Internet Engineering Task Force (IETF)
Request for Comments: 6936
Category: Standards Track
ISSN: 2070-1721

G. Fairhurst
University of Aberdeen
M. Westerlund
Ericsson

April 2013

Applicability Statement for the Use of IPv6 UDP Datagrams with Zero Checksums

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 it examines the role of the IPv6 UDP transport checksum. The document also identifies issues and constraints for deployment on network paths that include middleboxes. An appendix presents a summary of the trade-offs that were considered in evaluating the safety of the update to RFC 2460 that changes the use of the UDP checksum with IPv6.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6936.

Table of Contents

1. Introduction

The User Datagram Protocol (UDP) [RFC0768] transport is defined for IPv4 [RFC0791], and it 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 applications do need to provide many important transport functions on top of UDP. The UDP usage guidelines [RFC5405] provide 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. The inclusion of the pseudo-header in the checksum computation provides a statistical check that datagrams have been delivered to the intended IPv6 destination node. Algorithms for checksum computation are described in [RFC1071].

The inability to use an IPv6 datagram with a zero UDP checksum has been found to be 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, whether an endpoint is the intended user or an innocent bystander (i.e., when a packet is received by a different endpoint to that intended).

This document identifies a set of issues that must be considered and mitigated to enable safe deployment of IPv6 applications that use a zero UDP checksum. The appendix compares the strengths and weaknesses of a number of proposed solutions. The comparison of methods provided in this document is also expected to be useful when considering applications that have different goals from the ones whose needs led to the writing of this document, especially applications that can use existing standardized transport protocols. The analysis concludes that using a zero UDP checksum is the best method of the proposed alternatives to meet the goals of 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 decrease as updates are provided in middleboxes that support the zero UDP

checksum for IPv6. Therefore, in this document, we derive a set of constraints required to ensure safe deployment of a zero UDP checksum.

Finally, the document 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 the remaining issues that need future work.

Appendix A evaluates the set of proposals to update the UDP transport behavior and other alternatives intended to improve support for tunnel protocols. It concludes by assessing the trade-offs of the various methods and by 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 they 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] and Locator/Identifier Separation Protocol (LISP) [RFC6830]). These protocols provided several motivations to update IPv6 UDP checksum processing so that it would benefit from simpler checksum processing, including:

- Reducing forwarding costs, motivated by redundancy present in the encapsulated packet header, because in tunnel encapsulations, payload integrity and length verification may be provided by higher-layer encapsulations (often using the IPv4, UDP, UDP-Lite [RFC3828], or TCP checksums [RFC0793]).

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

- Enhancing the ability to traverse and function with middleboxes.

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

1.3.2. Reducing Forwarding Costs

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

Automatic IP Multicast Tunneling, known as AMT [AMT], currently specifies UDP as the transport protocol for packets carrying tunneled IP multicast packets. The current specification for AMT states that the UDP checksum in the outer packet header should be zero (see Section 6.6 of [AMT]). That section 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; therefore, a different pseudo-header must be built to form the header for each tunnel egress that receives replicated multicast packets.

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. For example, there are implementations that have optimized 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, for certain classes of tunnel endpoints, this off-loading is not available and is 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 provides only the first n bytes of a packet to the forwarding engine, where typically n <= 128.

When this design is used to support a tunnel ingress and egress, it prevents fast processing of a transport checksum over an entire (large) packet. Hence, the currently defined IPv6 UDP checksum is poorly suited for use within a router that is unable to access the entire packet and does not provide checksum off-loading. Thus, enabling checksum calculation over the complete packet can impact router design, performance, energy consumption, and cost.

1.3.4. Interactions with Middleboxes

Many paths in the Internet include one or more middleboxes of various types. Large classes of middleboxes will handle zero UDP checksum packets, but do not support UDP-Lite or the other investigated proposals. These middleboxes include load balancers (see Section 1.3.5) including equal-cost multipath (ECMP) routing, traffic classifiers, and other functions that reads some fields in the UDP headers but does not validate the UDP checksum.

