UDP-Liter: An Improved UDP Protocol for Real-Time Multimedia Applications over Wireless Links

Patrick Pak-kit Lam
Department of Information Engineering
Chinese University of Hong Kong
Shatin, Hong Kong
lampk3@ie.cuhk.edu.hk

Soung C. Liew, *Senior Member, IEEE*
Department of Information Engineering
Chinese University of Hong Kong
Shatin, Hong Kong
soung@ie.cuhk.edu.hk

***Abstract*— For wireless real-time multimedia applications, the excessive packet loss rate suffered at the receiver, especially when the packets are transported on UDP, is a major issue to resolve. The previously proposed UDP-Lite project was designed to replace traditional UDP, so that erroneous UDP payloads could be passed up to the application layer instead of being discarded. The main argument of UDP-Lite is that many real-time multimedia applications actually prefer damaged packets over lost ones because of their error resilience capabilities. The major drawback of UDP-Lite is the backward incompatibility against traditional UDP stacks in numerous UDP devices. In addition, applications have no choice but to accept UDP-Lite packets as normal ones. This paper introduces another approach, UDP-Liter, which makes use of the concept of UDP-Lite, and yet maintains 100% backward compatibility. Better yet, UDP-Liter will give applications the flexibility to handle normal packets and corrupted packets differently.**

Keywords-UDP-Lite; UDP; Multimedia Applications; Wireless;

I. INTRODUCTION

Many people in the wireless industries, including WLAN and cellular, believe that real-time multimedia applications (RTMAs), such as voice and video, will be the next "killer applications" for the wireless IP networks. Unfortunately, excessive packet loss in UDP (real-time applications usually use UDP as their transport layer) is one of the most mentioned challenges yet to be overcome. Investigations show that many of these "lost" packets have actually made through the network to the destination, but are discarded by the UDP protocol stack [1]. The major reason is that they fail the checksum and are considered useless by the UDP protocol stack. Let us consider a bit-error rate of 10^{-3} normally assumed in wireless environments (the BER in wireless channel normally ranges from 10^{-5} to 10^{-3} [2]), the packet removal rate at UDP could exceed 30% (voice UDP packets in wireless environments are usually 40 to 44 bytes long, excluding IP headers [3]). Among these discarded packets, most of them consist of only single-bit errors. Since many popular RTMAs encodes/decodes (e.g. by using H.263 and MPEG-4) the data with error detection and recovery capability [4], most of the discarded packets are actually useful packets to the RTMAs. In fact, it is found that up to roughly 85% of the packets discarded by the receiving transport layer due to checksum errors could have been tolerated by the RTMAs [2]. Therefore, most of these applications will perform better if the "damaged" packets are delivered to them rather than dropped at the transport layer. One may think that, since checksum operation is optional in UDP, it could be advantageous for the UDP protocol stack to simply disable the checksum all together. However, this argument has been proved to be impractical [5][6].

A novel version of UDP, UDP-Lite [1][7], has been introduced to reduce the packet loss for UDP. The concept of UDP-Lite has been shown to be able to improve overall performance, including packet delay, inter-arrival time, packet loss, as well as perceived video/image quality, in RTMAs [4].

However, there are 2 major drawbacks for UDP-Lite – backward incompatibility and lack of flexibility to handle corrupted packets at the application layer.

This paper proposes a new protocol that makes use of the concept introduced by UDP-Lite, and yet is 100% compatible with all the legacy UDP applications. Better yet, it gives the application layer flexibility to selectively process normal and corrupted packets in different ways. We call this new protocol stack UDP-Liter.

It is noteworthy that the handling of IPv4 in UDP-Liter is different to IPv6 (the reasoning will be provided in section VIII). This paper focuses on IPv4 discussion only. We will discuss the impact of IPv6 of UDP-Liter in our future work.

II. UDP-LITE

UDP-Lite uses the same header format as traditional UDP. The only difference between them is that the "UDP Length" field in traditional UDP is replaced by the "Coverage" field in UDP-Lite. The "Coverage" field in UDP-Lite specifies the number of bytes in the packet that is sensitive to errors and therefore must be verified.

