EURASIP Journal on Applied Signal Processing 2005:2, 242–247
© 2005 Hindawi Publishing Corporation

Passing Corrupt Data Across Network Layers: An Overview of Recent Developments and Issues

Michael Welzl

Distributed and Parallel Systems Group, Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria
Email: michael.welzl@uibk.ac.at

Received 6 August 2003; Revised 16 June 2004

Recent Internet developments seem to make a point for passing corrupt data from the link to the network layer and above instead of ensuring data integrity with a checksum and ARQ. We give an overview of these efforts (the UDP Lite and DCCP protocols) and explain which circumstances would justify delivery of erroneous data; clearly, the missing piece in the puzzle is efficient and meaningful interlayer communication.

Keywords and phrases: UDP Lite, DCCP, TCP, checksum, wireless links, link layer ARQ.

1. INTRODUCTION

Network layers are a powerful concept of abstraction; a programmer of a web browser cannot be aware of lower-layer issues such as routing or even parity control. The ISO/OSI model is very valuable as a means to classify mechanisms, thereby facilitating communication among network professionals as well as teachers and their students. From a purely technical, efficiency oriented perspective however, it turns out that strict layering can lead to duplicated functionality (checksums in TCP, IP, and underneath) as well as misused (TCP over the ATM "available bit rate" (ABR) service), inefficient (if a frame belonging to a large IP packet is lost, the whole packet becomes useless), or even unusable technology (ATM and Internet QoS).

The OSI model sometimes seems to contradict the important design principles (in a sense, they have become commandments for Internet designers) called "end-to-end arguments," which say that functions required by communicating applications can be correctly and completely implemented only with the knowledge and help of the applications themselves [1]. Basically, this means that most decisions should be left up to applications and functions should be moved upward in the stack as far as possible.[1]

In this paper, we are concerned with one particular function which is normally (but should not always be) performed at the data link layer: checking a packet or frame for transmission errors, discarding it, and maybe requesting its retransmission if the checksum fails. After looking at two transport protocols that encompass features which only make sense if the link layer hands over corrupted data, we discuss what it would really mean to do so in Section 4. Possibilities to cope with these problems via interlayer communication are discussed in Section 5; Section 6 concludes.

2. UDP LITE

The "UDP Lite" protocol, a simple yet far reaching proposal, has been under discussion in the IETF for several years [2]. It differs from UDP in the following aspects.

(i) The redundant (except when padding is used after the UDP payload, which does not fill any purpose) "length" field in the UDP header is replaced by a "checksum coverage" field.

(ii) The checksum now covers the UDP Lite header and the so-called "pseudoheader" (some fields from the IP header which are used to ensure end-to-end integrity) and, depending on the checksum coverage field, a part of the payload. A UDP Lite packet with a checksum coverage value equal to the packet length (or 0) would be treated just as a normal UDP packet whereas a value of 8 means that only the header is checked and payload errors are ignored.

This simple design was motivated by ease of deployment and being backward compatible with UDP; it introduces extra flexibility without adding virtually any complexity.

[1] Notably, the end-to-end arguments also explicitly mention that an incomplete version of the function provided by the communication system may sometimes be useful as a performance enhancement. These rules are frequently misinterpreted as being more strict than they really are ("keep the inner network as dumb as possible at all costs").

The goal of UDP Lite is to support applications which would rather have damaged data delivered than discarded by the network. This includes, for example, real-time multimedia applications using codecs that are designed to cope with such errors. The main problem with this concept is that link layers typically do not pass erroneous data to the network layer, and UDP Lite will hardly encounter an erroneous packet. The protocol was therefore tested using a patched device driver [3].

In reality, it turns out that one partial checksum may not even be enough for certain applications. For instance, the standard for RTP transmission of "adaptive multirate" (AMR) and "adaptive multirate wideband" (AMR-WB) encoded speech signals allows to send several speech frames within one RTP packet, each of which could correspond with an optional CRC [4]. Since it would obviously not make sense to specify support for an arbitrary number of checksums with corresponding coverage fields in UDP Lite, such an application would simply have to restrict the checksum to the header by setting the checksum coverage field to 8 and realize the several embedded payload checksums itself.

