J. Basic. Appl. Sci. Res., 3(1)830-836, 2013

ISSN 2090-4304
Journal of Basic and Applied Scientific Research
www.textroad.com

An Alternate Encapsulation and Implementation of DCCP for Multimedia Applications

Jawad Shafi, Rab Nawaz Jadoon, Usman Ashraf, Syed Asad Hussain, Munib Ahmad and Touseef Tahir

Department of Computer Science
COMSATS Network Research Center (CNRC)
COMSATS Institute of Information Technology Lahore, Pakistan

ABSTRACT

Datagram Congestion Control Protocol (DCCP) is a transport layer protocol which provides bidirectional unicast connection for congestion-controlled unreliable datagrams. It is suitable for applications that require low end to end delay and small degree of data loss. In this paper we have implemented a substitute encapsulation technique for DCCP using UDP-Lite with Customized Generic Header (DCCP-UDPLite-CGH). This encapsulation allows DCCP to be carried through the current generation of network address translation (NAT) devices without any modification and updation. We have also implemented DCCP protocol stack optimized for portability so that it can be run on Linux as well as on Windows platform. The proposed framework is implemented and compared against well known protocols including SCTP and UDP. Implementation results show that the proposed framework outperforms SCTP and UDP in terms of packet loss and network throughput.
KEYWORDS: DCCP, CCID2, Network Address Translation, UDP-Lite, IP, customized generic header etc.

I. INTRODUCTION

Internet today connects the computer world wide. Computers communicate with each other using predefined rules of communications, so called protocols [1]. These protocols are divided into different layers which provide a specific functionality for each layer.

Transport protocols are used to deliver the data in different computer networks to the different processes running in the same or different machines across the networks. The internet today has two mostly deployed transport layer protocols TCP and UDP [2].

TCP used for the reliable and connection oriented approach for data transmission, it means that no data lost in TCP. TCP achieved this by creating a connection between sender and receiver and the packet received at the receiver transmit acknowledge back to the sender. In this way a packet sent but not received can be retransmitted at the sender side. TCP also have congestion control mechanism which reacts when the congestion occurs in the network. When packets being lost in the network, TCP decreases its sending rate to minimize the risk of packets being lost in the network due to congestion.

UDP cannot create any connection prior to the transmission of data between sender and receiver. UDP is mainly used for short, request-response transfer, like DNS and SNMP that wish to avoid TCP's three-way handshake, retransmission, and/or stateful connection [1]. The sender sends data without knowing that whether the data is delivered at the receiver or not. This prevent UDP from knowing about any congestion exists in the network. However, since UDP traffic volume is small relative to congestion-controlled TCP flows, the network didn't collapse.

The next flavor of the UDP was developed with the name UDP-Lite [1, 2, and 3]. UDP-Lite is a lightweight protocol with increased flexibility in the form of a partial checksum implementation. It gives the opportunity to the senders to specify packets as partially insensitive to errors. The coverage of the checksum is specified by the sending application on a per-packet basis. UDP-Lite protocol is easily integrated into an existing UDP because of its close relationship to it.

Network capabilities in the world wide are increasing with every coming day, rapid growth of the internet users, high speed internet connections and applications, including streaming audio, internet telephony, and multiplayer and massively multiplayer on-line games, share a preference for timeliness over reliability [1]. These applications required some mechanism which starve other flows in the network and back-off in case of congestion between the paths from sender to receiver, so that the data-flows get a fair amount of available bandwidth.

TCP can introduce arbitrary delay because of its reliability and in-order delivery requirements; thus an applications use UDP instead. This growth of long lived non-congestion controlled traffic, relative to congestion-controlled traffic, poses a real threat to the overall health of the Internet [5, 6].