There are also middleboxes that either validate or modify the UDP checksum. The two most common classes are firewalls and NATs. In IPv4, UDP encapsulation may be desirable for NAT traversal, because 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, it hopefully will 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 becomes 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 becomes 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 utilize the IPv6 flow label [RFC6437] as a basis for entropy for load balancing. This would have the desirable effect of freeing IPv6 load-balancing devices from the need to assume semantics for the use of the transport port field, and also, it works for all types of transport protocols.

This use of the Flow Label for load balancing is consistent with the intended use, although further clarity was needed to ensure the field can be consistently used for this purpose. Therefore, an updated IPv6 flow label [RFC6437] and ECMP routing [RFC6438] usage were specified. 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, the flow label value could be copied to the outer packet header.

However, many current endpoints 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 has been clarified, there will be a transition time before a significant proportion of endpoints start to assign a good quality flow label to the flows that they originate. The use of load balancing using the transport header fields would continue 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 is also a set of network-layer encapsulation tunnels, such as IP-in-IP and Generic Routing Encapsulation (GRE). These solutions, which are already standardized, are discussed first, before discussing the issues, because they provide background for the description of the issues and allow some comparison with existing issues.

2.1. UDP with Standard Checksum

UDP [RFC0768] with standard checksum behavior, 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 and is specified as a proposed standard, RFC 3828. A MIB is defined in [RFC5097], and unicast usage guidelines are defined in [RFC5405]. There has been at least one open-source implementation of UDP-Lite as a part of the Linux kernel since version 2.6.20.

UDP-Lite provides a checksum with an option for 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 or 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 it 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 Cyclic Redundancy Check (CRC) that covers the sensitive part.

2.2.1. Using UDP-Lite as a Tunnel Encapsulation

Tunnel encapsulations, such as Control And Provisioning of Wireless Access Points (CAPWAP) [RFC5415], can use UDP-Lite, because it 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 verify only delivery of the tunneled payload and uses a calculated checksum for control information.

Currently, support for middlebox traversal using UDP-Lite is poor, because UDP-Lite uses a different IPv6 network-layer Next Header value than that used for UDP; therefore, 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 as UDP-Lite has achieved better support in middleboxes and endpoints.

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, because tunnel encapsulations are typically layered directly over the Internet layer (identified by the upper layer type in the IPv6 Next Header field) and because they are not used as endpoint transport protocols. There is little chance of confusing a tunnel-encapsulated packet with other application data. Such confusion could result in corruption of application state or data.

From an end-to-end perspective, the principal difference between an endpoint transport and a tunnel encapsulation is the value of the network-layer Next Header field. In the former, it identifies a transport protocol that supports endpoint applications. In the latter, it identifies a tunnel protocol egress. This separation of function reduces the probability that corruption of a tunneled packet could result in the packet being erroneously delivered to an

application. Specifically, packets are delivered only 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.

2.4. Relationship of Zero UDP Checksum to UDP-Lite and UDP with Checksum

The operation of IPv6 with UDP with a zero checksum is not the same as IPv4 with UDP with a zero checksum. Protocol designers should not be fooled into thinking that the two are the same. The requirements below list a set of additional considerations for IPv6.

Where possible, existing general tunnel encapsulations, such as GRE and IP-in-IP, should be used. This section assumes that such existing tunnel encapsulations do not offer the functionally required to satisfy the protocol designer's goals. This section considers the standardized alternative solutions rather than the full set of ideas evaluated in Appendix A. The alternatives to UDP with a zero checksum are UDP with a (calculated) checksum and UDP-Lite.

UDP with a checksum has the advantage of close to universal support in both endpoints and middleboxes. It also provides statistical verification of delivery to the intended destination (address and port). However, some classes of device have limited support for calculation of a checksum that covers a full datagram. For these devices, this limited support can incur significant processing costs (e.g., requiring processing in the router's slow path) and hence can reduce capacity or fail to function.

UDP-Lite has the advantage of using a checksum that can be calculated only over the pseudo-header and the UDP header. This provides a statistical verification of delivery to the intended destination (address and port). The checksum can be calculated without access to the datagram payload, requiring access only to the part that is to be protected. A drawback is that UDP-Lite currently has limited support in both endpoints (i.e., is not supported on all operating system platforms) and middleboxes (which must support the UDP-Lite header type). Therefore, using a path verification method is recommended.