The basic concept of UDP-Lite is to selectively verify the error-sensitive portion (given by the coverage field) of the UDP datagram. All the errors occurring in the insensitive portion of the datagram (usually the payload) will be ignored and the associated payload will be passed to the application as if they are normal packets. As a result, the applications (e.g.

This work is sponsored by the Areas of Excellence scheme established under the University Grant Committee of the Hong Kong Special Administrative Region, China (Project Number AoE/E-01/99).

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 5, 2009 at 03:11 from IEEE Xplore. Restrictions apply.

video/audio decoders) will be given a chance to fix the errors in the payload. If the errors are single-bit only, most codec applications can fix them and the data will be presented flawlessly. Even though a packet has unfixable errors, it will only cause a glitch in the perceived media. However, a lost packet will cause annoying pauses or noticeable disturbances [1]. Hence, passing an erroneous packet to the application layer will be more preferable to RTMAs than discarding them. This is exactly what UDP-Lite intends to do.

III. Issues Associated with UDP-Lite

A. Backward Incompatibility with Traditional UDP

UDP-Lite has been proven to improve the performance and quality of many UDP based real-time applications over wireless links [2][4]. However, it is not difficult to spot the backward incompatibility problem it introduces to real life deployments. As UDP-Lite requires incompatible modifications to be made on the traditional UDP protocol, it can only inter-work with UDP-Lite capable applications. Particularly, changing the "UDP Length" field to the "Coverage" field causes the traditional UDP applications to incorrectly interpret the UDP-Lite header. In addition, this backward incompatibility issue also means that all the traditional UDP applications (e.g. the RTP stacks) will become useless when UDP-Lite is installed on the operating system.

This incompatibility problem is recognized by Larson et al. [7], and they have proposed three solutions. First, UDP-Lite can use a different protocol identifier than traditional UDP, so that the receiver can correctly handle them in different ways. We found this solution impractical because every UDP-Lite equipped device must then be incorporated with 2 UDP stacks. In addition, this solution still requires explicit support of UDP-Lite on both sides, and therefore it is not solving the fundamental backward incompatibility problem.

The second proposed solution is to have the sender use explicit application in-band signaling to learn the UDP-Lite awareness on the receiver side. This solution will require certain amount of UDP overhead to establish the in-band signaling hand-shaking. For example, the in-band signaling must know when to setup and tear down an UDP-Lite session, and must handle packet loss during the in-band signaling. The possible delay caused by the signaling itself may already cancel out the performance gain.

The third solution is to employ out-of-band signaling such as H.323 and SIP to convey the acceptance of UDP-Lite at the receiver. We also find this solution to be impractical because UDP-Lite must gain its popularity before the major standards will be modified to accommodate it. However, this solution relies on the application signaling to enable deployment of UDP-Lite. As a result, this reliance on application layer signaling really creates a "chicken and egg" problem.

B. Lack of Flexibility for the Application Layer to Handle Corrupted Packets

There is another drawback that has yet to be mentioned in related literature -- UDP-Lite does not provide flexibility to the application layer on how corrupted packets are handled. That is, when UDP-Lite passes a corrupted packet up to the application layer, the application layer must accept it as normal packet. This may raise application stability concern if a particular application is incapable of handling erroneous packets. For example, communication signaling protocols (e.g. Session Initiation Protocol [8]) mostly use UDP as their transport protocol. Corruption in these packets at the receiver can be disastrous as wrong signaling information will be relayed to the parties involved in a call.

IV. UDP-Liter

In this paper, we propose two minor changes to the traditional UDP protocol stack, so that it will take advantage of the concept of UDP-Lite, and also address the backward incompatibility and lack of packet handling flexibility issues. We call this new UDP protocol UDP-Liter. We choose this name to partly give credit to the inventors of UDP-Lite, and partly indicate that it is a "lighter" (i.e. less restrictive) implementation than the traditional UDP as well as UDP-Lite.

UDP-Liter employs the concept of UDP-Lite, but it does not modify the UDP header specification at all. That is, "UDP Length" still means "UDP Length" and "UDP Checksum" still means "UDP Checksum" in traditional ways. This is very important because a traditional UDP receiver may simply process the packet the way it always does without any loss of information. Specifically, UDP-Liter proposes the following 2 changes to the traditional UDP implementations:

- The UDP protocol stack continues to perform the checksum calculation, but it now has a run-time option not to discard packets when the checksum fails. When this option is enabled, it should pass the payload to the upper layer (e.g. RTP) regardless of the checksum result, and notify the upper layer if corruption occurs.
- The BSD socket API [9] needs to be slightly modified to reflect the changes made in the bullet point above. Therefore, the caller of this API may obtain the corruption notification (CN) and then determine whether it should accept the packet, or whether it should handle the packet with a different algorithm.

