Transport Area Working Group
Internet-draft
Expires: May 2002

S. Bailey (Sandburst)
J. Chase (Duke)
J. Pinkerton (Microsoft)
A. Romanow (Cisco)
C. Sapuntzakis (Cisco)
J. Wendt (HP)
J. Williams (Emulex)

TCP ULP Framing Protocol (TUF)

draft-ietf-tsvwg-tcp-ulp-frame-01

Status of this Memo

Copyright Notice

Abstract

The TCP ULP Framing (TUF) protocol defines a shim layer protocol between an Upper Layer Protocol (ULP) and TCP. TUF also depends on a specified TCP segmentation convention between TUF endpoints. Together, the shim and segmentation conventions enable a TUF/TCP receiver to recognize ULP data units within a TCP segment independently of other TCP segments. This capability simplifies

the design of enhanced network interfaces implementing direct data placement for ULPs using TCP. Direct data placement is a key step to making IP networking competitive with high-end interconnect solutions in data centers and other high-performance application domains.

Table Of Contents

1. Definitions

The following terms and abbreviations are used in this document.

data delivery - the delivery of received ULP payloads to the ULP application, i.e, notifying the application of data arrival by completing a receive operation or generating an event.

data placement - the storage of received ULP payloads to host memory, pending delivery to the ULP application.

direct data placement - the storage of received ULP payloads directly to application-specified buffers without intermediate buffering or copying.

EMSS - the effective maximum segment size. EMSS is the TCP maximum segment size (MSS) defined in RFC 793 [TCP] and exchanged during TCP connection establishment, adjusted by the current path maximum transfer unit (MTU) [PathMTU].

FPDU - framing protocol data unit. The protocol data unit defined by TUF.

MULPDU - maximum upper layer protocol data unit size. The size of the largest ULPDU that fits in an EMSS-sized FPDU.

NIC - network interface controller. The device that provides a host's access to a physical network link.

PDU - protocol data unit. A self-contained block of control and data defined by a particular protocol.

RDMA - Remote Direct Memory Access protocol. A data transfer protocol which uses memory access-style transfer mode(s) to provide generic direct data placement capabilities for arbitrary ULPs.

TUF - TCP ULP Framing protocol. The protocol defined in this document.

ULP - upper layer protocol. The client protocol using the services of the transport layer, or TUF.

ULPDU - upper layer protocol data unit.

ULPDU containment property - the property that a TCP segment contains exactly an integral number of ULPDUs.

2. Overview

This section summarizes the motivation for the TCP ULP Framing (TUF) protocol and explains its operation in brief. Section 3 (`Rational for TUF') develops the rationale for TUF in detail. Section 4 (`The Protocol') defines the protocol itself. Section 5 (`Protocol Characteristics') examines various properties of the protocol's operation. Implementors may wish to refer directly to sections 4 and 5.

2.1. Motivation

The IP protocols are not usually used for high-performance high speed data transfers due to overhead in TCP processing. Instead, a number of special purpose protocols have been used. The domain of application for such high speed buffer transfer includes storage, video delivery and processing, and various applications of cluster computing, such as scalable database or application service. For reasons discussed below, today, there is great industry interest in developing an IP standard for low overhead high bandwidth data transfer, which would decrease the costs of high speed interconnects and supplant special purpose protocols.

The approach typically used for low overhead transfers is called direct data placement, in which the network interface places data directly in application buffers, avoiding the latency and memory bandwidth costs associated with copying. Direct data placement can in principal be done with either of IP's reliable transports--SCTP or TCP. This document considers what is needed to do direct data placement with TCP.

In order to place data directly in application buffers, the network interface needs to use information in the Upper Layer Protocol Data Units (ULPDUs) contained in the TCP stream. This can be accomplished routinely except when TCP segments arrive out of order. If TCP segments arrive out of order, the location of the ULPDUs in the TCP segment cannot be found. The TUF protocol addresses this problem of finding ULPDU headers in the TCP stream, even when TCP segments arrive out of order.

2.2. Approach

TUF is implemented as a shim layer between an ULP and TCP. The end-to-end data flow is:

0. Use of TUF is negotiated end-to-end by the ULP.

1. The ULP delivers a data stream with ULPDUs delimited to TUF.

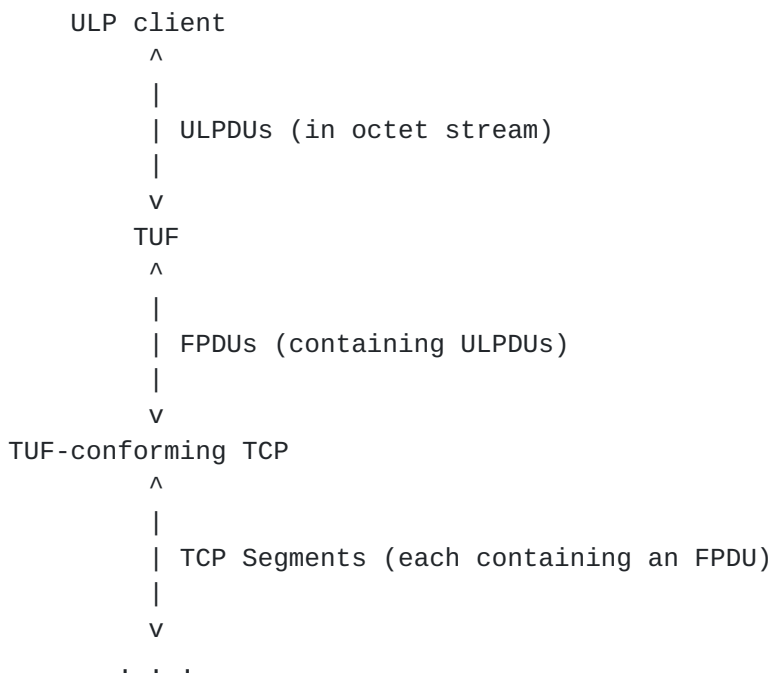
2. TUF inserts a header and delivers the shimmed ULPDUs to TCP.

3. The TUF-aware TCP sender preserves boundaries of shimmed ULPDUs (TUF FPDUs) as much as possible when delivering segments to the IP layer.