IPv6 and UDP with a zero checksum can also be used by nodes that do not permit calculation of a payload checksum. Many existing classes of middleboxes do not verify or change the transport checksum. For these middleboxes, IPv6 with a zero UDP checksum is expected to function where UDP-Lite would not. However, support for the zero UDP checksum in middleboxes that do change or verify the checksum is

currently limited, and this may result in datagrams with a zero UDP checksum being discarded. Therefore, using a path verification method is recommended.

For some sets of constraints, no solution exists. For example, a protocol designer who needs to originate or receive datagrams on a device that cannot efficiently calculate a checksum over a full datagram and also needs these packets to pass through a middlebox that verifies or changes a UDP checksum, but that does not support a zero UDP checksum, cannot use the zero UDP checksum method. Similarly, a protocol designer who needs to originate datagrams on a device with UDP-Lite support, but needs the packets to pass through a middlebox that does not support UDP-Lite, cannot use UDP-Lite. For such cases, there is no optimal solution. The current recommendation is to use or fall back to using UDP with full checksum coverage.

3. Issues Requiring Consideration

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

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

- That the endpoint address was not corrupted within a router, i.e., a packet was intended to be received by this destination, and 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 of these 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. Older evidence presented in "When the CRC and TCP Checksum Disagree" [Sigcomm2000] shows that this was an issue with IPv4 routers in the year 2000 and that 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]. During the development of this document in 2009, a number of individuals provided reports of observed rates for received UDP datagrams using IPv4 where the UDP checksum had been detected as corrupt. These rates were as high as 1.39E-4 for some paths, but close to zero for other paths.

There is extensive experience with deployments using tunnel protocols in well-managed networks (e.g., corporate networks and service provider core networks). This has shown the robustness of methods such as Pseudowire Emulation Edge-to-Edge (PWE3) and MPLS that do not employ a transport protocol checksum and that have not specified mechanisms to protect from corruption of the unprotected headers (such as the VPN Identifier in MPLS). Reasons for the robustness may include:

- A reduced probability of corruption on paths through well-managed networks.

- IP forms the majority of the inner traffic carried by these tunnels. Hence, from a transport perspective, endpoint verification is already being performed when a received IPv4 packet is processed or by the transport pseudo-header for an IPv6 packet. This update to UDP does not change this behavior.

- In certain cases, a combination of additional filtering (e.g., filtering a MAC destination address in a Layer 2 tunnel) significantly reduces the probability of final misdelivery to the IP stack.

- The tunnel protocols did not use a UDP transport header. Therefore, any corruption is unlikely to result in misdelivery to another UDP-based application. This concern is specific to UDP with IPv6.

While this experience can guide the present recommendations, any update to UDP must preserve operation in the general Internet, which is heterogeneous and can include links and systems of widely varying characteristics. Transport protocols used by hosts need to be designed with this in mind, especially when there is need to traverse edge networks, where middlebox deployments are common.

Currently, for the general Internet, there is no evidence that corruption is rare, nor is there evidence that corruption in IPv6 is rare. Therefore, it seems prudent not to relax checks on misdelivery. The emergence of low-end IPv6 routers and the proposed use of NAT with IPv6 provide further motivation to protect from misdelivery.

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.

Using a checksum 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 effect of modifications to the destination and source IP address fields, the port fields, and the fragmentation information.

3.1.1. Corruption of the Destination IP Address Field

An IPv6 endpoint destination address could be modified in the network; for example, it could be 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. When using IPv6, however, such modification in the network cannot be detected at

the network layer. Detection of this corruption by a UDP receiver relies on the IPv6 pseudo-header that is incorporated in the transport checksum.

There are two possible outcomes:

- Delivery to a destination address that is not in use. The packet will not be delivered, but an error report could be generated.

- Delivery to a different destination address. This modification will normally be detected by the transport checksum, resulting in a 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 Section 3.1.4 on port processing.)

