Checksum Coverage and Delivery of Errored Content

Lloyd Wood, Wesley M. Eddy, Jim McKim and Will Ivancic

***Abstract*—We examine how the overall reliability of network protocol stacks is affected by the use of error-detecting checksums and Cyclic Redundancy Checks at each protocol layer. How these checksums cover their frames and payloads can affect the reliability of the rest of the stack, particularly higher layers, and the resulting delivery of data. We then apply insights gained to a comparison of new protocols promising delivery of packets with payload errors, to determine how well those protocols achieve that goal when used in realistic stacks.**

***Index Terms*—checksum, CRC, partial checksum, partial reliability, Internet, IP, TCP, UDP, DCCP, UDP-Lite, Saratoga, Licklider, LTP, DTN, delay-tolerant networking, bundles.**

I. Introduction

Reliability in the processing and delivery of received data is fundamental to packet networking. By this, we mean being able to trust the data in the payloads of the packets that you receive, and being able to trust the structure of the very packets and frames so that you can decode them correctly and get at the payload data they hold. This is distinct from (although supplemental to and a prerequisite for) the notion of delivery reliability from acknowledgement and retransmission protocols. Even when applications can tolerate errors in the payload data they receive, delivered by the protocols discussed in section V, they must be able to rely on the structure and headers of the packets and frames delivering that content.

The well-known *end-to-end principle* [1] describes the need for checks on data delivery at the highest layer in the protocol stack, in order to be able to recognize and recover from errors introduced in transmission or in the functionality at all lower layers. We summarize the normal application of this principle to the five-layer protocol stack in Fig. 1, where checksum protection of frames and their payloads is shown for each layer. The physical layer often protects against channel-induced errors with symbol coding, but does not normally use checksums to reject errored frames. Framing bits is normally a function in higher layers.

In Fig. 1, the scenarios shown in the protocol stacks used in (a), with an end-to-end checksum at the highest layer, and (b), with checksums at all layers, are considered reliable. A ramification of the end-to-end principle is that the added lower-layer checks in (b) *may* be unnecessary, and that overall performance may be increased in individual cases by their omission, although that is not universally so.

Draft working paper, 17 July 2007.

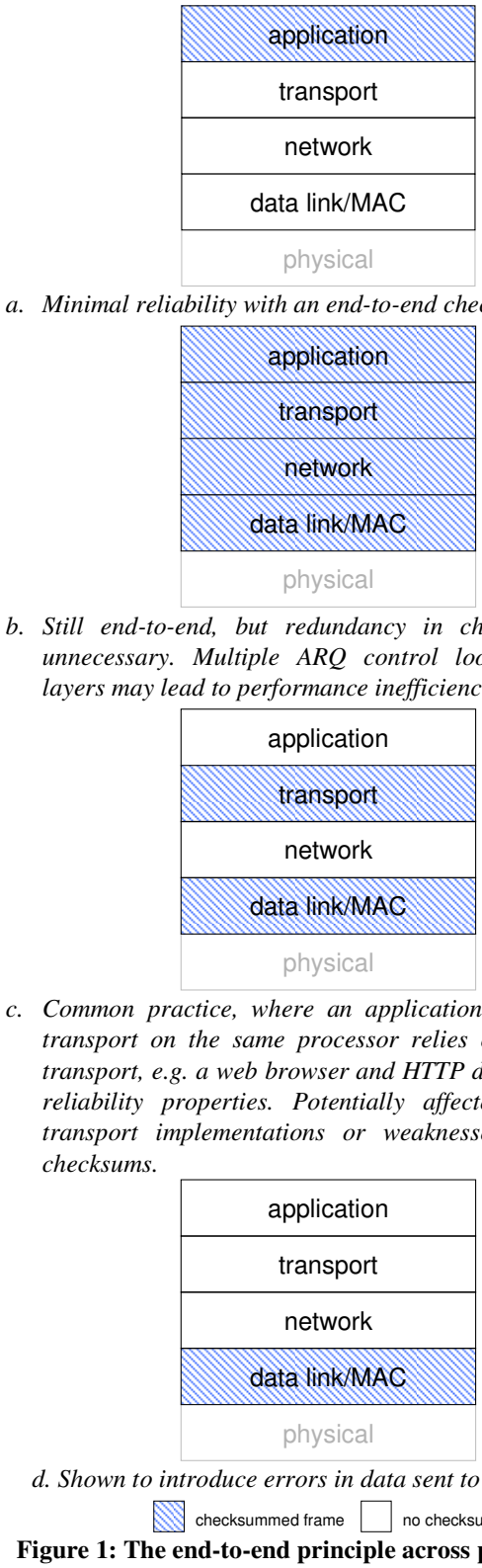
Work on this document at NASA's Glenn Research Center was funded by NASA's Earth Science Technology Office (ESTO).

L. Wood is with the Global Government Solutions Group, Cisco Systems, Bedfont Lakes, Feltham, Middlesex, England (email lwood@cisco.com).

W. M. Eddy is with Verizon Federal Network Systems, contracted to NASA Glenn Research Center (email weddy@grc.nasa.gov).

J. McKim is with RS Information Systems, contracted to NASA Glenn Research Center (email James.H.McKim@grc.nasa.gov).

W. Ivancic is with NASA Glenn Research Center, Cleveland, Ohio (email William.D.Ivancic@grc.nasa.gov).



a. Minimal reliability with an end-to-end checksum.

b. Still end-to-end, but redundancy in checking may be unnecessary. Multiple ARQ control loops at different layers may lead to performance inefficiencies.

c. Common practice, where an application colocated with transport on the same processor relies on reliability in transport, e.g. a web browser and HTTP depend on TCP's reliability properties. Potentially affected by bugs in transport implementations or weaknesses in transport checksums.

d. Shown to introduce errors in data sent to the application.

Figure 1: The end-to-end principle across protocol stacks

Omitting these lower-layer checks may be detrimental to performance, as these checks often trigger short-delay localized error recovery methods and prevent higher-latency end-to-end recovery from being necessary. An example of this is a short-delay 'last wireless hop' in a wireless hotspot being used for Internet communication.

ARQ locally across the wireless hop to repeat errored or lost frames is far faster than end-to-end recovery across the longer Internet path using a higher-layer protocol; TCP can be thought of as a much slower ARQ protocol over a longer path.

Fig. 1(c) is how the end-to-end principle is commonly applied in practice. This assumes that data is delivered between application and transport layer correctly. An overall checksum of data at the application would add protection, especially considering the relative weakness of commonly-used transport-layer checksums, and provide the ability to check reassembly of transport packets into a file. However, as with HTTP or FTP transfers, this is often not done. In practice, the link-layer CRCs are usually stronger than the transport-layer checksum, protecting against channel noise as an obvious source of errrors. Putting the strongest checksum highest in the stack would make the most sense for overall end-to-end reliability.

Fig. 1(d) is widely recognized as prone to data corruption. Examples of this configuration in practice include disabling IPv4 UDP checksums without a higher-level error check in place. This has led to subtle, hard-to-detect, corruption of files stored in network filesystems [2].