4. The receiving TCP delivers shimmed ULPDUs to the receiving TUF layer.

5. TUF removes the shim and delivers the ULPDUs to the ULP.

In other words, the layering of TUF is:



Note that while the semantics of this protocol layering must be maintained, the receiving network interface may use the information in the framed ULPDUs to place the data in memory on the host. Whatever the case, the data is only delivered to the ULP when all preceding TCP data has arrived.

3. Rational For TUF

This document defines the TUF protocol as a shim layer between an Upper Layer Protocol (ULP) and TCP. TUF also depends on a TCP segmentation convention between TUF/TCP endpoints specified in this document. Taken together they provide the capability for a TUF/TCP receiver to recognize ULPDUs by processing each TCP segment independently, without requiring state from previous segments.

The purpose of TUF is to enable practical designs for enhanced network interfaces (NICs) implementing direct data placement for TCP-based ULPs. The purpose of direct data placement is to eliminate the need for a host to copy received data after it arrives in host memory. This copying incurs CPU, memory and bus costs that are substantial and are not masked by advancing hardware technology.

A general and practical solution to the receive copy problem has eluded the IP networking community for almost two decades. There is a long history of research and experimental schemes to reduce or eliminate receiver copying overhead for IP networking in general, and for TCP/IP communication in particular. While these systems have convincingly demonstrated the potential performance benefits of reducing copy costs, all such schemes suffer from one or more of the following limitations: they require a significant restructuring of operating system buffering and/or APIs; they are limited to specific modes of communication (e.g., bulk data transfer) or specific application ULPs; they do not scale on multiprocessor hosts; their benefits depend on specific properties of the network (e.g., large MTUs) or host buffer size and alignment. Moreover, all such schemes require some degree of support from NICs to separate payloads from headers and/or ensure that their placement in host memory meets specific requirements (e.g., for page placement and alignment).

Inherent copying costs for IP communication are one motivation to use alternative non-IP technologies for high-speed networking. A number of specialized technologies have been developed for high speed data transfers in which network interfaces transfer data from application buffer to application buffer without software touching the data. Some examples include the VAXCluster Interconnect in 1983, Fibre Channel (FC) in 1994, and today InfiniBand (IB) and Virtual Interface Architecture (VIA). These alternatives have eroded the popularity of IP technologies in application domains including network storage, video processing and delivery, and cluster computing for scientific applications and scalable database-related services.

Until recently, several factors have limited interest in promoting IP networking as a solution in these application domains. First, the competing network technologies offered significantly higher link speeds than the network hardware available for use with IP. Second, these application domains were a relatively small segment of the network market. Recently, however, Ethernet networks have closed the bandwidth gap and even exceeded the bandwidth of alternatives such as FibreChannel, at much lower cost. At the same time, an increasing number of applications are server-hosted in data centers to enable sharing and access from a growing number of IP-connected client devices and locations. With the growth in importance and number of data centers, high-speed interconnection within the data center is now central to the everyday operation of Internet services.

Thus, technology changes have created an opportunity and demand to extend the benefits of IP technologies to high-performance application domains, while simultaneously increasing the importance of those domains. The ubiquity of IP offers economies of scale heavily favoring IP in these domains. For example, reliance on specialized non-IP technologies for high-performance domains creates a need to support multiple protocols and redundant network infrastructure in data centers, and it compromises portability and interoperability of data center solutions. Moreover, comprehensive support for network management and security is developing rapidly in the IP space. Use of IP technologies would allow data centers to benefit from these enhancements.

3.1. Direct Data Placement

Direct data placement is a key step toward making IP networking competitive in data centers and other high-performance domains. Direct data placement refers to the ability of a NIC to place data directly from the network into designated application buffers, without intermediate copying. Direct data placement is attractive relative to other solutions to the receive copy problem. It is the only solution that can be implemented in a way that is compatible with existing operating systems, since the receiving NIC takes over most of the responsibility to avoid receive copying. Also, direct data placement generalizes easily to a range of ULPs. In particular, the establishment of an IETF standard for an IP transport-based direct data placement protocol, which would allow NICs to directly place data independent of the application ULP using it.

The TUF protocol is necessary to permit easily deployable enhanced NICs supporting direct data placement. Such NICs already exist and their usage is growing rapidly, but their development is impeded by

the lack of standards. Direct data placement is unnecessarily difficult and expensive to design and implement for existing TCP-based ULPs; the key objective of TUF is to define transport conventions to simplify the design of these NICs. A related impediment is that in the absence of a general direct data placement protocol these products are limited to specific ULPs such as iSCSI. TUF, and possibly additional, higher layer protocol definitions outside the scope of this document, would encourage the market by ensuring interoperability of product offerings from different vendors.

This document defines a framing protocol (TUF) and TCP segmentation conventions that enable simple support of direct data placement for a class of TCP-based ULPs. It does not propose a generic direct placement ULP, such as an RDMA protocol, or any facility for direct data placement, but only the foundations for building such a facility on TCP. A key objective of TUF is to do this in a way that is compatible with existing standards and with the spirit of TCP's stream communication model. TUF can simplify support for direct data placement for ULPs such as iSCSI, and it can serve as a basis for a future RDMA proposal.

The key limitation of TUF as a solution to the receive copy problem is that it works only if the ULP standard and the sending and receiving implementations all support it. Impact on the sender and ULPs is minimal, but ULPs must be adapted to allow use of TUF at the ULP/transport boundary. The necessary modifications may be quite small. Use of TUF is a negotiated option between the sender and receiver for each ULP session, preserving interoperability among senders and receivers that do not support TUF.

3.2. Direct Data Placement with TCP

Direct data placement is widely used to accomplish high-performance data transfer in non-IP technologies such as block storage channels (SCSI, Fibre Channel, etc.), and other specialized high performance networks like InfiniBand. This section considers how direct placement can be done with TCP.

The Internet Protocol suite provides two transports that are prime candidates for use with direct data placement -- SCTP and TCP. The framing features of the SCTP Stream Control Transmission Protocol [SCTP] make it more directly adaptable for direct data placement for future ULPs using SCTP. However, the maturity and ubiquity of TCP make it desirable to define a flexible method for direct data placement for TCP-based ULPs as well.

There has been a great deal of `moral confusion' concerning the

interaction of direct data placement with TCP's ordering guarantees. These ordering guarantees do not prohibit direct data placement, even if data is placed as it arrives out of order.