In [10], the performance of VoIP using DCCP has been evaluated and shows that the performance is significantly degraded by the size of initial slow start threshold over large delay links. So there is a clear need of a new transport

***Corresponding Author:** Rab Nawaz Jadoon, Department of Computer Science, COMSATS Network Research Center (CNRC), COMSATS Institute of Information Technology Lahore, Pakistan. Email: rabnawaz@ciitlahore.edu.pk

protocol which combines both unreliable datagram delivery and congestion control mechanism to fulfill the needs of real time applications.

The difference between the proposed approached and the old methods is shown in the following table 1,

Table 1: Comparison between SCTP, TCP, UDP, DCCP and Proposed Technique (DCCP-UDPLite-CGH)

Transport protocol	Type	Congestion basement	RTT Fairness	Packet Loss Rate
TCP	Standard TCP	Loss based	Quite Bad, only keeps fairness when RTT is short	Good due to the low utilization ratio, normally depend on the condition of link
UDP	Standard TCP	No Congestion Control	Do not have Ack, so independent of RTT	Bad, but it is not concerned for UDP
DCCP	Standard UDP	Loss based	Same as TCP, depends heavily on RTT length	Bad, but packet loss is acceptable for DCCP
SCTP	High Speed TCP Variant	Loss based	Becomes better when RTT increase	Bad, quite high loss rate
DCCP-UDPLite-CGH (Proposed Technique)	Standard DCCP and UDP Lite Flavor	Delivery Based	Independent of RTT	Negligible

The scientific contribution of this paper is that we have proposed a technique which solves the problem of DCCP to allow traffic through the current generation of network address translation (NAT) middle boxes/devices without any modification and updation. For this purpose we have proposed DCCP-UDPLite-CGH (Datagram congestion control protocol using UDP lite with Customized Generic Header), which solved the above mentioned problem by introducing a middleware between UDP-Lite and DCCP. After simulation we have seen the significant improvement in the proposed technique.

The rest of the paper is organized as follows. Section II describes the problem statement. Section III presents the proposed framework and section IV state the implementation detail and result discussion respectively. Finally section V concludes the paper.

II. PROBLEM STATEMENT

The use of extensive multimedia application over the internet and mobile communications, some of these applications requires a certain minimum throughput to operate, while others require a minimum latency. Some application does not tolerate dropped packets while others contain time-sensitive information whose delivery is better cancel than delay.

When huge amount of data is transmitted over the same link then there is an excessive accumulation in quality of service QoS deteriorates, result in delay, packet loss or blocking of the new connection. This leads to use transport protocol which minimizes the congestion and over comes with the time-sensitive information and take some precautionary measurements if congestion occurs in networks. Secondly if we want to test the DCCP over the internet then the NAT devices must be configure or updated accordingly to provide end to end communication using DCCP transport layer protocol. This is long term objective and many efforts are under consideration to overcome the problem stated. In this paper we have also find a solution against the problem stated. The detail of implementation is in section III.

III. PROPOSED FRAMEWORK

When the packet moves from the upper layer to the lower layer in TCP/IP stack, it must be encapsulated in IPv4 packets. When packet moves in the network it has to pass through many network address translation middle boxes, so all these devices must be updated or modified with the DCCP so that it can be distinguished and changed accordingly. But its long-term objective to modify the DCCP behavior by introducing a thin layer of UDP Lite (as UDP-Lite can serve in error-prone network environments that prefer to have partially damaged payloads delivered rather than discarded[13]) between the DCCP and IP layer (modification eliminates the redundancies between the UDP Lite and DCCP header). Note that this is strictly tunneling approach, DCCP packets with a customized generic header so that the redundancies between UDP Lite and DCCP packet can be eliminated. IP addresses of the ending party will be carried in the IP header which can be modified by the NAT devices by attaching the IP Address to the header. The IP address which was carried in the IP header now can be modified by NAT end devices and need not to carry any other IP address, so there is no need to attach any other IP address with the header. In this approach we have worked with only IPv4.