II. TYPES OF CHECKSUMS

The types and strengths of checksums vary, from simple 16-bit one's-complement checks that will not detect swapped words, through Fletcher, Adler, and other checksums, and up to strong CRCs, and even stronger cryptographic hash functions.

This paper is primarily concerned with whether a data unit's header and/or payload have any internal protection against errors via checksum coverage, rather than with the relative strengths and weaknesses of the particular checksum used.

For more detailed discussion of strengths and weaknesses of checksums and CRCs, see Stone's work, that also clearly motivates the need for end-to-end checksums that can detect errors in intermediate processing that even the use of strong link-frame checksums cannot detect [3][4].

III. TERMINOLOGY

To be able to describe the uses of checksums in protocols, we must first define some terminology for the concepts we talk about. We discuss *errors in content* (delivery of content containing payload bit errors, where permitted because the application can tolerate errors) and *reliable delivery* (delivery of all packets, with resends of lost or too-corrupted packets). Errored reliable delivery, where bit errors in the content may be tolerated as long as the correct amount of (partially incorrect) data reaches the correct endpoint, is distinct from error-free reliable delivery, and these are in turn different from unreliable delivery, where packets may be lost and not resent.

Applications that can benefit from delivery of errored content include some streaming audio and video codecs, where the user experience is degraded less by a small number of bit errors in a complete, continuous, stream than by an entirely erased frame that makes the stream discontinuous.

We are careful to distinguish between *permitting errors in content or payload data*, which, if tolerated, can allow the payload to be passed up to the next protocol layer to decode and decide whether the error can be coped with, and *errors in the frame format*, which should be detected and protected against. A protocol that can detect and protect against errors in its own format and fields (regardless of whether its payload is protected or not) is said to have a *robust format*. Being robust is desirable; more robustness of important header fields can by given by a stronger checksum. A frame that protects against errors in payload and header fields is *error-rejecting*. We talk of checksums giving *protection* of frames against errors when error rejection takes place and 'bad' errored frames are discarded.

IV. SOME EXAMPLES OF CHECKSUM COVERAGE

We examine checksum coverage and protection in common communication protocols used in the link layer and above.

A. Ethernet

Ethernet frames include a 32-bit Cyclic Redundancy Check (CRC) in the frame trailer, covering the header and payload as a contiguous whole.

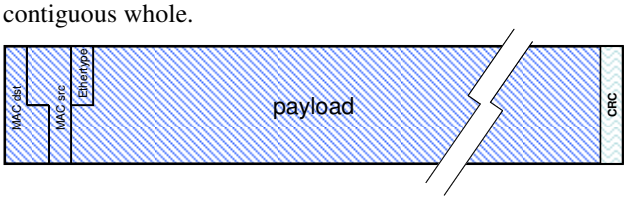


Figure 2. Ethernet Type II frame

B. HDLC, PPP and Frame Relay

High Level Data Link Control (HDLC) frames [5], which are delineated by start and end flags, include a 16- or 32-bit CRC or Frame Check Sequence (FCS) in the frame trailer, covering both the header and the payload. The Point-to-Point serial Protocol (PPP) [6][7] and Frame Relay [8] are similar overall to HDLC, with trailing checksums, although header fields differ between them.

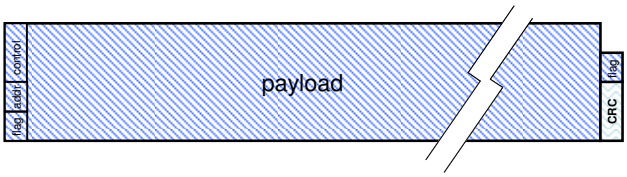


Figure 3. Typical HDLC frame layout

C. ATM and the AAL layers

The ATM Adaptation Layer 5, AAL5, the most common of the ATM adaptation layers [9] , is used to carry other protocol frames. A 32-bit CRC covers the frame and ends the trailer.

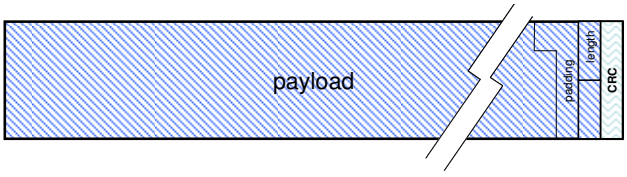


Figure 4. ATM Adaption Layer 5 (AAL5) frame

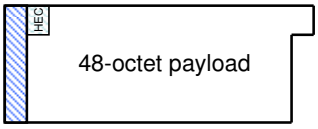


Figure 5. 53 octet ATM cell

ATM cells can be used to carry parts of ATM frames. The ATM cell has minimal header protection and robustness with the Header Error Control (HEC) octet, which is an 8-bit CRC of the previous 4 header octets.

Payload checksums are not computed across individual ATM cells, as this would be computationally and spatially prohibitive, as well as unnecessary for carrying some applications (e.g. uncompressed digitized voice telephony). Instead, checksum protection is left to the higher AAL layer.

D. Multi-Protocol Label Switching

Multi-Protocol Label Switching (MPLS) is used extensively to enable efficient forwarding, short failure recovery times, and traffic engineering [8]. MPLS adds one or more independent 32-bit headers (called labels) to the front of a payload for fast switching and tunneling.

The MPLS label stack does not include any internal checksums, and so relies on lower-layer coverage. The indication that the stack is ending and the payload is beginning is a single-bit flag set in the last label; corruption of this flag in any label would be problematic, and could cause unpredictable propagation of incorrectly framed payloads until stopped by some other error condition.

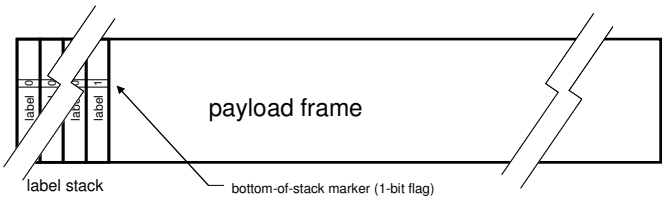
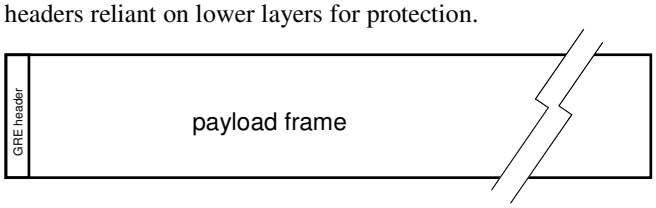


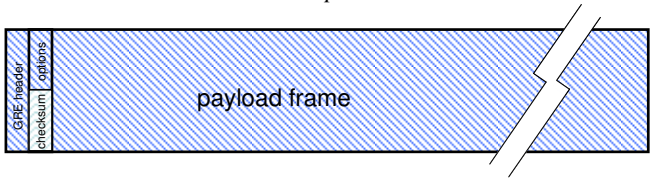
Figure 6. MPLS label stack on front of payload

E. Generic Routing Encapsulation

Generic Routing Encapsulation (GRE) headers [11], used to tunnel packets, consist of a 32-bit header followed by an optional field that includes a one's-complement checksum. This checksum covers both the GRE header and the payload. Often, only the first 32-bit word is present, making GRE headers reliant on lower layers for protection.



a. Common GRE use without options



b. GRE use with optional header including checksum

Figure 7. GRE encapsulation

F. Internet Protocol version 4 (IPv4)

The IPv4 header [12] is robust, in that it contains an internal checksum covering just the IP header, including the IP header options, if any of those are present. This checksum is a simple and relatively weak 16-bit ones-complement checksum, which cannot detect changes in the header such as swapped words.