TCP guarantees data delivery to the application ULP as an ordered, sequential stream [RFC793]. Data is delivered only when TCP has notified the application of its arrival and transferred ownership of the receive data buffer. TCP does not specify how received data is stored prior to its delivery, and it does not preclude placement of data in application buffers out of order, as long as no data is delivered until all preceding data has also been delivered. Out-of-order placement greatly simplifies direct data placement NICs because it streamlines data paths and eliminates the need for a TCP reassembly buffer on the NIC.

An implementation performing direct data placement must still respect all TCP delivery semantics. For example, if a checksum integrity check fails, the data must not be placed in ULP-supplied buffers, because, for example, the TCP ports and the TCP sequence number are not trustworthy.

3.2.1. The Simple Case: ULP-unaware Placement

Direct data placement into a ULP client-supplied buffer designated to hold the next data delivered to the ULP, regardless of the contents of the received data, is one of the simplest possible forms of direct data placement. This form of direct data placement is already fully supported by existing TCP mechanisms. New NIC products currently, or soon to be available, which claim to offer `full zero copy operation' typical provide only this ULP-unaware form of direct data placement.

While ULP-unaware direct data placement works well for ULPs like FTP where the entire contents of a TCP connection are known to be nothing but a single stream of bulk client data, most widely used ULPs, e.g. HTTP [HTTP], BEEP [BEEP] and storage protocols, multiplex control and data, and possibly even interleave data from different requests on the same TCP connection. The simple ULP-unaware direct data placement is inadequate to avoid data copies for these ULPs.

3.2.2. The Complex Case: ULP-aware Placement

An explicit goal of this proposal is to support out-of-order direct data placement for ULPs that provide additional transport-like features such as control and data multiplexing, layered above TCP (e.g., iSCSI or a generic direct data placement protocol such as RDMA). In many ULPs, such as storage protocols, control

information contained in the ULP uniquely identifies the destination application buffer of each particular piece of data.

For example, suppose a client requests a read operation using a network storage ULP, specifying the destination buffer for the requested data. The requesting ULP includes control information in the request (e.g., in the ULPDU header) uniquely identifying that buffer, and the responder includes that information in the read response. For some protocols, the identifier is a unique request ID, allowing the client ULP to identify the buffer indirectly through a table of pending requests. If the storage protocol uses RDMA, the response may specify the buffer directly by means of a region identifier.

A network interface that understands the relevant ULP control information can use it to place the incoming data (e.g., read response payload) directly in the correct buffer. In this case, data placement is guided by ULPDU headers embedded in the TCP data stream. The NIC accesses these headers as hints for placement of the ULP payloads--a form of integrated layer processing for each TCP segment as it arrives. This is compatible with TCP's ordering properties if completion of ULP header processing and delivery of the payload data to the application are strictly in order.

3.2.3. The Problem of ULP-aware Placement with TCP

The problem with performing direct data placement as a function of ULP control information in TCP is that it may be difficult to locate the ULP control information (ULPDU headers) within a TCP segment.

If all TCP segments are received in sequence order, ULP control information can be unambiguously located by the rules that permit any ULP implementation to do so. For example, each ULPDU may contain a length field that implicitly specifies the location of the beginning of the subsequent ULPDU.

If TCP segments are not received in sequence order, without taking additional measures, it may not be possible to unambiguously locate ULP control information needed for direct data placement. For example, if ULPDU length information is in a TCP segment that is delayed or lost in transmission, assuming the ULPDU length is the only means of locating the beginning of the subsequent ULPDU, it is impossible to locate ULP control information for ULPDUs in subsequent TCP segments until the lost or delayed TCP segment is received. ULP control information, and the data whose placement depends on it may even be in different TCP segments. If the ULP control information is in a TCP segment that is delayed or lost, it

is impossible to directly place the data until the ULP control information is received.

3.2.4. Finding ULPDUs In Out-of-order Segments

Early attempts at ULP-aware direct data placement in TCP took the approach of only directly placing data for TCP segments received in-order. Otherwise, data was copied through a reassembly buffer as in a traditional implementation. Unfortunately packet loss, and attendant out-of-order reception is a frequent, continuous characteristic of both wide-area, and switched local area networks of almost any size, as TCP adjusts to varying congestion conditions. Under these conditions, a large portion of the data transferred ends up being copied, rather than being directly placed.

Another solution to this problem is to build a reassembly buffer into the network interface. Data received out-of-order can be held in the network interface reassembly buffer until all preceding data is received, and then direct placement can be performed on the reassembled data. Within certain implementation assumptions, this is reasonable approach, but, unfortunately there are a number of issues including very large memory requirements, limited scalability, and increased latency, that make the reassembly approach undesirable.

The size of reassembly buffer needed in the network interface is a direct function of the bandwidth * delay product of all active TCP connections. Reasonable assumptions on the active bandwidth * delay product can imply a large amount of reassembly memory. Furthermore, this large reassembly memory must run at high speed---more than two times the link speed, to maintain full link bandwidth.

Finally, performing reassembly in the network interface requires that the bandwidth from the network interface to host memory be not just equal, but substantially greater than the maximum bandwidth of the network link, to ensure that the reassembly buffer is drained when reassembly is complete. System bus and interconnect bandwidth are particularly scarce and expensive resources in most systems.

What is needed to permit ULP-aware direct data placement without reassembly buffering is a way to ensure that the ULP control information and the data associated with it is highly likely to be contained completely within a single TCP segment, and a way for a receiver to validate this containment property on TCP segments it receives. If the receiver can determine that a ULPDU starts at the beginning of a TCP segment, the receiver can perform ULP-aware

direct placement for that ULPDU, and subsequent ULPDUs contained in that TCP segment. The property that a ULPDU is completely contained within a TCP segment is called the `ULPDU containment property'.

3.2.5. The TUF Solution

The TUF protocol defines a shim layer above TCP and below the ULP that allows the receiver to validate the ULPDU containment property for each TCP segment received, independently of any other TCP segment. The TUF protocol also defines a segmentation behavior for the TCP sender that ensures the ULPDU containment property holds as often as possible while still respecting the protocol requirements for TCP senders.

The TUF-specified TCP segmentation behavior ensures that the ULPDU containment property is maintained as long as the receiver window size is at least equal to the effective MSS (EMSS), the path MTU (PMTU) does not change, and the TCP stream is not resegmented by an intermediary. In conditions where the TCP receiver window size is smaller than EMSS, or the PMTU changes, the segmentation behavior further ensures that once the relevant condition is restored, the ULPDU containment property will be satisfied again.