The following subsections describe these two modifications in detail. All other aspects that are not covered in this discussion (e.g. the pseudo header given by the IP layer) are regarded to be identically specified as in traditional UDP [10].

A. The Checksum Operation in UDP-Liter

UDP-Liter maintains the exact header format and specification of the traditional UDP. It also maintains the same checksum operation in a way that is identical to traditional UDP. The only difference is that the application is now given an option, through the call to *socket()*, to ask UDP-Liter to either retain the traditional UDP behavior, or perform the services provided by UDP-Liter. In the former case, UDP-Liter simply discards the packets that fail the checksum. In the latter, UDP-Liter will pass to the application all packets received together with the CN for each packet. The CN is carried by the parameter *pCorrupted* (to be defined later in this section). If the packet passes the checksum, the **pCorrupted* parameter should be set

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 5, 2009 at 03:11 from IEEE Xplore. Restrictions apply.

to zero (FALSE); otherwise, it should be set to one (TRUE). Thus, the application now has the knowledge of whether the receiving packet is normal or not, and therefore can handle it accordingly.

B. The BSD Socket API in UDP-Liter

In order to let applications enable UDP-Liter at their will, the BSD socket API will need to be slightly modified. Furthermore, these modifications should keep the disturbance to the traditional UDP to minimum, so that real-life deployments will be easy and practical. In this paper, we propose to modify the call to the *socket()* function, and add a new version of the *recvfrom()*, named *recvfrom_liter()*, function to the BSD socket API. The following is the detailed description of these changes.

1) The socket() call

The signature of the BSD *socket()* function is [9]:

int *socket*(int *family*, int *type*, int *protocol*);

Since only the type parameter is concerned in this discussion, we will not discuss the other 2 parameters in detail.

In UDP-Liter, there will be a new socket type constant – SOCK_DGRAM_LITER. This new socket type specifies that the socket being created by this *socket()* call intends to use services provided by UDP-Liter. As a result, the transport layer (i.e. UDP-Liter) will forward all received packets to the application layer regardless of their checksum results, together with the CN to the caller of *recvfrom_liter()*.

It is important to note that a call to *socket()* with the socket type constant set to SOCK_DGRAM indicates that traditional UDP behavior should be retained. That is, packets that fail the checksum will be discarded. This allows all the existing legacy applications to continue to function normally on top of UDP-Liter without any code change.

2) The recvfrom_liter() call

In order to enable the CN functionality, we propose to add a new function call to the API, instead of modifying the original one. Thus, legacy applications will continue to run normally.

The proposed additional function call is:

ssize_t *recvfrom_liter*(int *sockfd*, void **buff*, size_t *nbytes*, int *flags*, struct sockaddr **from*, socket_t **addrlen*, **int *pCorrupted**).

The function *recvfrom_liter()* is a counterpart to the *recvfrom()* function in the traditional socket API. The only difference between them is the extra *pCorrupted* parameter in *recvfrom_liter()*. This additional parameter is a pointer to integer which, when the function returns, contains the boolean value of **pCorrupted* (i.e. TRUE/FALSE, or 1/0 in C) generated by the checksum to indicate the corruption occurrence. The default value of **pCorrupted* is zero (FALSE), meaning that there is no corruption in the packet.

With the information provided by **pCorrupted*, the applications can differentiate corrupted packets from normal ones, and may handle them differently or with different algorithms at run-time.

Please note that the link layer modification suggested by [1] (e.g. disabling the checksum or CRC in the link layer protocols) will also need to be taken place in UDP-Liter. This will prevent the link layer protocols from discarding the damaged packets before they reach the transport layer.

V. BENEFITS OFFERED BY UDP-LITER

A. 100% Backward Compatibility with Traditional UDP

We have emphasized repeatedly that UDP-Liter is 100% backward compatible with traditional UDP. In this section we will illustrate that this claim holds for all scenarios.