We deliberately show the IPv4 options header in Fig. 8, even though it is very rarely used, in order to make the separate pseudo-header fields included in the TCP and UDP checksums more visible later.

The length of the header and options, used by the checksum, is given by the Header Length (HL) field. One common criticism of the IPv4 header is that the checksum also covers the Time to Live (TTL) field, which was originally specified as a decreasing timer, but was implemented as a hop limit field in practice. This hop limit is decremented at each hop in the path, requiring the IP header checksum to be recomputed to cover the new hop count value [13][14]. This recomputation and possible injection of errors during the header modification process turns an intended end-to-end header checksum into a *de-facto* per-link header checksum, weakening it considerably. A header checksum that simply skipped the TTL octet would have avoided this problem; recomputing the header checksum at each hop, with the potential to introduce new errors, makes that checksum much less robust.

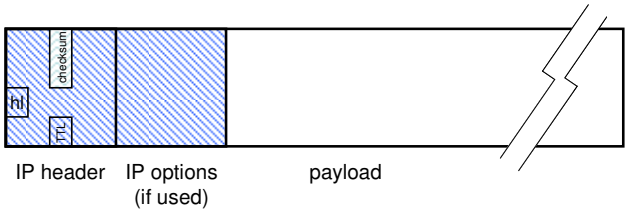


Figure 8. IPv4 header

G. Internet Protocol version 6 (IPv6)

The IPv6 header design [15] removes the header checksum entirely, relying on the TCP and UDP 'pseudo-header', discussed later, and on lower-layer frame checksums to provide a modicum of protection for the header. This makes Network Address Translation (NAT) of IPv6 headers easier.

An IPv6 packet is not in itself internally robust – though a header checksum that did not cover the hop count, traffic class, and Explicit Congestion Notification fields would have avoided the per-hop checksum recomputation as easily as the complete omission of a checksum.

A common IPv6 header is shown in Fig. 9. This diagram neglects jumbograms and the varying optional IPv6 header options, which are also not internally robust.

The lack of a checksum in the IPv6 header is mitigated somewhat by the IPv6 specification requiring UDP to always use checksums when composed with IPv6 and including a pseudo-header construction and a checksum in ICMPv6, in that it helps to ensure that these particular upper layers' data are being delivered to at least the correct destination node and upper layer protocol with a correct source address. However, this measure does not help with IP-in-IP tunneling [16], where IP headers are used without pseudo-header protection. Here, IPv4 tunneling has a clear robustness advantage over IPv6 tunneling, due to the IPv4 header checksum.

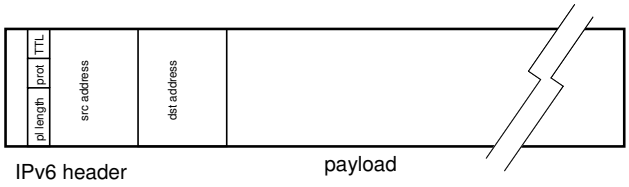


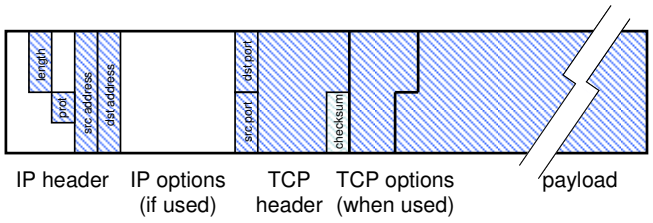
Fig 9. IPv6 header

H. TCP and UDP

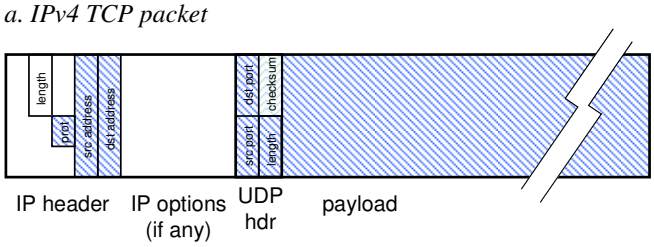
The Transmission Control Protocol, TCP [17], and User Datagram Protocol, UDP [18] protect their internal headers and their entire payloads with 16-bit one's-complement checksums. (Alternate checksum methods for TCP have been specified [19].)

These checksums also protect a number of IP header fields via the inclusion of a 'pseudo-header'. Protecting the address and protocol fields provides a check that the TCP or UDP segment has been demultiplexed in the correct place, while including the packet's length ensures that the payload is neither truncated nor extended. The UDP pseudo-header calculation relies on its own redundant packet length field, rather than using the IP header's packet length field; this is discussed later in the context of UDP-Lite.

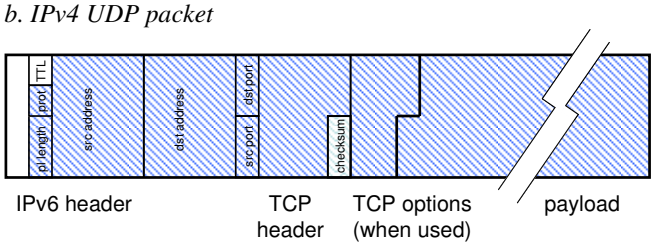
IPv6 is more reliant on this pseudo-header protection than IPv4, thanks to removal of the header checksum that could have detected errors in the IPv6 header earlier in transit.



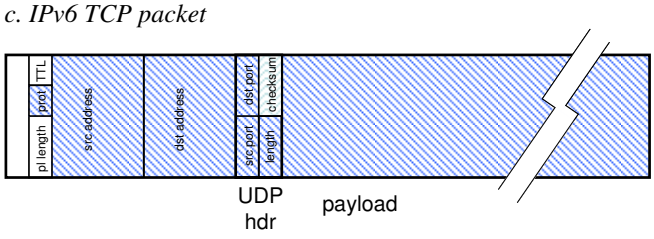
a. IPv4 TCP packet



b. IPv4 UDP packet



c. IPv6 TCP packet



d. IPv6 UDP packet

Figure 10. TCP and UDP packets

UDP checksums are mandatory in IPv6 to provide this demultiplexing protection. Other transport protocols running directly over IPv6 should include a pseudo-header check.

Although the UDP checksum can be turned off in IPv4 for performance reasons, doing so removes the demultiplexing check, and places more reliance on higher-level data checks. Turning off UDPv4 checksums is recognized as a bad idea [2].

I. Stream Control Transmission Protocol (SCTP)

The Stream Control Transmission Protocol, SCTP, began as a way of transporting SS7 (Signalling System 7) telephony control information. SCTP can act as a replacement for TCP for reliable data delivery. SCTP includes support for a number of features that TCP lacks, including multihoming. SCTP originally included an Adler-32 packet checksum [20], but this was later changed to be CRC that is much stronger in that it detects more errors [21]. This CRC is roughly equivalent in strength to the Ethernet CRCs. Although SCTP is carried in IP just like TCP and UDP, it does not include a pseudo-header demultiplexing check. The verification tag, established when an SCTP connection is set up, provides an equivalent demultiplexing check, while individual payload chunks within the SCTP frame, which are padded out to multiple of four bytes, provide their own internal length fields.