For the high-performance applications that this protocol targets, small receiver window sizes, and PMTU changes are rare transients. Thus, the specified protocol ensures that ULP control information and its associated data are virtually always together in a single TCP segment.

3.2.6. TUF's ULP Assumptions

A key assumption of TUF is that ULPs running on TUF can adjust ULPDU sizes to fit completely within an EMSS-sized TCP segment. Clearly, if a ULPDU does not fit within an EMSS-sized TCP segment, the ULPDU containment property can not be satisfied. Most storage protocols (e.g. iSCSI), and other performance-targeted protocols (e.g. RDMA protocols) support this capability. ULPs that can not adjust ULPDU sizes to fit within an EMSS-sized TCP segment, but still want the performance advantages of direct data placement, can be mapped on top of an intermediate protocol (e.g. an RDMA protocol) that does support this data `chunking'.

TUF does not change the stream delivery semantics of TCP to the ULP, through the TUF implementation. It merely inserts a shim header that can be used by direct placement network interfaces to verify the ULPDU containment property. The shim header is inserted by the sending TUF implementation and removed by the receiving TUF

implementation, leaving a stream to be delivered to the ULP.

4. The Protocol

This section defines the TUF protocol itself. The first two sections are the core of the protocol defining:

- the shim layer PDUs, called FPDUs,
- a TCP-conforming segmentation behavior which ensures the ULPDU containment property holds under most conditions.

The remaining sections cover other aspects of the protocol which are primarily implications of the core protocol:

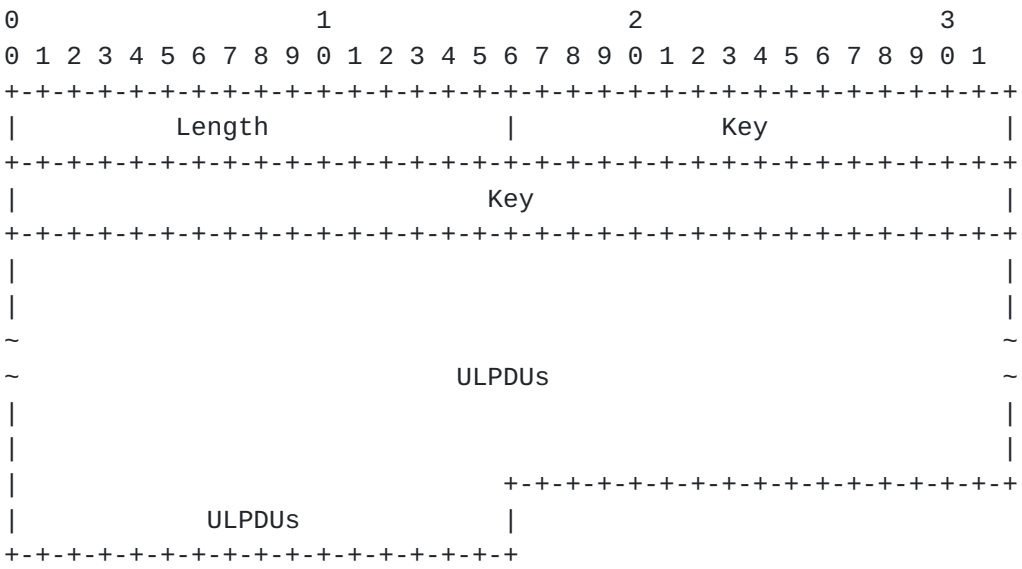
- what ULP-specified negotiations to enable TUF must accomplish,
- how receivers can process received TCP segments to establish whether the ULPDU containment property holds.

4.1. The Framing Protocol Data Unit (FPDU)

TUF sends groups of one or more complete ULPDUs in a framing protocol data unit (FPDU).

4.1.1. FPDU Format

The format of an FPDU is:



Length: 16 bits (unsigned integer)

This is the length in octets of the set of framed ULPDUs. It does not include the length of the FPDU header itself.

Key: 48 bits (unsigned integer)

This is used by the receiver to validate the ULPDU containment property. It is selected at random by the sender, and initially signaled to the receiver in a ULP-specified way, before the receiver attempts to test the ULPDU containment property. All FPDUs sent on the same connection in the same direction must use the same key value. A good quality random number generator MUST be used to generate the initial key. RFC 1750 discusses relevant characteristics and provides references for good quality random number generation [RFC1750].

The length of an FPDU is 8 + L octets, where L is the length of the set of framed ULPDUs. The 16-bit length field is sufficient to permit a TCP segment with an FPDU to completely fill a maximum-size IPv4 or IPv6 datagram.

4.1.2. FPDU Size Selection

Each FPDU SHOULD contain as many contiguous, complete ULPDUs as will fit within the current EMSS, unless ULPDU packing is disabled. If ULPDU packing is disabled each FPDU SHALL contain a single ULPDU. ULPDU packing mode may be negotiated, or specified a priori by a ULP. Disabling ULPDU packing is analogous to disabling the Nagle algorithm in TCP.

TUF SHALL present the size of the largest ULPDU size fitting in an EMSS-sized FPDU (MULPDU) to the ULP. MULPDU is EMSS - the FPDU header size (8 octets). ULPs SHOULD submit as large ULPDUs as possible to TUF, up to MULPDU, subject to limits imposed by specific ULP properties. The ULP MAY also chose to pack several ULPDUs into an EMSS-sized unit before submitting them as one ULPDU to TUF. Depending upon the ULP, ULP packing may improve data transfer efficiency, and is unlikely to have any detrimental effect.

A TUF implementation probing for PMTU increase SHOULD present an increased MULPDU value to the ULP until a large enough FPDU to perform the probe results.

Under exceptional circumstances, the EMSS can become too small to accommodate even a single ULPDU. For example, a ULP may define fixed-sized PDUs that are incompressible, or variable size PDUs with some absolute minimum size, such as the size of a data PDU

containing a minimum amount of data. It is possible for the EMSS to shrink to as small as 8 octets [PathMTU]. If the EMSS is too small to accommodate an incompressible ULPDU, the FPDU MUST contain only that ULPDU. ULPs using TUF SHOULD NOT define ULPDUs with a minimum size greater than 128 octets.