3. DCCP

A new and very promising protocol that is currently being developed in the IETF is called "datagram congestion control protocol" (DCCP). Unlike TCP, it provides a connectionless service and is intended to be used by many of the applications that were implemented directly on top of UDP so far. Basically, DCCP provides access to existing congestion control mechanisms in a uniform manner, thereby allowing application programmers to merely choose between a number of available schemes instead of implementing an appropriate one from scratch (which is supposedly bothersome enough to make some programmers abandon the idea of deploying any kind of congestion control at all). DCCP is a tool that can finally help turn some of the existing "TCP-friendly" (i.e., fair towards TCP) congestion control ideas into real services.

In addition to this basic functionality, the DCCP specification encompasses a wealth of other features—some examples are authentication via nonces, path MTU discovery (a mechanism to detect the ideal packet size), support for explicit congestion notification (ECN) [5], and last but not least, partial checksums as in UDP Lite in order to support applications that can cope with known corrupt data.

Since, unlike UDP Lite, DCCP is performing congestion control, the means of interpreting transmission errors had to be refined; thus, in its present state, the specification encompasses a so-called "data checksum option" which was proposed by the author of this paper [6]. It works as follows.

(i) An additional checksum is introduced; if the regular checksum does not cover all of the packet, the remaining part is covered by the new one.

(ii) The idea is to support applications which require error-free data while making use of corrupt packets for congestion control. In this context, two different things can happen.

(1) A packet arrives with ECN = 1 in the IP header. The receiver can now notify the sender that congestion has occurred, leading to earlier and possibly more precise congestion detection: a lost (unacknowledged) packet could invoke a timeout, which usually corresponds with a more severe reaction than an ECN notification. TCP, for example, would enter the "slow start" phase instead of staying in "congestion avoidance" state and halving the congestion window.

(2) A packet arrives with ECN = 0 in the IP header. The message to the sender is "corruption has occurred." Since corruption does not necessarily correspond with congestion, one could imagine designing a different—perhaps less conservative—reaction to such a notification.

In any case, the intact header information from such an acknowledgment could be used to drive the RTT estimation, and certain control information are retained.

(iii) Optionally, a DCCP implementation may provide an API through which an application can access the corrupt data. In this case, the advantage of the data checksum option is the fact that application is explicitly informed about corruption.

These issues, and especially the idea of a less strict response to corruption than to congestion, raised a lot of questions and led to long discussions in the IETF. Eventually, it was decided that they must be seen in a broader context; for example, it does not suffice to talk about DCCP and UDP Lite when interpretation of corruption notifications could also be relevant for TCP. The pros and cons were presented in the "IAB Plenary" session of the 57th IETF Meeting which took place in Vienna, Austria. It thoroughly subsumed the current state of the discussions. In what follows, we will take a closer look at some of the problems that were mentioned during this presentation (which does not necessarily mean that a conveyed opinion matches the presenter's point of view) and discuss the tradeoffs when tuning (or even disabling) link layer ARQ.

4. ISSUES

4.1. The link layer perspective

One recurring argument against protocols like UDP Lite is that most link layers already take care of corrupted data via FEC and ARQ, and that obtaining corrupted data at the transport layer is therefore a false notion. Clearly, this is a "chicken-and-egg" problem: link layer designers will not be motivated to disable ARQ unless there is a mechanism on top which shows advantages with corrupt data.[2] While designing, standardizing and deploying such a mechanism that will not really show an immediate advantage is quite critical, it has the potential to change the situation.

[2] Tuning link layer mechanisms is often a tradeoff among various factors including system economics, application tolerance, and other things; such considerations are beyond the scope of this paper.