1) The sender uses traditional UDP, and the receiver uses UDP-Liter.

In this case, all the incoming packets at the UDP-Liter receiver are delivered to the applications regardless of the checksum results. If the packet passes the checksum, *recvfrom_liter()* will be returned to the application with **pCorrupted* set to 0 (FALSE). Otherwise, *recvfrom_liter()* will be returned to the application with **pCorrupted* changed to 1 (TRUE). In either case, the sender does not even notice that the receiver is not a traditional UDP server.

2) The sender uses UDP-Liter, and the receiver uses traditional UDP.

As a sender, UDP-Liter operates identically to traditional UDP. Since the UDP-Liter packet is also identical to a traditional UDP packet, the traditional UDP receiver will process it normally. Of course, when this packet fails the checksum at the UDP receiver, it will be dropped.

3) A legacy UDP application is installed on an operating system with UDP-Liter installed.

With UDP-Liter, the legacy applications simply run normally as with traditional UDP. Since all the traditional socket API are still in place, including the traditional **recvfrom()* function, all the socket calls will function just normally. Also, legacy applications will have setup the socket with socket type SOCK_DGRAM, therefore the UDP-Liter will just run as traditional UDP.

B. Application Flexibility Enhanced by Corruption Notification (CN)

In UDP-Lite, applications are "forced" to accept all packets delivered from the transport layer as normal packets, regardless the checksum results. This may cause various problems mentioned in section III.B.

With UDP-Liter, if applications intend to use its services, they must be coded particularly to make use of the UDP-Liter enabled socket API. Consequently, each packet delivered from the UDP-Liter will be associated with a CN through the parameter **pCorrupted*. With this CN, applications can then decide what to do with the corrupted packets.

For example, suppose a video codec can decode corrupted packets with a time consuming error-correction algorithm. With UDP-Liter, this algorithm only has to be used when a corrupted packet is received. This should improve the image rendering performance because majority of the packets (normal

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 5, 2009 at 03:11 from IEEE Xplore. Restrictions apply.

packets) can be presented to the user using a faster algorithm, while the corrupted packets can fill the gap as soon as the slower algorithm finishes processing.

Last but not least, the CN feature allows an application to determine "how bad" the corruption situation is in real-time. Thus, the application can ask the sender to slow down, or use another channel, when a particular channel receives excessive number of corrupted packets.

VI. SHORTFALL OF UDP-LITER

A potential drawback of UDP-Liter is the incapability to determine precisely, when a packet is corrupted, where the bit error occurs in the packet. Before we proceed further to analyze the impact, recall that the checksum operation in traditional UDP, as well as UDP-Liter, not only validates the UDP header, it also validates the "pseudo-header", which includes a 20-byte information provided by the IP header [5]. If the bit error so happens to occur in the pseudo- and UDP headers (referred as *header* hereinafter) instead of the payload, UDP-Liter still forwards this packet to the application layer just as any other corrupted packets.

These header errors, however, do not affect traditional UDP applications running on UDP-Liter at all (because the traditional UDP checksum will be enabled for these applications). Therefore we will only focus on its impact toward UDP-Liter applications in the following analysis. In addition, we should point out that errors in some header fields are not harmful in any way. For example, if the *Protocol* field is in error, the packet is not even in the UDP format, and UDP-Liter will simply ignore it. Errors in the *UDP Length* and *Checksum* fields do not prevent UDP-Liter applications from processing normally. Finally, errors in the *Source* and *Destination Address* fields should have been caught by the IPv4 checksum and therefore already dropped at the IP layer.

Although errors in the *Source port* field could mislead the receiver about the sender's identity, this field is rarely used by RTMAs. Its impact is therefore considered harmless here.

Therefore, the *Destination port* field is the only header field that could cause glitches. That is, if the destination port is in error, the IP packet could be directed to an incorrect application which has bound to the port that happens to be identical to the erroneous destination port number. In other words, the application could have received a packet it is not supposed to receive.