Devices offering or using DCCP services through DCCP customized generic header encapsulation are listened on the UDP Lite port. In our implementation we have used 2023 port for incoming packets and then it will passes the packet along with the DCCP customized generic header to the application which is using the DCCP as transport layer protocol. After that the DCCP provides all the same basic operation as mentioned in [2].

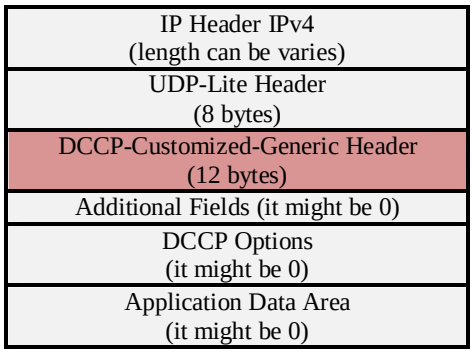

Fig. 1 Proposed framework stack

The purpose of adding the UDP-Lite header is that the devices which are providing the DCCP services must forward the packets to the UDP-Lite port which is part of the UDP-Lite header [3] and then passes the packets to the application which has providing the DCCP services along with the port number of DCCP, defined in the following header,

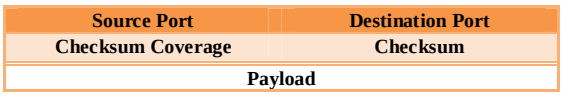

Fig. 2 UDP lite header

3.1 Source and Destination Ports: 2 byte each

These identify the UDP-Lite ports on which the source and destination application are listening for the UDP-Lite packets and then transmit the header along with DCCP customized generic header. One point of important is that this source and destination port does not identify the DCCP customized generic header Ports.

3.2 Checksum Coverage: 2 byte 0 or 8+

This field consists of number of octets in UDP-Lite starting from most significant bits and check each single octet that are covered in the checksum field. If there is some error in octet then it can detect it after mapping to the particular octet. So it will deliver the damage data payload to DCCP application instead of dropping it to the receiving end. So DCCP header can be passed even if it has errors. The value of this field must be 0 or at least 8. And if it is within the range (0-7) then the packet must be discarded by the receiving entity.

3.3 Checksum: 2 bytes

This field consists of the pseudo header of the entire checksum of network layer and also the entire UDP-Lite header of the packet. If the received packet has checksum 0 then it must be consider as incorrect checksum. For DCCP customized generic header if the computed checksum is 0 then it must be transmitted as all 1's. Before computation the checksum field must be set to 0.

3.4 DCCP Customized Generic Header:

Unlikely DCCP Generic header layout, the DCCP customized generic header supports only long sequence number; its format is as under,

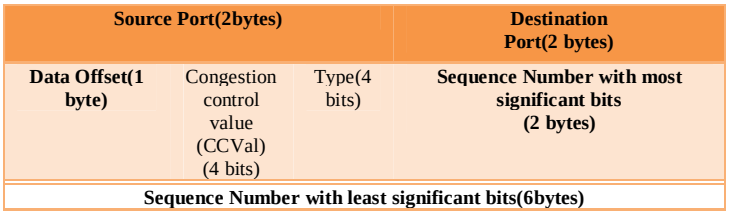

Fig. 3 DCCP customized generic header

All function of DCCP customized generic headers are same as defined in [2].

3.4.1 DCCP Customized Generic Header (CsCov and Checksum field):

The function of Checksum in the DCCP generic header field is performed by the UDP-Lite header checksum field.

If UDP-Lite received a packet with the checksum value ZERO or the checksum value is invalid then the packet must be ignored according to specifications in [2]. If the UDP-Lite header in received packet is less than the length of the

actual UDP-Lite header plus the whole DCCP customized generic header or the UDP-Lite length field is greater than the length of the packet from the beginning of the actual UDP header to the end of the packet, then the packet must be dropped due to invalid checksum.

The sender must implement and continue to update the CCVal window counter as specified, even when the send RTT estimate feature is on as mentioned in [11].