SCTP can permit "partial reliability" via setting a limit to ARQ repeat persistence on certain payload chunks [22]. This provides unreliable delivery of data – but all the data received is checksummed so that errored packets can be rejected, and delivery of errored data to applications is precluded by this behaviour.

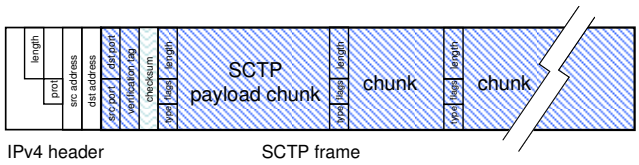


Figure 11. SCTP packet format

V. PROTOCOLS PERMITTING ERRORED CONTENT

Occasionally, where an application and its data stream include a degree of redundancy or are resilient to errors in the received data, fully reliable payload delivery is considered unnecessary (and perhaps even undesirable) by the application.

For instance, several errored bits within a single high-definition television frame may only subtly or imperceptibly alter the video stream when viewed by a human, whereas either the delay in retransmitting that entire frame, or the deletion of that entire frame would be perceptible and degrade the viewing experience.

Having the application accommodate errors in its received data can be particularly useful for:

a. topologies with very long delays, where the added delay and jitter of occasional ARQ protocol resends is considered prohibitive, or even unnecessary and undesirable for real-time traffic such as Voice over IP.
b. unidirectional links, where there is no way to request a frame be resent from the receiver end of the link.

In these cases, data is often heavily forward-error-corrected (FEC'd) in the physical layer for broadcast, with the

expectation that the data should normally arrive correctly. Long-distance unidirectional links are considered common for deep-space probes, where a modified HDLC stream with ARQ disabled can be used effectively [23].

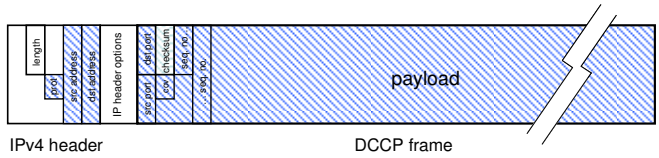
Where errored content can be tolerated, the system of redundant and overlapping checksums at each protocol layer defeats the performance goal of getting errored data to the application. Rather than allowing the application to determine that a couple of bits in the payload may be misset, and decide whether this affects the reliability of the delivered payload, an earlier lower-layer pass-or-fail checksum of its entire frame fails, causing the entire frame to be rejected before the application gets to see its payload. If a resend is not possible, this can turn a single-bit error into a packet-sized hole in the received datastream (i.e. an "erasure"). Although FEC erasure coding protocols [24][25] can be used to recover from transport-level errors without requiring ARQ resends, these protocols do have overheads in terms of network capacity used, trading ARQ resend latency for link capacity.

We now describe four transport-layer protocols that permit delivery of errored payloads. These are DCCP, UDP-Lite, and two new protocols developed in the context of Delay/Disruption Tolerant Networking (DTN) – the Saratoga Transfer Protocol and the Licklider Transmission protocol (LTP).

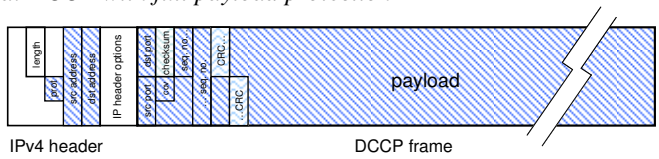
A. *Datagram Congestion Control Protocol (DCCP)*

The Datagram Congestion Control Protocol, DCCP [26][27] implements UDP-like unreliable delivery with flexible congestion control options. Fig. 12 illustrates a DCCP packet using a generic header with a long sequence number(short sequence numbers are also possible in DCCP).

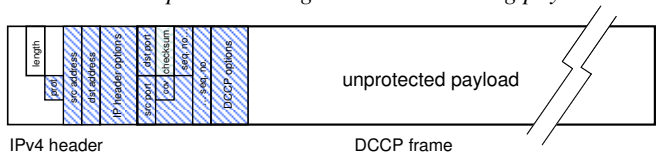
Along with a 16-bit checksum in its header, DCCP includes a checksum coverage field.



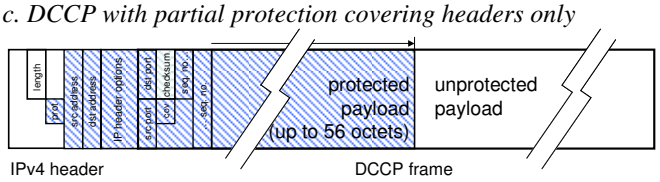
a. DCCP with full payload protection



b. DCCP with optional strong CRC also covering payload



c. DCCP with partial protection covering headers only



d. DCCP partial protection extending to first part of payload

Figure 12. DCCP packet format

When the checksum coverage field is zero, the checksum covers the pseudo-header and all packet contents. When the checksum coverage field contains a value x for which $0<x<16$, the checksum covers the DCCP header, options, pseudo-header and the first $4(x-1)$ octets of the payload. This means that, when the payload is partially unprotected, a maximum of the first 56 octets of the payload can be reliably sent to cover internal payload headers.

DCCP may also include a six-byte option containing a 32-bit CRC, calculated the same way as the SCTP CRC, covering the application data. This optional CRC is itself in turn covered by the header checksum.

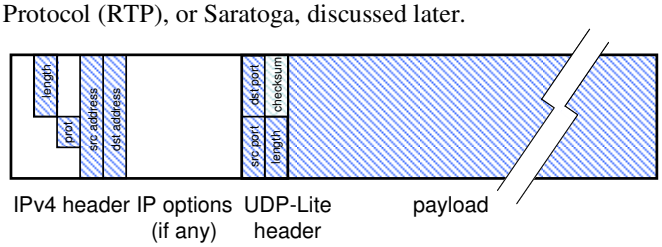
B. *UDP-Lite*

The UDP-Lite transport protocol [28] was originally developed as an after-the-fact modification of the UDP protocol, taking advantage of the redundant packet length fields present in both the IP header and the UDP header. The 'redundant' UDP header's packet length field is turned into an indication of how much of the packet length is checksummed.

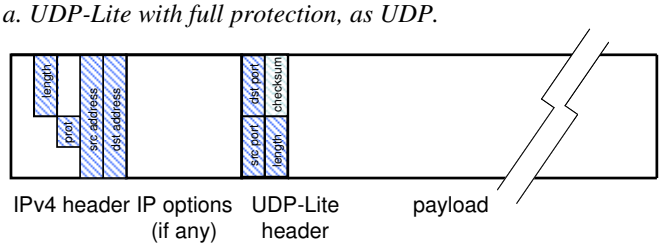
When the length field matches the IP header's length field, UDP-Lite behaves just as UDP does, and is resistant to errors.