4.2. IPv6

CPU cycles are scarce in core Internet routers; since the IP header changes at each hop (e.g., a router must decrease the IPv4 "time-to-live (TTL)" field ("hop limit" in IPv6) by at least 1 on forwarding), it is necessary to constantly recompute the header checksum. As most links use a CRC anyway, this is usually a waste of costly processing power. In addition, IETF transport layer protocols are typically designed to include the relevant fields of the IP header (a so-called pseudoheader) in their checksum calculation in order to ensure end-to-end integrity at all times. For IPv6, it was therefore decided to remove this checksum altogether. As a compensation, transport protocols are required to include a pseudoheader consisting of the source address, destination address, upper-layer packet length, and next header field which identifies the type of header immediately following the IPv6 header. This means that fields which convey per-hop semantics—the "flow label," which is used for quality of service, the "version" field which is used to identify IPv6, IPv4, and other protocols, and the hop-by-hop options which can define nodes that must be "visited" along the way to the destination—remain unchecked by the network layer [7].

The problem with this design is that it makes assumptions about underlying layers, thereby conflicting with the common notion of "IP over everything." When IPv6 was crafted, UDP Lite was not around. Now, IPv6 aggravates the chicken-and-egg problem: if we deploy UDP Lite, how are we going to convince a link layer designer to disable payload checksums when this can mean sending IPv6 packets to Mars (in accordance with the vision of the "interplanetary Internet")?

4.3. Encryption and authentication

Obviously, one has to be very careful about why and when to use erroneous packets: even a few damaged bits of encrypted data can destroy a large chunk of data. According to [2], the decryption transform will typically spread errors such that the packet becomes too damaged to be of use. Notably, there are encryption mechanisms called "stream ciphers" which do not spread errors in this way provided that the damage occurs in the insensitive part of the packet.

Authentication mechanisms require a path to provide end-to-end data integrity, that is, having the payload change on the way to the receiver is completely unacceptable. It must therefore be ensured that such mechanisms only operate on the checked part of a packet (or partial checksums are simply disabled). These issues do not seem to play a severe role. It is simply a matter of knowing when and how to use partial checksums.

4.4. Congestion versus corruption

As explained earlier, congestion control mechanisms normally rely on packet loss as an indicator of network overload. This can lead to a misinterpretation of loss from corruption as a sign of congestion. Partial checksums may circumvent this problem by allowing an end node to better distinguish between these two network effects. Here, the problem is that the information "a packet experienced corruption" can be misleading; some congestions are in fact known to manifest as corruption, for example, on shared wireless links. Apparently, there is no thorough study available which clearly shows when corruption is caused by congestion and when it is not; the Internet community still seems to be somewhat unsure about the best way to react upon a corruption notification.

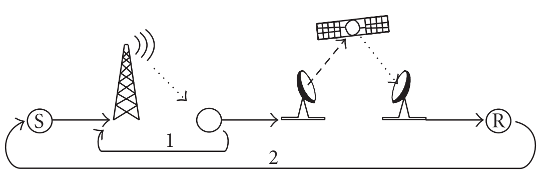

Figure 1: An example of the scenario showing advantageous link layer ARQ.

4.5. Link layer ARQ considerations

Perhaps the greatest problem with link layer ARQ is that sometimes it is more efficient than handing over corrupt data, and sometimes it is not. For instance, if the transport protocol is TCP (or TCP-friendly, as could be expected of DCCP), retransmission of frames at the link layer basically means installing a control loop on top of a control loop; such design is known to cause strange interactions and was studied thoroughly in the case of TCP over ATM ABR. The specific problem with TCP and link layer ARQ is that local retransmissions artificially prolong the end-to-end round-trip time (RTT). The estimated RTT, which is continuously updated with each acknowledgment using an exponentially weighted moving average function, plays a crucial role in TCP: its stability depends on a property called "ACK clocking" (sometimes also called the "conservation of packets principle"), which is based on the RTT estimate. In equilibrium, a packet should only be sent into the network when a packet has left the network [8].

Feedback delay also has a direct impact on the rate of a TCP sender. The following equation models the steady-state behaviour of TCP:

$$T = \frac{s}{R\sqrt{2p/3} + t_{\mathrm{RTO}}\left(3\sqrt{3p/8}\right)p\left(1 + 32p^2\right)}. \tag{1}$$

Here, the sending rate T is described as a function of the packet size s, RTT R, steady-state loss event rate p, and the TCP retransmit timeout value t_{RTO} (roughly $4R$) [9].

