[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 (TXT).
[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 (TXT).

[RFC4963]	Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates," RFC 4963, July 2007 (TXT).
[RFC5405]	Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers," BCP 145, RFC 5405, November 2008 (TXT).
[RFC5415]	Calhoun, P., Montemurro, M., and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP) Protocol Specification," RFC 5415, March 2009 (TXT).
[RFC5722]	Krishnan, S., "Handling of Overlapping IPv6 Fragments," RFC 5722, December 2009 (TXT).
[Sigcomm2000]	http://conferences.sigcomm.org/sigcomm/2000/conf/abstract/9-1.htm, "When the CRC and TCP Checksum Disagree," 2000.
[UDPTT]	"The UDP Tunnel Transport mode," Feb 2010.
[UDPZ]	"UDP Checksums for Tunneled Packets," (Oct 2009.

Appendix A. Document Change History

TOC

{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.
>
> **TO BE DONE ** This version requires review from proponents and opponents to the UDP zero checksum proposal.
>
> *

Authors' Addresses

TOC

	Godred Fairhurst
	University of Aberdeen
	School of Engineering
	Aberdeen, AB24 3UE,
	Scotland, UK
Phone:	
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
Fax:	
Email:	magnus.westerlund@ericsson.com
URI:	