3.1.2. Corruption of the Source IP Address Field

This section examines what happens when the source IP 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. Detection of this corruption by a UDP receiver relies on the IPv6 pseudo-header that is incorporated in the transport checksum.

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 from 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 pre-established context may disregard the datagram as invalid or could map it to another context (if a context for the modified source address were 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. Therefore, reception of a datagram with a corrupted source address will result in the 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 the one 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 when 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 are corrupted in transit, packets may be delivered to the wrong application process (on the intended machine), responses or errors may be sent to the wrong application process (on the intended machine), or both may occur.

3.1.4. Delivery to an Unexpected Port

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

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

- The packet could be delivered to a different node that implements the same application, so the packet may be accepted, but side effects could occur or accumulated state could be generated.

- The packet could be delivered to an application that does not implement the tunnel protocol, so the packet may be incorrectly parsed and may be misinterpreted, causing side effects or generating 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 corrupted 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, Datagram Congestion Control Protocol (DCCP), and Stream Control Transmission Protocol (SCTP)). 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 wildcard values in endpoint identifiers and do not always validate the 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 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 it is 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 cannot assume that it is the only protocol using a zero UDP checksum. Therefore, it needs to handle misdelivery gracefully. It must be robust when malformed

packets are received on a listening port, and it must 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 whether there are more fragments. Corruption of any of these fields 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, so it 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 of the packet incomplete, unless that packet has been duplicated before the corruption. The incomplete packet will eventually be timed out and discarded.

- Erroneous reassembly: The reassembled 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, completing the reassembly is much less probable, because this would require consistent corruption of the IPv6 header's payload length and offset fields. To prevent erroneous assembly, the reassembling stack must provide strong checks that detect overlap and 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 provided by the transport checksum. The Upper-Layer Packet length field included in the pseudo-header assists in verifying correct reassembly, because 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 data that is all-zero in a chunk that is a multiple of 16 bits.

The most significant risk of corruption results following mis-association of a fragment with a different packet. This risk can be significant, because the size of fragments is often the same (e.g., fragments that form when the path MTU results in fragmentation of a larger packet, which is common when addition of a tunnel encapsulation header increases the size of a packet). Detection of this type of error requires a checksum or other integrity check of the headers and the payload. While such protection is desirable for tunnel encapsulations using IPv4, because the small fragmentation ID can easily result in wraparound [RFC4963], this is especially desirable for tunnels that perform flow aggregation [TUNNELS].

Tunnel fragmentation behavior matters. There can be outer or inner fragmentation tunnels in the Internet Architecture [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 detect reassembly anomalies only 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. The following list describes some tunnel fragmentation behaviors:

- 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 fragmentation of the encapsulated packet, unless the fragmentation is performed on the innermost IP header.

- Unless there is verification at each reassembly, the probability of undetected errors will increase with the number of times fragmentation is recursively applied, making both IP-in-IP and UDP with zero UDP checksum 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 that offers no 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 CRC that reduces the probability for corrupted packets being forwarded, but there still exists a probability that errors may propagate undetected.

Unfortunately, the Internet community lacks reliable information to identify the most common functions or equipment that results in packet corruption. However, there are indications that the place where corruption occurs can vary significantly from one path to another. However, there is a risk in taking evidence from one usage domain and using it to infer characteristics for another. Methods intended for general Internet usage must therefore assume that corruption can occur, and mechanisms must be deployed to mitigate the effects of corruption and any 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, thus changing 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 RFC 2460. Thus, it may be expected that any device on the path that has a reason to look beyond the IP header, for example, to validate the UDP checksum, will consider such a packet as erroneous or illegal and may discard it, unless the device is updated to support the new behavior. Any middlebox that modifies the UDP checksum, for example,

a NAT that changes the values of the IP and UDP header in such a way that the checksum over the pseudo-header changes value, will need to be updated to support this behavior. Until then, a zero UDP checksum packet is likely to be discarded, either directly in the middlebox or at the destination, when a zero UDP checksum has been modified to be non-zero by an incremental update.