On the other hand, in a scenario like the one depicted in Figure 1, enabling link layer ARQ is clearly beneficial: the feedback delay (RTT) of loop 1 is much shorter than the feedback delay of the end-to-end TCP connection from sender S to receiver R (loop 2). Link layer optimization is also a convenient and an easily deployable way of enhancing performance because it is transparent to the end systems.

In this example, it is hardly possible for the nodes involved in loop 1 to be aware of the satellite connection; if strict layering is enforced, it *cannot* be aware of it. There may be many TCP connections on a wireless link, and an unknown number of them may resemble the S-R connection in Figure 1 unless the wireless link layer can distinguish between individual network layer flows (distinguishing between two transport flows is unnecessary up to now, but this may change as transport level multihoming is deployed [10]). Ideally, link layer ARQ should be tuned (in varying degrees of persistence [11]) for each individual flow based on the signal-to-noise ratio and (1), which means that the link layer would have to be aware of all the variables involved.

If the end node is capable of utilizing the information "a packet is corrupt," the decision for or against link layer ARQ would ideally have to involve the specific advantage gained with this kind of message in addition to all of the factors above. Given the complexity, it seems to be impossible for the link layer to make a sensible decision on its own. At this point, it may be helpful to remember the end-to-end arguments; *Functions required by communicating applications can be correctly and completely implemented only with the knowledge and help of the applications themselves. Providing these functions as features within the network itself is not possible* [1].

The arguments have several facets, and, according to [12], two complementary goals:

(1) higher-level layers, more specific to an application, are free to (and thus expected to) organize lower-level network resources to achieve application-specific design goals efficiently (*application autonomy*);

(2) lower-level layers, which support many independent applications, should provide only resources of broad utility across applications, while providing to applications usable means for effective sharing of resources and resolution of resource conflicts (*network transparency*).

In our case, it seems to make sense if we apply these rules as follows:

(1) higher-level layers, more specific to an application, should have some means to "organize lower-level network resources," that is, communicate their requirements to lower layers;

(2) lower-level layers, which support many independent applications, should use mechanisms of broad utility across applications (general purpose, tunable link layer ARQ), while reacting to commands from applications.

In other words, since only higher layers can be expected to be aware of what they need and the requirements that should be met, the solution seems to be link layer ARQ with various degrees of persistence, to be tuned and/or disabled via effective interlayer communication. In what follows, we will take a look at some possibilities to enable such communication (for simplification, we restrict our observations to the possibility of enabling/disabling link layer ARQ and notifying end nodes of corruption).

5. INTERLAYER COMMUNICATION

5.1. TrigTran: corruption experienced

There was a recent IETF effort called "triggers for transport" (TrigTran) (now continued in the framework of another effort called "access link intermediaries assisting services" (ALIAS) available at http://mailman.berkeley.intel-research.net/mailman/listinfo/alias), which is about messages ("triggers") between transport end points (S and R in Figure 1) and the network as a performance enhancement. In addition to notifications for links going down or up, a message called "corruption experienced" was discussed. So far, there does not seem to be a concise outcome, which is apparently due to the aforementioned difficulties with the interpretation of such a message.

It is an open question when a corruption experienced notification should be sent, and who should receive it. Questions of timescale and scalability arise, and having intermediate systems that send messages to end nodes on their own has been identified as a bad idea before [13].

5.2. Transport protocol detection

The UDP Lite specification [2] makes a set of recommendations for link layers; basically, UDP Lite packets should be identified, and there should only be partial error detection for such packets. This means that link layer end points (the left and right ends of loop 1 in Figure 1) must examine each and every packet and, in the case of fragmented datagrams carrying UDP Lite payload, maintain state until all fragments were received or a timeout occurred. Keeping per-flow state is a highly critical operation as it is known not to scale well.