3.4.2 Service Codes for DCCP and its Port Registry:

DCCP specifies a Service Code as a 4-byte value that describes the application-level service to which a client application wants to connect [8]. If the application developer needs the new version that provides momentous clean capabilities (e.g. if it will change the performance of middleboxes) then it is up to the middleboxes to allocate any new service code interconnected with the same or different well known ports of the DCCP which is mentioned in [2]. A port registration is applicable to all the combination of encapsulation and IPv4.

As port registry allows multiple applications to register on same port as long as the service code are different but those registrations are also valid for all the combination of IPv4 [9].

3.4.3 Congestion Control Identifier 2:

CCID 2 is best for applications which would like to take advantage of the available bandwidth especially in such an environment where the abrupt changes in the congestion window can be adapted. So the applications which intends to send that data as much as possible, will use the CCID 2. It is recommended also for the streaming applications at application receiver. Also useable for application where halving the sending rate when congestion occurs in the networks so it would not affect on the application-level performance. All function which is defined in CCID2 will remain same as in [4].

3.4.4 Congestion Control Identifier 3:

CCID 3 is also called TCP-Friendly Rate Control (TFRC), in the Datagram Congestion Control Protocol (DCCP). It is used by senders that want a TCP-friendly sending rate, possibly with Explicit Congestion Notification (ECN), while minimizing the sudden rate changes [7].

3.5 Behavioral Description

Client is an entity which always initiates the connection request and the server is always in listen state. In DCCP-UDPLite-CGH, each connection has one server and one corresponding client.

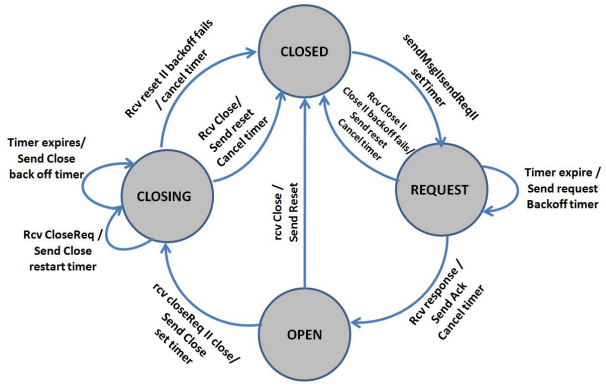

Fig. 5 Client actions

All the endpoint starts in the CLOSED state. When an application issues the call to Listen () then the endpoint called application becomes Server by changing its state to LISTEN. Connection initiation is performed by the client in the fig. 5, when the application call the Connect ( ) then the end point becomes client and change the state to REQUEST and then send a packet DCCP-Request to server.

Fig. 6 depicts the server reaction upon reception of the client packet. Once server receives the packet it enters in the RESPOND state and acknowledges the state by sending the packet DCCP-UDPLite-CGH_respond. Then the client sends the packet DCCP-UDPLite-CGH_Ack to complete the three way handshake and change the state OPEN. The server received the acknowledgment packet and then changes the state OPEN as well. Feature negotiation is performed on the handshake. After both end points are in OPEN state the server and client will ignore packets with DCCP-UDPLite-CGH_Request and DCCP-UDPLite-CGH_response.

During the OPEN state the client and server will exchange the DCCP-UDPLite-CGH_Ack, DCCP-UDPLite-CGH_Data and DCCP-UDPLite-CGH_DataAck packets as governed by the congestion control identifier 2[4].

There are two different connections for tear down procedure to initiates the close operation. If server wants to close the connection it sends the DCCP-UDPLite-CGH_CloseReq and enters CLOSEREQ state. It waits unit client close the

connection. In both cases the connection is closed when the final DCCP-UDPLite-CGH_Reset arrived at receiver. If it is in CLOSING state it transit to the TIMEWAIT state after receiving the DCCP-UDPLite-CGH_Reset and wait for 2xMSL. When this timer is expired then it transit to CLOSE state. If client want to close the connection, it sends the reset packet then it will enters in the CLOSE state and sends a DCCP-UDPLite-CGH_Closepacket. Servers send the packet DCCP-UDPLite-CGH_Reset and then enter in the LISTEN state again.