When the UDP-Lite length is less that the total packet length, only part of the packet is protected. Thus, like DCCP, UDP-Lite can permit delivery of errored content to applications, which can be useful for e.g. VoIP data. UDP-Lite's variable payload coverage is more flexible and fine-grained that that of DCCP.

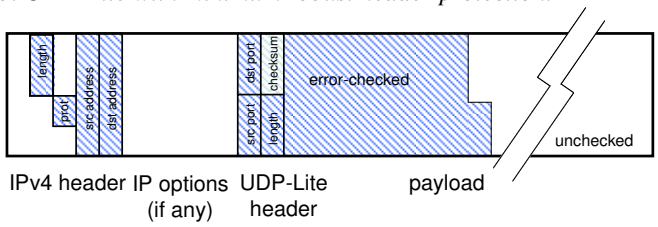
Unlike UDP, UDP-Lite includes the IP header's packet length field in its pseudo-header check. The UDP-Lite length should always be sufficient to provide checksum coverage of the IP pseudo-header, UDP headers, and any other higher layer headers present, e.g. the headers of the Real-Time Protocol (RTP), or Saratoga, discussed later.



a. UDP-Lite with full protection, as UDP.



b. UDP-Lite with minimum robust header protection.



c. UDP-Lite with partial protection dictated by length field

Figure 13. UDP-Lite variable coverage.

UDP-Lite was conceived as a 'compatible upgrade' to UDP, but was eventually given its own, separate, protocol number. This avoids any adverse impact on existing UDP implementations, which rely on the previous semantics of the UDP length field for the pseud-header check, rather than using the IP length field.

The ability of the payload-error-tolerant DCCP and UDP-Lite to convey errored data to applications is predicated on DCCP or UDP-Lite first receiving that errored data at all from underlying layers in the network stack.

If packets of either type are carried within a lower-layer link frame format that checks its payload and rejects errored frames, then errored data is unlikely to reach the transport protocol and the application. This means that UDP-Lite and DCCP are most useful in a protocol stack where only the transport provides a limited end-to-end reliability check, and lower layers only verify the integrity of their own framing headers, without mandatory attempts to verify payload integrity that would prevent useful payload data from reaching the transport layer and the application using it.

In section IV we reviewed common link-layer protocols, which fully protect their payloads. DCCP and UDP-Lite's error tolerance is of little use when run over these protocols, as the link checksums protect against a common form of error – channel noise. However, Stone points out corruption in delivered packets caused by other things than link errors [4]. This corruption can be due to bugs in intermediate processing in routers and interfaces, or due to memory corruption – single event upsets caused by radiation altering RAM contents without error correction coding to detect it are not unknown. This corruption is more likely to be in the payload or at the end of a packet (via truncation or packet contents being overwritten), simply because if the corruption was at the start of the packet, the packet would not be delivered to the right place. So, in tolerating errors in their payloads, DCCP and UDP-Lite are actually making errors in intermediate processing harder to detect.

C. Using UDP-Lite: Saratoga

Saratoga is a file-transfer transport protocol developed for high utilization of dedicated links [29][30].. Saratoga is intended for use in the scenarios where delay-tolerant networking (DTN) [31][32] is appropriate, where connectivity is sporadic and intermittent, and data is transferred hop-by-hop in a store-and-forward approach, rather than along a complete end-to-end path. As well as normal files, Saratoga can also carry streams of data and DTN bundles.

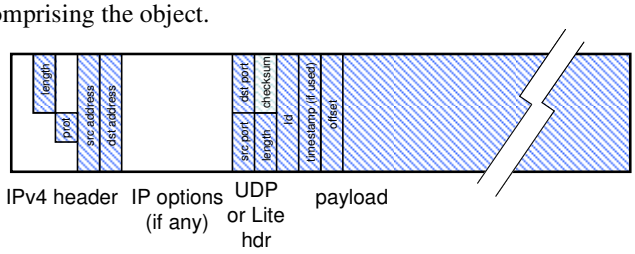
Saratoga uses a negative ARQ mechanism to detect and compensate for packet loss, when desired, and can disable this mechanism within receivers if it is not needed for an application.

Saratoga presumes that either IPv4 or IPv6 is in use on individual links within these delay tolerant networks, so it is designed to run over UDP. This permits Saratoga to also take advantage of the very similar UDP-Lite, if an application prefers and its content to be transferred is error-tolerant. Saratoga thus inherits all of the advantages (fully variable partial coverage) and disadvantages (a weak one's-complement checksum) of UDP-Lite. Using UDP-Lite, Saratoga can deliver errored parts of particular streams, files, or bundles to the application if the sender thinks that that is appropriate for a particular piece of content. However, this errored data will only reach UDP-Lite, Saratoga and the higher application if it has not already been rejected by checksums at lower layers. Saratoga's packet structure remains internally robust, thanks to its reliance on the UDP/UDP-Lite checksum for coverage.

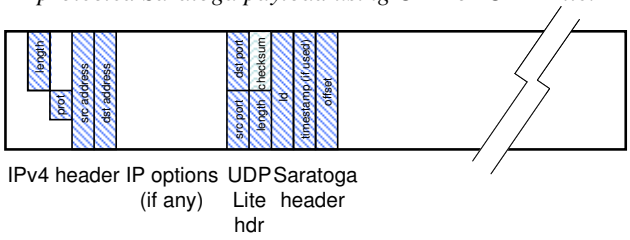
A UDP-Lite/Saratoga packet that is errored in its unchecked unprotected payload part will be passed up the stack and saved as part of a file or bundle. A UDP-Lite/Saratoga packet that is errored in the protected part – any protected payload part or the header information, which must be robust – can prompt an ARQ repeat request. Saratoga also implements an optional end-to-end MD5 integrity checksum over the entire transferred object, to provide assurance of the received and reassembled data. This checksum compensates for the weaker UDP transport checksums, and covers reassembly of the received packets.

This high-level checksum option is useful when data must be transferred reliably end-to-end and the application does not have its own end-to-end check, (An application check is still better, as it covers errors in communication between Saratoga and the application, which Saratoga's MD5 checksum can not.) The UDP/Lite checksum only covers a single Saratoga packet over a single hop between communicating neighbours.

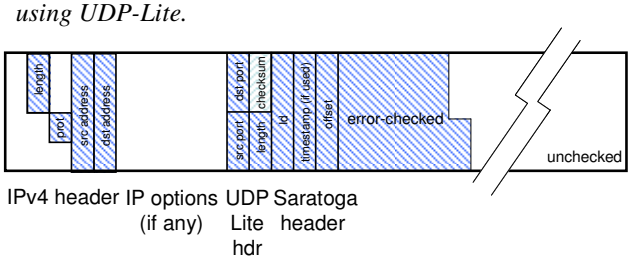
When transfers permitting errors are carried out with UDP-Lite, this MD5 checksum can only indicate if there is a difference between what was sent and what was received in the complete object, not within individual datagrams comprising the object.



a. A-protected Saratoga payload using UDP or UDP-Lite.



b. Robust protected headers and unprotected payload, using UDP-Lite.



c. Saratoga with partial payload protection via UDP-Lite.