Internet quality of service efforts have therefore evolved from a completely stateful to a hierarchical approach, where core routers are not aware of individual connections and edge routers of a domain do most of the work. Thus, one cannot expect a core router (or an equivalently overloaded network node) to keep per-flow state, that is, fully support the implicit interlayer communication model suggested in [2]. However, if the network node in question resembles a typical edge router in terms of resource availability, the method may be feasible.

5.3. Other possibilities

There are, of course, numerous other ways to inform the data link layer that corrupted packets should be handed over to the network layer instead of performing link layer ARQ or notifying an end system about corruption within the network. However, each method seems to have its distinctive disadvantages.

As an example, to circumvent the problems that occur when the data link layer has to examine the transport layer, one might consider introducing a corruption acceptable (and/or corruption experienced) bit in the IP header. The IP header, however, has no bits to spare. As an alternative, it would be possible to design a new IP option.[3] On the other hand, common Internet routers are known to process "standard" packets in the so-called "fast path" (pure hardware) whereas packets requiring special treatment end up in the "slow path" (software or mixed hardware/software processing). Option interpretation is one such special treatment because the work involved is hard to determine beforehand.

The negative effects of delaying IP packets (artificially prolonging the RTT) were already discussed at the beginning of this section. IP options should therefore be used with care. While per-flow state could be avoided by adding an IP option to every packet of a connection, the extra delay perhaps outweighs the advantage. Sometimes, routers are configured to handle only a certain number of IP options per second. This kind of configuration makes the use of IP options in all packets pointless. If only a certain number of packets carry the special option, the data link layer has to keep track of transport flows, and the advantage over transport protocol detection is lost. The same is true for extra signaling from end points to the data link layer (e.g., an ICMP message carrying a corruption-acceptable notification).

It seems as though the best method to set up communication between the transport and data link layers remains to be found; sensibly introducing such a mechanism would require to carefully study the tradeoffs in a large number of possible scenarios—a work that has yet to be carried out.

6. CONCLUSION

In this paper, we examined some recent developments that have been taking place in the IETF. The underlying idea is that it may sometimes make sense to pass corrupt data from the data link layer to the transport layer and above—a concept that brings about a large number of design issues and led to many discussions in the related IETF working groups.

The list of developments that were presented in this paper is, of course, not comprehensive. An immense amount of work has been done in the area of TCP enhancements for wireless links, ranging from ideas of connection splitting to more sophisticated mechanisms like the "Snoop protocol." A good overview can be found in [14, 15], which includes a proposal for a new cross-layer communication method. Also, there are mechanisms like explicit congestion notification (ECN) [5], which is the congestion counterpart to a corruption-experienced message; while it is not sensible to solely rely on ECN and ignore packet loss for various reasons, such usage would avoid the typical misinterpretation that occur with noisy links. Thus, ECN is at least a step in the direction of separating these two fundamental network effects.

Another example is the "congestion avoidance with distributed proportional control" (CADPC) mechanism, which exclusively relies on explicit traffic feedback from the "performance transparency protocol" (PTP) and ignores packet loss; this mechanism, which has been shown to perform very well with long-term flows in isolated simulations, does not have the problem of misinterpreting corruption for congestion [16]. "TCP-HACK" is a TCP enhancement based on partial checksums; it was shown to perform well in a large number of scenarios and implemented for Linux [17]. A similar TCP enhancement is currently under discussion in the IETF [18].

Given all these efforts and the fact that none of them are deployed, it seems to be clear that the main issue is the lack of effective and well-designed interlayer communication. To solve this problem would mean to carry out a thorough investigation of pros and cons that come with various communication methods; it is a purpose of this paper to motivate such research.

ACKNOWLEDGMENTS

The author would like to thank everybody who contributed to the UDP Lite and DCCP discussions on partial and separate checksums in the TSVWG and DCCP Working Groups of the IETF. In particular, this includes Gorry Fairhurst, Aaron Falk (who held the IAB Plenary Presentation entitled "Passing errored-packets to applications"), Eddie Kohler, and Colin Perkins.