A pair of endpoints intending to use the new behavior will therefore need not only to ensure support at each endpoint, but also to ensure 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., a firewall or NAT) 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 endpoints can be guaranteed in limited deployments by appropriate configuration. In general, it can be expected to take time for deployment of any updated behavior 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 should be able to renegotiate. 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 behavior will require periodic validation of the path, adding complexity to any solution using the new behavior.

3.4. Applicability of the Zero UDP Checksum Method

The update to the IPv6 specification defined in [RFC6935] modifies only IPv6 nodes that implement specific protocols designed to permit omission of a UDP checksum. This document provides an applicability statement for the updated method, indicating when the mechanism can (and cannot) be used. Enabling a zero UDP checksum, 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 zero UDP checksum method is widely available, we expect that it will be used by applications that perceive to gain benefit from it. Any solution that uses an end-to-end transport protocol rather than an IP-in-IP encapsulation needs to minimize 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.

A protocol or application that uses the zero UDP checksum method must ensure that the lack of checksum does not affect the protocol operation. This includes being robust to receiving an 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 endpoint devices and applications that are not modified to support the new behavior or, by default or preference, do not 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.

Therefore, a zero UDP checksum 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 for the implementation of IPv6 nodes that support the use of a zero value in the checksum field of a UDP datagram.

All implementations that support the 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 off-loading to insert an updated UDP checksum value in all UDP datagrams that it forwards. Note, however, that sending a calculated checksum requires the receiver to also perform the checksum calculation. Checksum off-loading 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 by 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 keepalive datagrams), not just when a socket is established in zero checksum mode.

5. The default IPv6 node receiver behavior MUST be to 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 by a 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., the encapsulating ingress, may choose to compute the UDP checksum or may calculate it 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. They SHOULD NOT add these to the standard log, because 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., a tunnel) that does not perform a UDP checksum calculation to verify the integrity at the transport endpoints. Before deciding to use the zero UDP checksum and lose the integrity verification provided by non-zero checksumming, a protocol developer should seriously consider if they can use checksummed UDP packets or UDP-Lite [RFC3828], because IPv6 with a zero UDP checksum is not equivalent in behavior to IPv4 with zero UDP checksum.

The requirements and recommendations for protocols and tunnel encapsulations using an IPv6 transport flow that does not perform a UDP checksum calculation to verify the integrity at the transport endpoints are:

1. Transported protocols that enable the use of zero UDP checksum MUST enable this only 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 are not increased (e.g., at the innermost packet of a UDP tunnel). A mechanism that isolates the causes of corruption (e.g., identifying misdelivery, IPv6 header corruption, or tunnel header corruption) is also expected to 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, 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 the use of a zero UDP checksum MUST be designed so that corruption of any header information does not result in accumulation of incorrect 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, because 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 keepalive messages with a zero UDP checksum. An endpoint that discovers an appreciable loss rate for keepalive packets MAY terminate the UDP flow (e.g., a 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, keepalive 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 the 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 a 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 a 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.)

Special considerations are required when designing a UDP tunnel protocol where the tunnel ingress or egress may be a router that may not have access to the packet payload. When the node is acting as a host (i.e., sending or receiving a packet addressed to itself), the checksum processing is similar to other hosts. However, when the node (e.g., a router) is acting as a tunnel ingress or egress that forwards a packet to or from a UDP tunnel, there may be restricted access to the packet payload. This prevents calculating (or verifying) a UDP checksum. In this case, the tunnel protocol may use a zero UDP checksum and must:

- Ensure that tunnel ingress and tunnel egress router are both configured to use a zero UDP checksum. For example, this may include ensuring that hardware checksum off-loading is disabled.

- The tunnel operator must ensure that middleboxes on the network path are updated to support use of a zero UDP checksum.

- A tunnel egress should implement appropriate security techniques to protect from overload, including source address filtering to prevent traffic injection by an attacker and rate-limiting of any packets that incur additional processing, such as UDP datagrams used for control functions that require verification of a

calculated checksum to verify the network path. Usage of common control traffic for multiple tunnels between a pair of nodes can assist in reducing the number of packets to be processed.

6. Summary

This document provides an applicability statement for the use of UDP transport checksums with IPv6.

It examines the role of the UDP transport checksum when used with IPv6 and 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.