Figure 14. Saratoga variable coverage

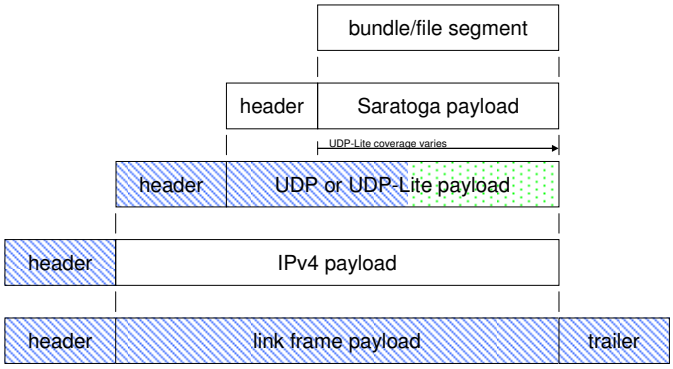


Figure 15. Saratoga use in the IP network stack

Further content checking must be left to the application and its knowledge of the internal structure of the data transferred.

Saratoga's use in the IP network stack, with checksum protection for payloads is shown in Fig. 15. This use is similar to the conventional checksum protection in Fig. 1(c).

D. The Licklider Transmission Protocol

The Licklider (or 'long-range' / 'long-haul') Transmission Protocol (LTP) [33] is, like Saratoga, intended for carrying bundles in DTN scenarios. Although LTP places great emphasis on coping with very long propagation delays, both protocols share similar functionality in this regard.

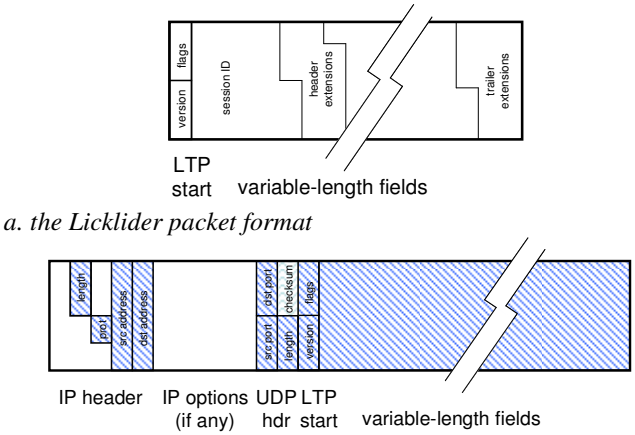
LTP is intended to be carried directly by a wide range of link-layer protocols, but has been prototyped over UDP for ease of development.

The internal format of an LTP packet is quite complex by typical Internet protocol measures, containing a number of variable-length fields which contain variable-length numbers described in a format unique to LTP and DTN protocols (although related to some constructs in ASN.1) – the Self-Delimiting Numeric Value, or SDNV [34]. This stems from a desire to aggressively conserve bits on highly-constrained data links.This complex specification makes the LTP packet format hard to draw. Our attempts to represent LTP packets are shown in Fig. 16.

What is immediately obvious about this packet format is that it is not internally robust, as it does not include a checksum for either its payload or its various header fields, nor does it include any pseudoheader check that would ensure its contents are even being processed by the correct IP node.

LTP can rely on the frame that carries it to provide error-checking, as it does when carried over UDP. When LTP is carried directly within a single lower-layer frame, that frame's error-checking may be sufficient; but if an LTP frame is segmented over the multiple frames of a lower layer and later reassembled, the individual lower layer checksums would not be sufficient to ensure that the LTP packet was correctly reassembled end-to-end.

Somewhat like DCCP and Saratoga/UDP-Lite, LTP claims to permit carriage of 'reliable' and 'partially reliable' payloads – that is, error-free and permitted-error delivery of separate payloads within the LTP protocol. Unlike DCCP and UDP-Lite, which can combine protected and unprotected payload content in one packet, these must be sent in separate LTP frames.



a. the Licklider packet format

b. Licklider over UDP

Figure 16. LTP packet format

These frames are either 'red' (error-free – acknowledged, retransmitted, and one would hope checksummed for integrity) or optional 'green' (errors permitted, unacknowledged, unretransmitted) packets. Red and green packets can be mixed in a transmission. LTP is considered a reliable transport protocol, in that it implements a negative ARQ mechanism. However, the optional green error-permitting packets are not repeated at all if lost. So, a green error-carrying packet with possible permitted bit errors may be turned into a full packet-sized erasure in the data stream if:

a. the green packet is lost in transmission, as it will not be repeated by the sender.
b. the LTP packet is carried by a protocol, such as UDP, that implements its own checksum covering its payload (in this case, the LTP packet), leading to rejection of the packet contents before LTP sees them. This is seen as a loss in transmission, and LTP does not resend green packets.

We have avoided using the term 'partial reliability' seen in the LTP specification, as that conflates delivery of errored content with use of delivery mechanisms with limited persistence. Anything that is partially reliable is, in fact, unreliable; reliability has to be carefully bounded. (SCTP uses 'partial reliability' to denote specified ARQ repeat persistence for checksummed payloads.)

Unlike the combination of Saratoga/UDP-Lite, when transferring error-tolerant payloads, no header fields in LTP green packets are checksummed, trustworthy, or robust. A lower-layer checksum in a frame carrying the LTP packet is desirable, because the LTP packet format does not specify its own internal frame structure checking, and, unlike many of the other protocols described here, is clearly not robust or error-rejecting. Using a lower-layer protocol checksum to protect headers and extensions and make the LTP frame robust will also support red "reliable" packets. Yet a full lower-layer checksum limits the utility of green packets and prevents delivery of errored content to the application; this is a paradox.

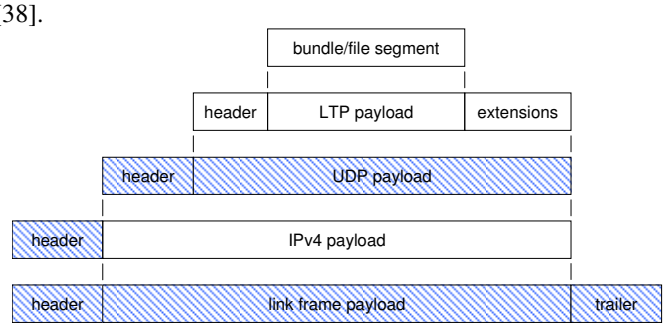
An optional authentication extension is described for LTP in the Licklider extensions draft [35]. This can be used to check the LTP packet for errors and data integrity, independently of checksums at lower layers. The need for checksum-style protection is recognized with inclusion of the optional NULL ciphersuite, which has a hardcoded key and does not require

implementation of a key management framework to protect the LTP segment. This authentication extension can help protect red packets, but does not help varying payloads of 'error-tolerant' green packets. In order to deliver errored bits, the payloads of green packets must forego both authentication and lower-layer checksums.