4.2. TUF-conforming TCP Sender Segmentation

TCP senders are allowed substantial freedom in the choice of how to segment an outgoing TCP stream. Within the confines of the receiver-advertised receive window, and the sender computed congestion window, any segmentation is permitted. Virtually all TCP implementations do attempt to segment outgoing TCP streams into EMSS-sized segments where possible because it improves performance.

TUF-conforming TCP sender behavior ensures that the ULPDU containment property holds most of the time. To do this, a TUF-conforming TCP sender MUST respect a single additional rule in performing segmentation:

> A TUF-conforming TCP sender MUST segment the outgoing TCP stream such that the first octet of every FPDU is sent at the beginning of a TCP segment

4.3. Negotiating TUF

Negotiating the use of TUF is the responsibility of the ULP. The use of TUF MAY be negotiated separately for each direction on a connection. The negotiation procedure MUST ensure that when TUF is enabled or disabled, the remote peer will not transmit its first TCP segment in the new mode until it is certain that the local peer has actually enabled or disabled TUF.

TUF operation is characteristically requested by the receiver and offered by the sender. Before enabling TUF, the relevant parameters:

1. the sender's 48-bit key

2. ULPDU packing mode

MUST be established at each peer.

A natural way to enable the use of TUF is a ULP-defined negotiation exchange of the TUF parameters culminating in enabling TUF, if requested, for each transfer direction. A three-way handshake protocol can be used to ensure that the point at which TUF is enabled is unambiguous and each end has time to perform local state

changes. A connection on which TUF is enabled is likely to be the same connection on which the negotiation occurs, but this is not required. A new connection could also use TUF from its initial establishment, if the TUF parameters and modes are known through some out-of-band mechanism.

Use of TUF could be disabled during a connection using a similar ULP-defined three-way handshake.

Other alternatives to parameter exchange include stipulating some parameters a priori. For example, a ULP could specify that TUF with ULPDU packing enabled is always used in both directions. In this case, only the 48-bit keys need to be exchanged before TUF is enabled. Or, a ULP could determine TUF characteristics on the basis of the TCP port number.

4.4. TUF Receiver ULPDU Containment Property Testing

A TUF receiver that wishes to use ULP control information to perform direct data placement must first verify the ULPDU containment property. To do this, the receiver MUST establish that the TCP segment contains exactly one FPDU. Abstractly, this can be done by assuming the TCP segment payload begins with an FPDU, and verifying the following properties of that putative FPDU:

- The received TCP segment payload length equals the FPDU length plus the length of the FPDU header (8 octets).

- The 48-bit key equals the value signaled to the receiver when TUF was enabled for the connection.

If these conditions are true, the TUF receiver MAY assume that the ULPDU containment property holds, and use ULP control information to directly place data in the contained ULPDUs.

TUF DOES NOT provide any information that a TUF receiver can use to locate ULP control information beyond the ULPDU containment property. In particular, a TUF receiver MUST NOT scan TCP segments in an attempt to locate FPDUs that do not begin at the beginning of a TCP segment. However, even if the ULPDU containment property does not hold, a TUF receiver may still be able to reliably locate and use ULP control information. For example, if a received TCP segment contains the next unreceived data in the TCP stream, the location of ULPDUs in that segment are unambiguous. The behavior of a TUF receiver acting on ULP control information located with properties other than the ULPDU containment property is not specified here.

5. Protocol Characteristics

This section discusses some characteristics and behavior which are implications of the TUF protocol.

5.1. Properties Of TUF-conforming TCP Senders

The general practice of TCP senders to send as much data as possible within a TCP segment (up to EMSS) implies that an FPDU whose size is less than or equal to EMSS, and whose first octet begins a TCP segment will be sent entirely within a single TCP segment. This ensures the ULPDU containment property for that TCP segment.

A TUF-conforming TCP sender still obeys all requirements of TCP. While the segmentation of a TUF-conforming TCP sender will have distinctive characteristics when viewed from the network wire, the same segmentation behavior could also result from a stock TCP sender.

The one property of a TUF-conforming TCP sender which arguably departs from traditional expectations is that a TUF-conforming TCP sender may not produce TCP segments which are as close in size to EMSS as a stock TCP sender. The need to ensure the ULPDU containment property may result in TCP segments which are not as full as if the property did not need to hold. While this is abstractly true, in practice, several characteristics combine to minimize this effect. Specifically:

- Packing ULPDUs into FPDUs gives behavior similar to that of stock TCP segmentation, albeit with coarser granularity.

- ULPs which benefit from data-dependent direct data placement (candidates for TUF) usually transfer large amounts of data in bulk. This means that most ULPDUs are data-carrying, and will be EMSS-sized. Even when control is interleaved with data, the combination of a small number of control ULPDUs with a data ULPDU can be packed to fill an EMSS-sized segment.

Therefore, a TUF-conforming TCP sender seems likely to behave similarly to a stock TCP sender under most circumstances. However, applications that both send and receive data over the same TCP connection, where there might be dependencies between incoming and outgoing data, are often subject to excessive delays attributable to TCP's Nagle algorithm and/or delayed-ACK algorithm [NagleDAck]. These algorithms generally perform best when TCP always sends full-EMSS segments. Because TUF can generate sub-EMSS segments as a by-product of aligning FPDU boundaries with TCP segment boundaries,

TUF might be especially vulnerable to the known problems with the Nagle and/or delayed-ACK algorithms.

Further work, including implementation experience with TUF, as well as existing and future proposals for improvements to the Nagle and/or delayed-ACK algorithms, might be necessary to optimize TUF performance while fully preserving the congestion-avoidance features of TCP. This work is currently outside the scope of this document.

5.2. Exception Cases

The complete operational specification of TUF is contained in the rules for forming FPDUs, and sending those FPDUs in TCP segments. However, the operation of TUF will be subject to a variety of transient or exceptional conditions. The behavior of TUF under those conditions is discussed below to illustrate specifically how TUF addresses them.

5.2.1. Resegmenting Intermediaries

Resegmenting TCP-layer intermediaries (middleboxes) are one of the most formidable obstacles to maintaining the ULPDU containment property. In the presence of such an intermediary, the segmentation chosen by the sender may not be the segmentation at the receiver. While such intermediaries may or may not be common in particular networks, in many cases the presence or absence of such resegmenting behavior is beyond the control or even knowledge of the end points using TUF. Therefore, TUF must detect such resegmentation by design.