REFERENCES

[1] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," in *Proc. 2nd International Conference on Distributed Computing Systems (ICDCS '81)*, pp. 509–512, Paris, France, April 1981.

[2] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson Ed., and G. Fairhurst Ed., "The lightweight user datagram protocol (UDP-Lite)," RFC 3828, July 2004.

[3] L.-A. Larzon, M. Degermark, and S. Pink, "UDP lite for real-time multimedia applications," in *Proc. IEEE International Conference of Communications (ICC '99)*, Vancouver, British Columbia, Canada, June 1999.

[4] J. Sjoberg, M. Westerlund, A. Lakaniemi, and Q. Xie, "Real-time transport protocol (RTP) payload format and file storage format for the adaptive multi-rate (AMR) and adaptive multi-rate wideband (AMR-WB) audio codecs," RFC 3267, June 2002.

[5] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ECN) to IP," RFC 3168, September 2001.

[6] E. Kohler, M. Handley, and S. Floyd, "Datagram congestion control protocol (DCCP)," Internet Draft, February 2004, http://www.icir.org/kohler/dccp/draft-ietf-dccp-spec-06.txt.

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

[8] V. Jacobson and M. Karels, "Congestion avoidance and control," in *Proc. ACM SIGCOMM Symposium on Communications Architectures and Protocols (ACM SIGCOMM '88)*, pp. 314–329, Stanford, Calif, USA, August 1988.

[9] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," in *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM SIGCOMM '98)*, pp. 303–314, Vancouver, British Columbia, Canada, August–September 1998.

[3] It has been brought to the author's attention that such an idea was once proposed in the IETF as a "crappy link IP option."

[10] R. Stewart, Q. Xie, K. Morneault, and et al., "Stream control transmission protocol," RFC 2960, October 2000.

[11] G. Fairhurst and L. Wood, "Advice to link designers on link automatic repeat request (ARQ)," RFC 3366, August 2002.

[12] D. P. Reed, J. H. Saltzer, and D. D. Clark, "Comment on active networking and end-to-end arguments," *IEEE Network*, vol. 12, no. 3, pp. 69–71, 1998.

[13] S. Floyd, "ECN vs. source quench," Technical Note, November 1997, http://www.icir.org/floyd/ecn.html.

[14] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," in *Proc. ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM SIGCOMM '96)*, pp. 256–269, Stanford, Calif, USA, August 1996.

[15] R. Krishnan, J. Sterbenz, W. Eddy, C. Partridge, and M. Allman, "Explicit transport error notification (ETEN) for error-prone wireless and satellite networks," *Computer Networks*, vol. 46, no. 3, pp. 343–362, 2004.

[16] M. Welzl, *Scalable Performance Signalling and Congestion Avoidance*, Kluwer Academic, Norwell, Mass, USA, 2003.

[17] R. K. Balan, B. P. Lee, K. R. R. Kumar, L. Jacob, W. K. G. Seah, and A. L. Ananda, "TCP HACK: TCP header checksum option to improve performance over lossy links," in *Proc. 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '01)*, vol. 1, pp. 309–318, Anchorage, Alaska, USA, April 2001.

[18] M. Welzl, "TCP corruption notification options," Internet Draft, June 2004, http://www.welzl.at/research/publications/draft-welzl-tcp-corruption-00.txt.

Michael Welzl received his M.S. degree from the University of Linz, Austria, and passed his Ph.D. defense at the University of Darmstadt, Germany, with distinction, in November 2002. His thesis "Scalable performance signalling and congestion avoidance," which was written under the supervision of Professor Max Muehlhaeuser (University of Darmstadt, Germany) and Professor Jon Crowcroft (University of Cambridge, United Kingdom), introduced the PTP protocol and showed its benefits in isolation; a revised version of this work is available at Kluwer Academic Publishers. Dr. Welzl spent two years as a Research Assistant at the Telecooperation Department, University of Linz, Austria, before joining the faculty of the newly founded Institute of Computer Science, the University of Innsbruck, Austria, in November 2001, where he is doing research on computer networks with a focus on grid computing.