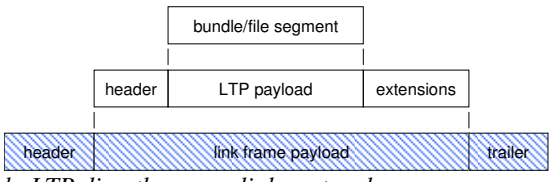
Also of interest is that the red/green packet handling requires considerable detail to explain in the LTP specification, whereas the Saratoga specification is far shorter and simpler due to its layering on top of the pre-existing UDP-Lite. We believe that this kind of component re-use is desirable to prevent the propagation of configuration complexity throughout the protocol stack and aids maintainability LTP is intended to be used directly over a variety of link protocols – although engineering support for LTP in each link protocol can involve expensive development on a case-by-case basis, and individual lower-layer protocols provide differing degrees of bit and frame error detection and correction, leading to different levels of reliability Development of LTP over UDP has been cheaper and easier in the near-term. The resulting protocol stacks are shown in Fig. 17.

UDP provides the simple checksum protection that LTP lacks, leading to a stack that follows Fig. 1(c). When LTP is used directly over a link layer without the optional authentication extension, there is no transport-level protection. This follows Fig 1(d), which is dangerous to data integrity.

If the bundle or file included internal checksum protection for an end-to-end check, lack of protection lower in the stack would not matter, as errors would be caught by the end-to-end check. However, just as the only way to gain internal checking for LTP is to use its optional security framework with the authentication extension, the only way to gain protection for bundles is to use the Bundle Security framework [36], with data integrity being a welcome side-effect of that. Key management becomes necessary because the Bundle Security framework provides no unkeyed authentication or integrity primitives. Lack of any internal bundle protection is also problematic for the proposed bundle-in-bundle encapsulation [37]. We have proposed an integrity checksum for bundles [38].



a. LTP as developed over UDP



b. LTP directly over a link protocol

Figure 17. LTP use in a protocol stack

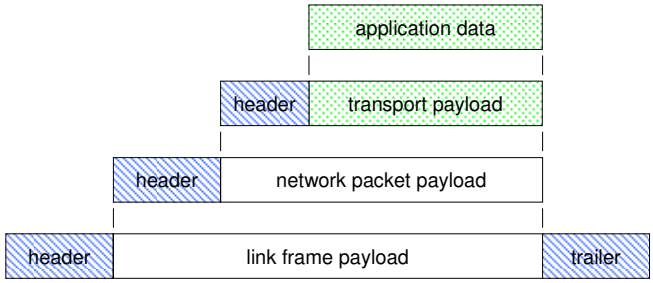


Figure 18. An error-tolerant protocol stack with robust header checks, but no lower-layer payload checks

Like Saratoga, LTP could be adopted to use UDP-Lite on mixed error-tolerant and error-intolerant content. This would require per-packet adjustment of the requested partial checksum coverage and packet size to match boundaries between red and green content types, and either segmentation in order to also provide checksum coverage of trailers used by the LTP extensions, or a rethinking of the trailer concept.

VI. A PROTOCOL STACK THAT DELIVERS ERRORED CONTENT

We can envisage a deliberately-engineered protocol stack of frame formats supporting delivery of errored content, where each frame checks only its own (and, like TCP/UDP, possibly selected lower-layer) header fields to ensure that its own headers are robust, while passing the possibly-errored payloads up to the transport layer and application for final checks and rejection This has been shown conceptually in Fig. 18.

There are no full payload checks, in order to prevent error rejection from occurring before the application sees the data. All headers are checked, to ensure that payload lengths are correct and frames reach the correct destination.

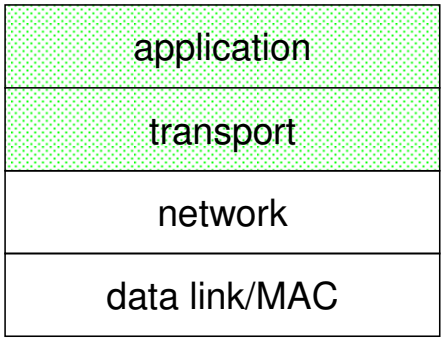
We can summarize the conditions needed for successful delivery of errored content in Fig. 19, which differs somewhat from the full payload protection discussed with Figure 1.

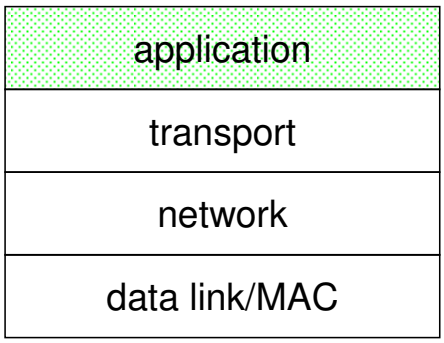
Fig. 19 shows that reduced payload error checking throughout the protocol stack is a necessity for delivery of errored data to work; if all links in the path have payload checksums, channel-induced errors will never reach the transport layer (although the transport layer can still see errors introduced by intermediate processing).

Across a path of multiple links in series, either wired and relatively error-free, or wireless with channel-induced errors, it would be possible to use both strong link payload checksums on the wired links and header-only checksums on the wireless links, and still deliver errored content to the transport protocol in the endhost.

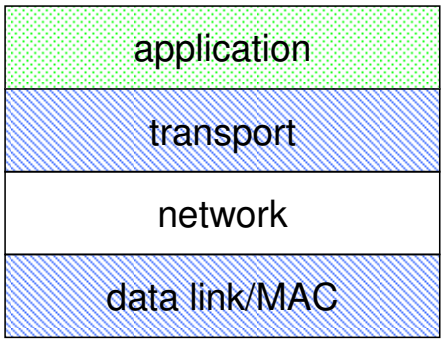
The goal of the levels of protection shown in Fig. 1 was to prevent any errors and ensure overall reliability. Here, the goal, when appropriate for the application, is to control where errors are permitted (the payload), and to protect against errors elsewhere (the important header information).

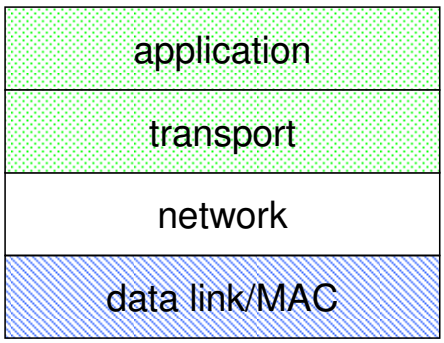
Use of security in any layer can prevent these errors from reaching the application. The cause of corruption of a secured payload cannot be determined, as errors cannot be distinguished from an attack attempt.





a. Delivery of errored payload data is possible





b. Delivery of errored content is prevented; payloads with errors introduced by the physical layer are rejected by lower layers before reaching a layer that can handle them.

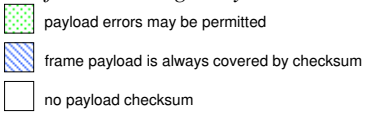


Figure 19. Errored delivery through protocol stacks

Delivery of content where errors have been introduced by channel noise can be permitted by some recent new wireless MAC protocols. We give two examples here

A. IEEE 802.16

The 802.16 Wireless Metropolitan Area Network standard is a link layer protocol used within the WiMax and WiBro frameworks [39].

802.16 includes a link-layer frame CRC and supports reliable transmission via ARQ persistence, but also supports disabling both this CRC and ARQ retransmissions on a per-connection basis.