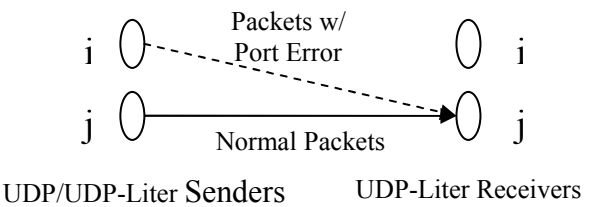

Figure 1. Packets mis-routed due to destination port error

This problem is illustrated in Figure 1. We assume the following notation in this paper.

P_e = probability of bit error = BER for the channel

P_{dport} = probability of an error in the destination port field

$P_{i,j}$ = probability for a packet from *application i* to be mis-routed to the receiver of *application j* due to destination port error

S_i = packet rate of *application i*

R_i = aggregated rate for receiver of *application j* to receive mis-routed packets from other UDP-Liter applications.

We would like to ensure that the ratio R_i/S_i is small enough for RTMAs to ignore. Assuming bit errors are random and identical for all applications, we have (for 16-bit destination port field)

$$P_{dport} = 1-(1-P_e)^{16} . \quad (1)$$

There is a total of 2^{16} destination port numbers in UDP. The probability for a particular UDP-Liter application to receive a packet with the aforementioned port-number error is therefore 2^{-16}. Thus, the probability for application *j* to receive a packet targeted to application *i* (i.e. mis-routed) is

$$P_{i,j} = P_{dport} / 2^{16} . \quad (2)$$

Suppose there are k (k>0) UDP-Liter RTMAs running simultaneously. A particular application could be receiving packets with corrupted destination port number from the other $(k-1)$ sources. Although there may be misrouted packets from sources of other traditional UDP applications running on the same device, their packet rates are likely to be much lower than those of UDP-Liter enabled RTMAs. Therefore, from the view point of the ratio R_i/S_i, they can be ignored. Considering the potential misrouted packets from all other UDP-Liter applications, we have

$$R_j = \sum_{i\neq j}^{k} \left(S_i * P_{i,j}\right) = \left(P_{dport}/2^{16}\right) * \sum_{i\neq j}^{k} S_i \quad (3)$$

On a wireless device, k is usually no more than 2 (e.g. one for voice and one for video in a video-conferencing session). Assuming $P_e = 10^{-3}$, $P_{i,j}$ is approximately 2^{-22}. Therefore, assuming S_i is 50 for voice [11], and 67 for video (this reflects the worst case scenario for a video conferencing application described in [12]), R_i/S_i for voice is $(117/50)* 2^{-22} = 5.58*10^{-7}$, while R_i/S_i for video is $(117/67)* 2^{-22} = 4.16*10^{-7}$. Therefore, there is about 1 mis-routed packet in each channel (voice and video) for every 10 hrs of video conferencing session. It is also noteworthy that, when k = 1, R_i/S_i = 0. This indicates the situation when there is only one UDP-Liter

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 5, 2009 at 03:11 from IEEE Xplore. Restrictions apply.

application running on a wireless device (e.g. voice call), there will be no noticeable mis-routing issue.

At the other extreme, let us consider how UDP-Liter performs in an enterprise scale and fully occupied voice gateway for wireless VoIP deployment (assuming 32 channels). In this case, P_e is still 10^{-3}, k is 32 and S_i is 50. Then, R_i/S_i is $31*2^{-22}$. This corresponds to about 1 mis-routed packet in about 45 minutes of operation in each voice channel. To further reduce the impact of mis-routing, the RTMAs can easily detect mis-routed packets by ensuring the sequence numbers in the RTP headers [11] between consecutive packets do not differ by an excessive margin.

According to [13], 1 to 3% of media degradation is acceptable in RTMAs. Hence, we are reasonably convinced that the impact of header errors in UDP-Liter is insignificant.

VII. Performance of UDP-Liter

A. Packet Loss Rate

Since UDP-Liter does not discard packets at all, the packet loss rate is much less then traditional UDP. On the other hand, the only difference between UDP-Liter and UDP-Lite is that UDP-Lite requires to verify the headers. According to our analysis in section VI, the header errors are actually insignificant to RTMAs. Consequently, UDP-Liter is delivering more useful packets to the application layer than UDP-Lite. Therefore, we are convinced that the performance of UDP-Liter is a little better than UDP-Lite for RTMAs.