A primary reason for the presence of a random key in the FPDU header is to detect such resegmentation. An alternative to the random key which has been proposed, is to use ULP-specific validation criteria to determine the ULPDU containment property. For example, some ULP PDUs include relatively strong data integrity checks such as CRCs, and other ULP control information can often be validated against various ULP-specific criteria.

While such ULP-specific validation criteria may involve checking many more bits than the combination of the FPDU's 16-bit length and 48-bit key, ULP-specific validation criteria may not actually offer a strong guarantee of the ULPDU containment property. For certain data streams, the probability of a false-positive indication of the ULPDU containment property can be extremely high.

Assume that the intermediary resegments to a granularity of no finer than G octets (e.g. 4). Also assume that the TCP data stream

contains predominantly application data. If the ULP is a storage protocol, simply transferring a file containing a continuous, repeated stream of well-formed ULPDUs which are some multiple of G in size increases the probability of a false-positive indication of the ULPDU containment property to approximately:

1 / (sizeof(repeated ULPDU)/G)

If the well-formed ULPDUs are relatively small (e.g. 32 octets where G=4 octets), the probability of a false-positive indication of the ULPDU containment property is approximately 1/8, for EACH TCP segment which does not actually begin with a ULPDU. Clearly, in this case, it would take only a very small number of TCP segments which do not begin with an actual ULPDU before the `fake' ULPDU in the application data is interpreted as an actual ULPDU. The consequences of such a false-positive interpretation could be dire, for example executing a destructive operation request.

The 48-bit random key in the FPDU results in a low probability of a false-positive indication of the ULPDU containment property because it is effectively secret with respect to the application data stream.

Note that although this analysis may appear to be security-minded, prompting the image of a sighted third-party adversary that can `sniff' the 48-bit key, it is actually considering a safety, rather than a security property. The security properties of TUF are discussed in Section 6 (`Security Considerations') below.

Even though TUF can detect the presence of a resegmenting intermediary, such an intermediary will almost certainly substantially reduce the chance of the ULPDU containment property being satisfied. A TUF implementation which detects a very low incidence of the ULPDU containment property for a sustained interval (>> RTT) may assume that a resegmenting intermediary is in operation and SHOULD discontinue the use of ULP control information found using the ULPDU containment property. In such cases, the ULP MAY elect to disable the use of TUF altogether, or simply just stop exploiting the ULPDU containment property.

5.2.2. PMTU Reduction

When a PMTU reduction is detected by a TUF-compliant TCP, the TUF-compliant TCP sender may send FPDUs already committed to the TCP layer in one of two ways:

- send unsegmented FPDUs in TCP segments of the old EMSS size, and rely on IP fragmentation to deliver the segments,

- segment FPDUs to fit in TCP segments which respect the new EMSS size.

Stock TCPs face a similar choice on PMTU change, and both alternatives are used in practice.

In the case that a TUF-compliant TCP chooses to segment FPDUs, it SHOULD segment them in such a way that, in the absence of resegmentation by an intermediary, the segments are guaranteed not to give a false-positive indication of the ULPDU containment property. There are various ways to ensure this. For example, no matter how the FPDU is segmented, the first segment is guaranteed not to give a false-positive indication of the ULPDU containment property---the 48-bit key will match, but the length will not. In the worst possible case, each subsequent TCP segment could be sent with fewer than 8 octets of data, also guaranteed not to give a false-positive indication of the ULPDU containment property. More efficient approaches are possible, but PMTU reduction is a rare event, and reacting to it is only a transient condition. Eventually a new MULPDU will be presented to the ULP, and FPDUs that fit in the new EMSS will result. During the transient condition, performance will suffer temporarily no matter how FPDUs are segmented.

No matter what segmentation is chosen by a TUF-compliant TCP sender when segmenting an FPDU, if the segments pass through a resegmenting intermediary, the correctness of the ULPDU containment property remains strictly a matter of probability.

5.2.3. PMTU Increase

As described in `FPDU Size Selection' above, a TUF-compliant TCP probing for PMTU increase will present an increased MULPDU value to the ULP. This should eventually lead to an FPDU large enough to actually perform the PMTU increase probe. The MULPDU value should not be further adjusted until the probe is actually performed. This behavior is similar to when a stock TCP would like to perform a PMTU increase, but less data is available than would fill the desired segment.

Also, note that depending on the ULP, the actual distribution of FPDU sizes may have a granularity coarser than a single octet. An FPDU with an particular, desired TCP segment size may never be generated. Therefore when probing for PMTU increase, a TUF-compliant TCP must be satisfied with an FPDU that produces a TCP segment size that is `close' to the desired size.

Finally, note that in cases where PMTU grows and shrinks relatively

frequently, better performance may result from not probing for PMTU increase at all, or probing very rarely. This is because the performance disruption resulting from PMTU decrease can be substantial, and in many cases, implementations of TUF will be in hardware, so performance may less sensitive to differences in PMTU.

5.2.4. Receive Window < EMSS

A TUF-compliant TCP sender that is presented with a receive window smaller than EMSS may be required to segment FPDUs. The TCP window probe is a limiting case of this condition where the advertised receive window is 0, and the amount of data typically sent in response is a single octet.

In this case, a TUF-compliant TCP sender will segment in accordance to the requirements of TCP, and the rule defined in `TUF-conforming TCP Sender Segmentation' above. In addition, as when resegmenting in response to PMTU decrease, a TUF-compliant TCP sender SHOULD segment in such a way that, in the absence of a resegmenting intermediary, segments are guaranteed not to give a false-positive indication of the ULPDU containment property. In situations where the receive window is smaller than EMSS, data transfer performance is likely to be limited independently of any segmentation behavior by the TCP sender. Furthermore, ULP implementations that choose to use TUF will almost certainly be designed to maintain a receiver window larger than EMSS, so a small receiver window should occur extremely infrequently.

5.2.5. Size of ULPDU + 8 > EMSS

In cases where EMSS shrinks below the minimum size of a ULPDU that a ULP wants to send, TUF will create FPDUs that are larger than EMSS, and a TUF-compliant TCP sender will face the same alternatives as during PMTU reduction:

- send unsegmented FPDUs and rely on IP fragmentation to deliver the segments

- segment FPDUs to fit in TCP segments which respect the EMSS size

A ULP which is presented with an MULPDU value that is too small to accommodate PDUs necessary operation SHOULD simply attempt to use ULPDUs which are as small as possible

If the EMSS shrinks to a pathologically small size, then a TUF implementation SHOULD discontinue the use of ULP control information found using the ULPDU containment property. In such

cases, the ULP MAY elect to disable the use of TUF altogether, or simply just stop exploiting the ULPDU containment property.