B. UMTS

Data frames in the Universal Mobile Telephone System (UMTS) link layer have two separately-computed CRC fields, one covering the frame header, and another covering the frame payload [1].

Inclusion and verification of the header CRC is mandatory, while the payload CRC is only optionally generated and verified. Thus, UMTS provides a robust frame format, and also allows for errored delivery.

VII. CONCLUSIONS

We have described and analyzed checksum coverage for protocol stacks, and have examined recently-designed protocols intended for delivery of errored data to applications.

We have shown that, to support delivery of errored data to applications that would benefit from it, a minimal approach to error-checking of payloads must be implemented throughout the protocol stack. For delivery of errored content to work, these protocols must be used over a stack that does not fully checksum its payloads, which is not current common practice.

While DCCP, UDP-Lite and Saratoga/UDP-Lite require a checksum-free lower stack to be able to receive and permit any errored content, these protocols are robust and can be trusted, as their packet headers are always covered by a checksum. We find a paradox in LTP, in that packets that permit errors (the 'green packets') require a lower stack free of payload checksums – yet LTP itself is the weakest of the four error-tolerant protocols examined, in that it does not attempt to protect the integrity of its own headers, extension data, or payloads – i.e. that LTP is not internally robust by design. LTP needs protection from lower-layer checksums to help prevent errors, or must use the optional authentication header in its security framework in order to ensure that its format can be delivered without errors and decoded reliably.

Specifying use of LTP over UDP-Lite may help resolve the paradox of LTP green packets. The lack of header and payload integrity in the DTN bundle format specification, requiring implementation of the security framework to get authentication, is also of concern, as it can be impractical or unworkable to implement a full security framework with key management in many realistic scenarios.

VIII. ACKNOWLEDGMENTS

We thank Kevin Fall and the IRTF Delay-Tolerant Networking Research Group for the discussions that led to recognition of the need for this paper.

We thank Cathryn Peoples for her comments on drafts of this paper.

REFERENCES

[1] J. Saltzer, D. Reed and D. Clark, "End-to-End Arguments in System Design," ACM Transactions in Computer Systems, pp. 277-288, November 1984.

[2] C. Partridge and L. Jolitz, "Applications with UDP checksum disabled", emails to the end2end-interest mailing list, http://www.postel.org/pipermail/end2end-interest/2002-March/, March 2002.

[3] J. Stone, M. Greenwald, J. Hughes, and C. Partridge, "Performance of checksums and CRCs over real data," IEEE Transactions on Networks, vol. 6 issue 5, pp. 529-543, October 1998.

[4] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," Proceedings of ACM Sigcomm, pp. 309-319, September 2000.

[5] ISO13239 *High-level data link control (HDLC) procedures*.

[6] W. Simpson, "The Point-to-Point Protocol," IETF RFC 1661, July 1994.

[7] W. Simpson, "PPP in HDLC-like framing," IETF RFC 1661, July 1994.

[8] International Telecommunication Union, *ISDN Data Link Layer Specification for Frame Mode Bearer Services*, ITU-T Recommendation Q.922, 1992.

[9] International Telecommunication Union, "B-ISDN ATM Adaptation Layer (AAL) Specification", ITU-T Recommendation I.363, 1993.

[10] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture", IETF RFC 3031, January 2001.

[11] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, "Generic Routing Encapsulation", IETF RFC 2784, March 2000.

[12] J. Postel (ed.), "Internet Protocol – DARPA Internet Program Protocol Specification", IETF RFC 791, September 1981.

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

[14] J. Touch and B. Parham, "Implementing the Internet Checksum in Hardware," IETF RFC 1936, April 1996.

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

[16] A. Conta and S. Deering, "Generic Packet Tunneling in IPv6 Specification", IETF RFC 2473, December 1998.

[17] J. Postel, "Transmission Control Protocol," IETF RFC 793, September 1981.

[18] J. Postel, "User Datagram Protocol," IETF RFC 768, August 1980.

[19] J. Zweig and C. Partridge, "TCP alternate checksum options," IETF RFC 1146, March 1990.

[20] R. Stewart *et al*., "Stream Control Transmission Protocol," IETF RFC 2960, October 2000.

[21] J. Stone, R. Stewart and D. Otis, "Stream Control Transmission Protocol (SCTP) Checksum Change," IETF RFC 3309, September 2002.

[22] R. Stewart *et al*., "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension," IETF RFC3528, May 2004.

[23] K. Hogie, E. Criscuolo and R. Parise, "Using standard Internet Protocols and applications in space," Computer Networks, vol. 47 no. 5, pp. 603-650, April 2005.

[24] M. Watson, "Forward Error Correction (FEC) Framework", work in progress as an internet-draft, February 2007.

[25] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, "Forward Error Correction (FEC) Building Block," IETF RFC 3452, December 2002.

[26] E. Kohler, M. Handley and S. Floyd, "Datagram Congestion Control Protocol (DCCP)," IETF RFC 4340, March 2006.

[27] E. Kohler, M. Handley and S. Floyd, "Designing DCCP: Congestion Control Without Reliability," Proceedings of ACM Sigcomm, pp. 27-38, September 2006.

[28] L. A Larzon, M. Degermark, *et al*., "The Lightweight User Datagram Protocol (UDP-Lite)," IETF RFC 3828, July 2004.

[29] L. Wood, W. Eddy, W. Ivancic, J. McKim and C. Jackson, "Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization," Third International Workshop on Satellite and Space Communications (IWSSC '07), September 2007.

[30] L. Wood, W. Eddy, W. Ivancic, *et al*., "Saratoga: A Convergence Layer for Delay Tolerant Network," work in progress as an internet-draft, July 2007.

[31] V. Cerf, S. Burleigh *et al*., "Delay Tolerant Network Architecture," IETF RFC 4838, April 2007.

[32] K. Scott, K. and S. Burleigh, "Bundle Protocol Specification," work in progress as an IRTF internet-draft, July 2007.

[33] M. Ramadas S. Burleigh and S. Farrell, "Licklider Transmission Protocol – Specification," work in progress as an IRTF internet-draft, April 2007.

[34] W. Eddy, "Using Self-Delimiting Numeric Values in Protocols," work in progress as an internet-draft, July 2007.

[35] S. Farrell, M. Ramadas and S. Burleigh, "Licklider Transmission Protocol – Extensions," work in progress as an IRTF internet-draft, April 2007.

[36] S. Symington, S. Farrell, and H. Weiss, "Bundle Security Protocol Specification", work in progress as an internet-draft, April 2007.

[37] S. Symington, R. Durst and K. Scott, "Delay-Tolerant Networking Bundle-in-Bundle Encapsulation," work in progress as an IRTF internet-draft, May 2007.

[38] W. Eddy and L. Wood, "The DTN Bundle Protocol Payload Checksum Block," work in progress as an internet-draft, July 2007.

[39] IEEE 802 LAN/MAN Standards Committee, "Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1", IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor1-2005, February 2006.

[40] 3rd Generation Partnership Project (3GPP) Technical Specification Group Radio Access Network (TSG RAN), "UTRAN lub Interface User Plane Protocols for Common Transport Channel Data Streams", 3GPP TS 25.435, V4.4.0, March 2002.