B. Computational Complexity

In terms of computational complexity, UDP-Liter is identical to traditional UDP because both of them need to compute the checksum for the entire packet. On the other hand, due to the smaller average checksum computation, UDP-Lite does have a lower computational complexity (O(1)). However, we notice that the checksum operation is basically the one's complement sum of 16-bit integers within the packet [14], which has a computational complexity of O(N). Nonetheless, we think that the extra complexity is worthwhile given the advantages of UDP-Liter described in Section V.

VIII. Future Work

Our analysis so far is based on an IPv4. With IPv6, the analysis cannot rely on the IP checksum for source and destination addresses anymore because the checksum is removed from IPv6 [15]. In our future work, we will investigate whether we can make use of the IPv6 options to further improve UDP-Liter.

Moreover, the link layer driver modification is also an area worth for future studies. For example, the impact of this modification (in favor of UDP traffic) toward TCP traffic may be negative, because the corrupted TCP packets must now be processed at the TCP layer (recalling that the link layer will not verify the payload). This may worsen TCP's performance.

IX. Conclusions

This paper has introduced an improved UDP protocol -- UDP-Liter. The proposal is based on the novel concept introduced by UDP-Lite -- damaged packets are preferred over no packet for RTMAs. We have identified a few major fundamental problems in UDP-Lite, namely backward incompatibility to traditional UDP protocol, and lack of flexibility for applications on handling corrupted packets. In this paper we have proposed minor modifications to the traditional UDP and BSD socket API, so that we can apply the concept proposed by UDP-Lite, yet maintain 100% backward compatibility and allow applications to handle corrupted packets flexibly at run time. Furthermore, we investigated the impact of performance in terms of packet loss rate and computational complexity introduced by UDP-Liter. We concluded that the improvement in performance and other benefits stated above should justify the higher computational complexity (by O(N)) of UDP-Liter over UDP-Lite. Finally, we have also identified the drawback for UDP-Liter, which is the inability of UDP-Liter to differentiate header errors from payload errors. We have carefully analyzed the impact of this drawback, and have concluded that the errors in the header are still tolerable by the RTMAs. We have also suggested other future research areas for further improvement to be achieved on RTMAs.

References

[1] L. Larzon, M. Degermark, and S. Pink, "UDP Lite for Real Time Multimedia Applications," HP Labs, Bristol, United Kingdom, Tech. Rep. HPL-IRI-1999-001, Apr. 1999.

[2] L. Larzon, M. Degermark, S. Pink, "Efficient Use of Wireless Bandwidth for Multimedia Applications", in Proceedings of International Workshop on Mobile Multimedia Communications, 1999.

[3] R. Lloyd-Evans, "QoS in Integrated 3G Networks," Artech House, 2002.

[4] A. Singh, A. Konrad, and A. D. Joseph, "Performance Evaluation of UDP Lite for Cellular Video," Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), June 2001.

[5] W. R. Stevens, TCP/IP Illustrated, Volume 1, Addison-Wesley, 1994.

[6] J. Stone and C. Partridge, "When The CRC and TCP Checksum Disagree," .ACM SIGCOMM, September 2000

[7] L. Larzon, M. Degermark, and S. Pink, "The UDP Lite Protocol," Internet-Draft (work in progress) draftlarzon -udplite-02.txt, Lule University of Technology, July 2000.

[8] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: session initiation protocol," RFC 3261, IETF, June 2002.

[9] W. R. Stevens. "UNIX Network Programming," Prentice Hall vol. 1, 2rid edition, 1998.

[10] J. Postel, "User Datagram Protocol," RFC 768, Internet Engineering Task Force, August 1980.

[11] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 1889, IETF, January 1996.

[12] J.C Bolot, "Characterizing End-to-End Packet Delay and Loss Behavior in the Internet.," In Proc. ACM SIGCOMM, Sept 1993.

[13] Toni Janevski, "Traffic Analysis and Design of Wireless IP Networks," Artech House, 2003.

[14] B. Braden, D. Borman, and C. Partridge, "Computing the Internet Checksum," RFC 1071, IETF, September 1988.

[15] S. Hagen, IPv6 Essentials, O'Reilly & Associates, Inc., 2002

Authorized licensed use limited to: Chinese University of Hong Kong. Downloaded on February 5, 2009 at 03:11 from IEEE Xplore. Restrictions apply.