Application designers should first examine whether their transport goals may be met using standard UDP (with a calculated checksum) or UDP-Lite. 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, application designers need to 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 cost of checksum processing 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 the omission of the transport-layer checksum. Recursive tunneling and fragmentation are potential issues that can raise corruption rates significantly, and they require careful consideration.

Other UDP applications at the intended destination node or another node can be impacted if the nodes 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, to protect both 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 by 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 innermost packet when fragmentation occurs across 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.

Any use of the updated method must consider the implications for firewalls, NATs, and other middleboxes. It is not expected that IPv6 NATs will handle IPv6 UDP datagrams in the same way that they handle IPv4 UDP datagrams. In many deployed cases, an update to support an IPv6 zero UDP checksum will be required. Firewalls are intended to be configured, and therefore, they may need to be explicitly updated to allow new services or protocols. Deployment of IPv6 middleboxes 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 should consider whether other standard methods may be more appropriate and may simplify application design.

7. Security Considerations

Transport checksums provide the first stage of protection for the stack, although they cannot be considered authentication mechanisms. These checks are also desirable to ensure that packet counters correctly log actual activity, and they can be used to detect unusual behaviors.

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. This can become a security risk for designs that can handle a significantly larger number of packets with zero UDP checksums compared to datagrams with a non-zero checksum, such as a tunnel egress. An attacker could attempt to inject non-zero checksummed UDP packets into a tunnel that is forwarding zero checksum UDP packets and cause overload in the

processing of the non-zero checksums, e.g., if it happens in a router's slow path. Protection mechanisms should therefore be employed when this threat exists. Protection may include source-address filtering to prevent an attacker from injecting traffic, as well as throttling the amount of non-zero checksum traffic. The latter may impact the functioning of the tunnel protocol.

Transmission of IPv6 packets with a zero UDP checksum could reveal additional information to help an on-path attacker identify the operating system or configuration of a sending node. There is a need to probe the network path to determine whether the current path supports the use of IPv6 packets with a zero UDP checksum. The details of the probing mechanism may differ for different tunnel encapsulations, and if they are visible in the network (e.g., if not using IPsec in encryption mode), they could reveal additional information to help an on-path attacker 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, because they present a large attack surface. This applicability statement therefore permits this method to be enabled only for specific port ranges.

When the zero UDP checksum mode is enabled for a range of ports, nodes and middleboxes must forward received UDP datagrams that have either a calculated checksum or a zero checksum.

8. Acknowledgments

We would like to thank Brian Haberman, Brian Carpenter, Margaret Wasserman, Lars Eggert, and others in the TSV directorate. Barry Leiba, Ronald Bonica, Pete Resnick, and Stewart Bryant helped to make this document one with 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.

9. References

9.1. Normative References

[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.

[RFC6935] Eubanks, M., Chimento, P., and M. Westerlund, "IPv6 and UDP Checksums for Tunneled Packets", RFC 6935, April 2013.

9.2. Informative References

[AMT] Bumgardner, G., "Automatic Multicast Tunneling", Work in Progress, June 2012.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[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.

[RFC5097] Renker, G. and G. Fairhurst, "MIB for the UDP-Lite protocol", RFC 5097, January 2008.

[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.

[RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, January 2013.

[Sigcomm2000] Stone, J. and C. Partridge, "When the CRC and TCP Checksum Disagree", 2000, <http://conferences.sigcomm.org/sigcomm/2000/conf/abstract/9-1.htm>.

[TUNNELS] Touch, J. and M. Townsley, "Tunnels in the Internet Architecture", Work in Progress, March 2010.

[UDPTT] Fairhurst, G., "The UDP Tunnel Transport mode", Work in Progress, February 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] such that it provides the option that some nodes may suppress generation and checking of the UDP transport checksum. It also compares this 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:

- IP-in-IP tunneling. Because 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 there are no ports to perform demultiplexing among different tunnel types. This reduces the available information upon which a load balancer may act.

- 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 minimize 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.

- Delta computation of the checksum from an encapsulated checksum field. Because 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 to be performed on each packet. The method would work only 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 has been modified to disable checksum processing (Zero UDP Checksum) [RFC6935]. This eliminates the need for a checksum calculation, but would require constraints on appropriate usage and updates to endpoints and middleboxes.