A path MTU which results in an EMSS < 128 + 8 octets is an extremely unlikely occurrence and when it does occur, poor data transfer performance is a likely result, independent of TCP sender segmentation behavior.

6. Security Considerations

This section discusses both protocol-specific considerations and the implications of using TUF with existing security mechanisms.

6.1. Protocol-specific Security Considerations

A third-party that can inject spoofed packets into the network which can be delivered to a TUF receiver could launch a variety of attacks that exploit TUF-specific behavior. For example a blind third-party adversary could inject random packets which appear in the valid TCP window and do not begin with valid FPDU headers. A barrage of such packets might cause a TUF receiver to conclude that a resegmenting intermediary is present and disable the use of TUF and direct data placement. This would substantially degrade performance. However, it would probably also have more dire consequences than performance, such as causing the ULP to interpret the bogus data as valid. Furthermore, such a third-party could also degrade performance just as effectively in a TUF-independent way by injecting spoofed ICMP packets which result in reduction of the path MTU to an inefficiently small size.

Fundamentally, the vulnerabilities of TUF to active third-party interference are no more acute than to TCP without TUF. In both cases, a communication security mechanism such as IPSec is the only way to completely prevent such attacks.

6.2. Using IPSec With TUF

Since IPSec is designed to secure arbitrary IP packet streams, including streams where packets are lost, TUF can run cleanly on top of IPSec without any change. IPSec packets may be decrypted in the order they are received, and a TUF receiver may test and exploit the ULPDU containment property just as if the IP datagram were unsecured.

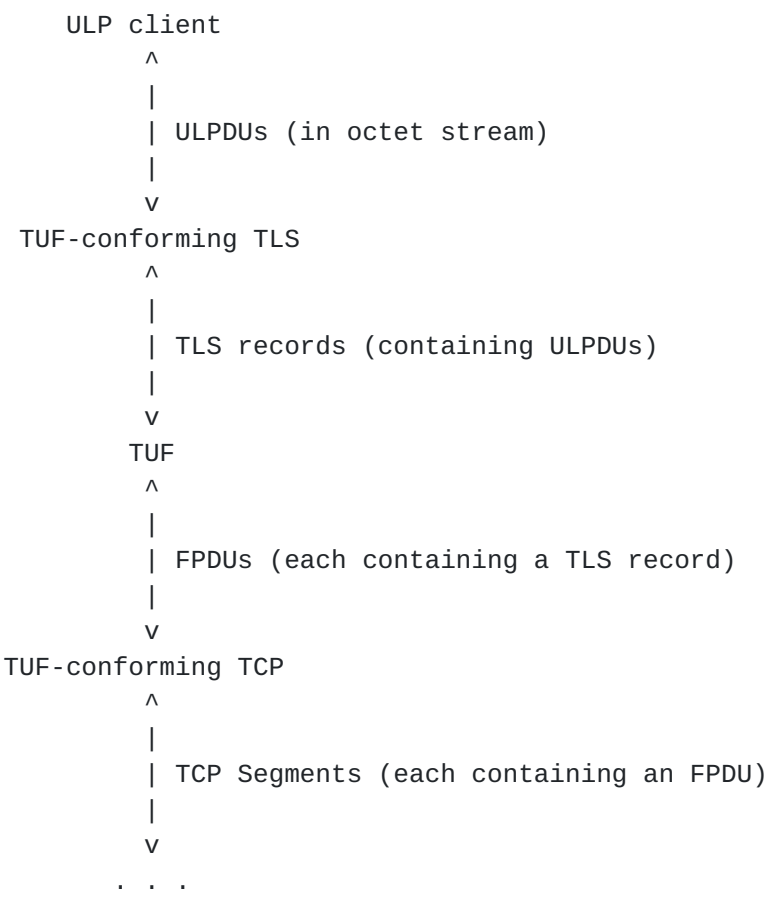
6.3. Using TLS With TUF

Using TLS [TLS] with TUF, particularly trying to exploit the ULPDU containment property to locate ULP control information, is not a

straightforward process. TUF can be directly layered on top of TLS, but many of the advantages of TUF are lost. This document does not define a way of using TLS with TUF that could offer better performance than stock reassembly buffer-based implementations. That task is left to a different document, if there is sufficient motivation to address the problems. This section does outlines some of the known complications of trying to do better than stock reassembly buffer-based implementations using TLS with TUF.

TLS is a record-oriented protocol. TLS records are PDUs with a similar structure to ULPDUs defined in application ULPs. As with other ULPs, the only way to avoid a complete reassembly buffer is to be able to find TLS PDUs in the presence of lost TCP segments. The ULPDU containment property could be used to do this, which suggests that TLS itself should be layered on top of TUF. In this case, the FPDU header will travel in the clear, but this will probably not present serious vulnerabilities other than denial of service attacks comparable to what is already possible without TUF.

Once the TLS records are located and processed it still remains to locate the ULPDUs. The simplest way to do this would be to have the TLS implementation be TUF-compliant, and ensure the ULPDU containment property within each TLS record. In this case, the protocol layering would look like:



An obvious complications of using TLS with TUF is that ciphers defined for use with TLS do not offer independence across TLS records. The most common cipher used with TLS is RC4, which is a stream cipher. Efficient decryption of an RC4 stream depends upon the entire preceding data stream. In other words, it is simply not feasible to decrypt TLS records encrypted with RC4 in any order other than the TCP stream order. This clearly defeats the purpose of TUF.

TLS is also defined to work with block ciphers such as 3DES in Cipher Block Chaining (CBC) mode. In this case, the dependency of the decryption operation on data in previous TLS records is less severe. To decrypt the current TLS record only requires ciphertext from the previous TLS record. While this does not allow complete independence of processing TLS records, a lost or delayed TCP segment containing a TLS record only prevents decrypting the immediately subsequent TLS record, not all TLS records after it.

TLS compression presents another complication to using TLS with TUF. TLS compression algorithms are allowed to increase the content length by up to 1024 octets. If the content length does

increase, the TLS record may not fit within an EMSS-sized TCP segment, even if the uncompressed ULPDU does. If the risk of exceeding an EMSS-sized TCP segment is small, it may be acceptable to occasionally send FPDUs containing TLS records that span several TCP segments, or use IP fragmentation. Some TLS compression algorithms may never increase the content length, or only increase it by some small, manageable amount.