- The proposed UDP Tunnel Transport [UDPTT] protocol 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 minimize per-packet processing.

- 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 endpoint, 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).

- 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.

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

A.2. Comparison of Alternative Methods

This section compares the methods listed above to support datagram tunneling. It includes proposals for updating the behavior 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. From a received packet, there is no way to identify the role of the receiving node.

A.2.1. Middlebox Traversal

Regular UDP with a standard checksum or the delta-encoded optimization for creating correct checksums has the best possibility 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 be able to traverse a middlebox with this behavior. However, a middlebox on the path that attempts to verify a standard checksum will not forward packets using either of these methods, thus preventing traversal. A method that ignores the checksum has the additional downside that it prevents improvement of middlebox traversal, because there is no way to identify UDP datagrams that use the modified checksum behavior.

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, because 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 types of middleboxes, because it requires support for a separate network-layer protocol number. Once enabled, the method to support incremental checksum updates 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 behavior that is equally as good as UDP. However, UDP-Lite is currently unlikely to be supported by deployed hashing mechanisms, which could cause a load balancer not to use the transport header in the computed hash. A load balancer that uses only the IP header will have low entropy, but this could be improved by including the IPv6 the flow label, provided 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 have low data overhead. The other proposals introduce a UDP-like header, which incurs an associated data overhead. Processing is minimized for the method that uses a zero UDP checksum and for the method that ignores the UDP checksum on reception, and processing is 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 endpoints, a need for negotiation, and the need to update middleboxes. These are summarized 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 needs changes only in the tunnel ingress.

- IP-in-IP tunnels should not require changes to the endpoints, but they raise issues regarding the traversal of firewalls and other security devices, which are expected to require updates.

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

- The remaining solutions (including the zero UDP checksum method) offer similar deployability. UDP-Lite requires support at both endpoints and in middleboxes. UDPTT and the zero UDP checksum method, with or without an extension header, require support at

both endpoints and in middleboxes. UDP-Lite, UDPTT, and the zero UDP checksum method and the use of extension headers may also 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 they do not verify the inner packet. They provide only 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. Section 3.1 discusses experience using specific protocols in well-managed networks.

A.2.6. Comparison Summary

The comparisons above may be summarized as, "there is no silver bullet that will slay all the issues". One has to select which downsides can best be lived with. Focusing on the existing solutions, they can be summarized as:

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

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

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

Delta computation of a tunnel checksum: The delta checksum is an optimization in the processing of UDP, and, 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 that has good multiplexing support, limited middlebox traversal that could improve over time, and good load-balancing support, and that, in most cases, requires application-level negotiation and validation to confirm that the path forwards a zero UDP checksum.

UDPTT: A medium-complexity encapsulation that has good multiplexing support, limited middlebox traversal that may possibly improve over time, and good load-balancing support, and that, in most cases, requires 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 encapsulation that has no multiplexing support, limited middlebox traversal, and poor load-balancing support that could improve over time, and that, in most cases, requires negotiation to confirm that the option is supported and validation to confirm the path forwards the option.

IPv6 Destination Option Combined with Zero UDP Checksum: A medium-complexity encapsulation that has good multiplexing support, limited load-balancing support that could improve over time, and that, in most cases, requires 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 that has good multiplexing support, medium middlebox traversal that can never improve, and good load-balancing support, and that, in most cases, requires negotiation to confirm that 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, 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, if low complexity is necessary, and one does not require load balancing, IP-in-IP tunneling is the simplest. If one wants strengthened error detection, but with the 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, provided that the usage can accept the currently limited support for 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, because it requires both load balancer and endpoint 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 a long time to evolve.

However, we note that the deployment of IPv6-capable middleboxes is still in its initial phase, and if a new method becomes standardized quickly, 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 best viewed as a trade-off among a number of more subjective questions:

- Is there sufficient interest in using a zero UDP checksum with the given constraints (summarized 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 answers to the above questions are sufficient to update IPv6 to standardize 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 the method defined in RFC 2460.

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