7. IANA Considerations

If framing is enabled a priori for a ULP by connecting to a well-known port, this well-known port would be registered for the framed ULP with IANA.

8. References

[BEEP]
Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[HTTP]
Fielding, R. and others, "Hypertext Transfer Protocol -- HTTP/1.1.", RFC 2616, June 1999. http://www.ietf.org/internet-drafts/draft-ietf-tsvwg-initwin-00.txt.

[NagleDAck]
Minshall G., Mogul, J., Saito, Y., Verghese, B., "Application performance pitfalls and TCP's Nagle algorithm", Workshop on Internet Server Performance, May 1999.

[PathMTU]
Mogul, J., and Deering, S., "Path MTU Discovery", RFC 1191, November 1990.

[RFC1750]
Eastlake, D., Crocker, S., Schiller., J., "Randomness Recommendations for Security.", RFC 1750, December 1994.

[RFC2581]
Allman, M., and others, "TCP Congestion Control," RFC 2581, April 1999.

[SCTP]
Stewart, R.R. and others, "Stream Control Transmission Protocol," RFC2960, October 2000.

[Stevens]
Stevens, W. Richard, "Unix Network Programming Volume 1,"
Prentice Hall, 1998, ISBN 0-13-490012-X.

[TCP]
Postel, J., "Transmission Control Protocol - DARPA Internet
Program Protocol Specification", RFC 793, September 1981.

[TLS]
Dierks, T. and others, "The TLS Protocol, Version 1.0", RFC
2246, January 1999.

Authors' Addresses

Stephen Bailey
Sandburst Corporation
600 Federal Street
Andover, MA 01810
USA

Phone: +1 978 689 1614
Email: steph@sandburst.com

Jeff Chase
Department of Computer Science
Duke University
Durham, NC 27708-0129
USA

Phone: +1 919 660 6559
Email: chase@cs.duke.edu

Jim Pinkerton
Microsoft, Inc.
1 Microsoft Way
Redmond, WA 98052
USA

EMail: jpink@microsoft.com

Allyn Romanow
Cisco Systems
170 W Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 525 8836
Email: allyn@cisco.com

Constantine Sapuntzakis
Cisco Systems
170 W Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 525 5497
EMail: csapuntz@cisco.com

Jim Wendt
Hewlett Packard Corporation
8000 Foothills Boulevard MS 5668
Roseville, CA 95747-5668
USA

Phone: +1 916 785 5198
EMail: jim_wendt@hp.com

Jim Williams
Emulex Corporation
580 Main Street
Bolton, MA 01740
USA

Phone: +1 978 779 7224
EMail: jim.williams@emulex.com

Appendix A. Sample Sockets Support For TUF

The sockets support for TUF described below is only a sketch. It is provided as an aid to understanding TUF. Implementing this interface is not a requirement for a TUF implementation.

Other software interfaces are possible. The described interface

draws from the sockets interface for UDP. The described interface might be natural for applications already designed to support both TCP and UCP, or that do network input and output in complete PDU units. For applications that perform octet-at-a-time style input and output, an alternative interface that draws from the tradition of the TCP URG pointer interface (e.g. using a MSG_OOB flag to send()) is equally possible. An implementation may even offer several different interfaces to TUF.

That said, the sockets support sketched below might well provide the basis for a complete, standard interface to be described outside this draft.

A.1 Basic Principles

The sockets support for TUF takes the form of a set of socket options that may be set or requested to enable the appropriate behavior.

A socket may be in one of two TUF-related modes in the send direction:

1. TUF-compliant TCP sender mode. No data (FPDU headers) is added to the TCP octet stream, but each data buffer presented in a sending operation is to be sent according to the rules of TCP and TUF-compliant TCP senders. This mode provides direct access to a TUF-compliant TCP sender for purposes such as implementing TUF.

2. TUF sender mode. An FPDU header is added to data presented by an integral number of sending operations, and the FPDU is passed to a TUF-compliant TCP sender for transmission

A socket may be in one TUF-related mode in the receive direction:

1. TUF receiver mode. FPDUs are expected in each TCP segment.

If a socket receiving operation is used to retrieve received data (as opposed to the data being directly placed), FPDU headers are removed before the data is returned.

A.2 Enabling TUF

```
/* Pick a sending mode */
if (sendMode == TUF_TCP)
  mode = TUF_SEND_TCP
else
  mode = TUF_SEND;

mode |= TUF_RECEIVE;

setsockopt (s, SOL_TCP, TUF_MODE, &mode, sizeof(mode));
```

A.3 Sending Data

The standard socket sending operations, including send(), sendto(), sendmsg(), writev(), and others are used to send ULPDUs in TUF. The EMSGSIZE error should be returned if the buffer passed to the sending operation would result in an FPDU that does not fit in an EMSS-sized TCP segment, unless oversized ULPDU errors are disabled, as described below.

When the path EMSS increases, the sending operation MAY return EMSGSIZE once to inform the client of the change.

A.4 Retrieving The Current EMSS or MULPDU

```
getsockopt (s, SOL_TCP, TUF_MULPDU, &emss, sizeof(emss));
```

If the socket is in TUF_SEND_TCP mode, this call returns the TCP EMSS. If the socket is in TUF_SEND mode, the call returns the maximum ULPDU that can be submitted in a sending operation without requiring fragmentation of the associated FPDU.

The number should not count any octets that go towards TCP options.

A.5 Disabling ULPDU Packing

```
flag = 0;
setsockopt (s, SOL_TCP, TUF_PACK_PDUS, &flag, sizeof(flag));
```

This call disables TUF from packing more than one ULPDU into an FPDU. By default, ULP PDU packing is enabled.

A.6 Disabling The Report of Oversized ULPDUs

```
flag = 0;
setsockopt (s, SOL_TCP, TUF_REPORT_OVERSIZED, &flag,
            sizeof(flag));
```

This call disables sending operations from returning EMSGSIZE in response to oversized ULPDUs. It may be called at any time on a socket, whether connected or not. It is used to continue ULP operation when MULPDU is already known to be too small to permit some ULPDUs to be sent with out segmentation. Oversized ULPDU reporting can be enabled again if PMTU is discovered to have increased.