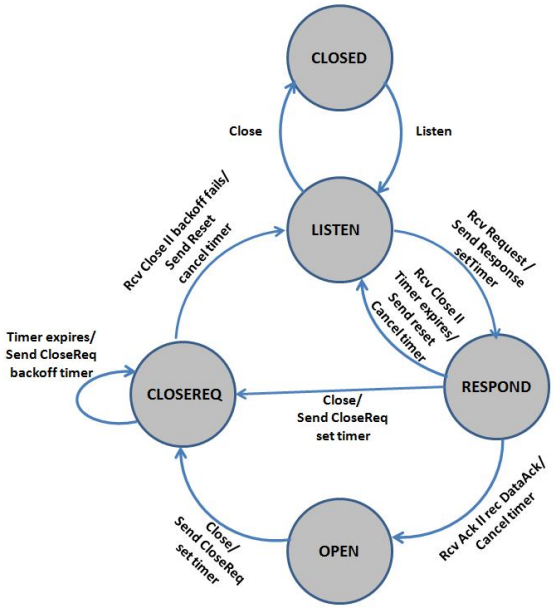

Fig. 6 Server Reaction

DCCP-UDPLite-CGH_Reset can be send by both server or client to abort a connection in case of any error occurs. The DCCP stack will reset the connection if any feature negotiation fails during the handshake. Also if the client request some mandatory options and that options are not supported by the ending point then it also transfer the DCCP-UDPLite-CGH_Reset packet. Upon receiving of the DCCP-UDPLite-CGH_Reset request in any state, the server will move to LISTEN and client moved to CLOSED. DCCP uses a retransmit timer in case when the ending points are in connection initiation or teardown.

IV. IMPLEMENTATION DETAIL

Following parameters are used to evaluate the performance of DCCP-UDPLite-CGH: Packet Loss, Throughput and Rate factor with implementation setup mentioned in Table 2.

Rate factor is a parameter used to increase or decrease the flow of control. It actually directly affects the sending rate of the sending node. The increase or decrease in the rate factor actually affects the network throughput.

To implement the DCCP-UDPlite-CGH, following hardware were used,

Table 2: Implementation Setup

Hardware/Software	Detail
Client Computer	Platform: Ubunto 10.04, Window Server 2003 CPU: 2.6 GHz RAM: 2GB IDE: Microsoft Visual Studio 2003, C
Routers	CISCO 2801Routers
Sever	Ericson Multimedia Server
Link capacity	1~10 Mbps
Packet size	100 bytes

4.1 Process of receiving the packet

The process consists of the following steps:

1. Device receives packets and generates the Interrupts.
2. Device Driver adds the packet to the global receiving queue and raises software interrupts.
3. Software interrupt handler calls the appropriate network-protocol handler for the packet received according to the network layer and based on the IPv4.
4. IP makes some check, including reorganization of the transport layer protocol and calls the higher level protocol id.

5. And then the Transport layer protocol transfers the data to the socket.

4.2 Process of sending the packet

The process is different depending on the transport layer protocol in use, but typical process consists of the following steps:

1. User application create the socket
2. then the user call the methods connect(), send() and write(), which performs the actions on the transport layer
3. Output IP routines add the packet to the sent packet ready queue, and then the particular driver will be invoked for sending it further.

4.3 RESULT DISCUSSION

Figure 7 shows that due to increase in the rate factor, the proposed protocol throughput increases because in DCCP-UDPLite-CGH packet loss is zero and the congestion control mechanism of DCCP, as the main purpose of the DCCP is to control the congestion is the network. So there is no packet loss and hence DCCP-UDPLite-CGH outperforms both SCTP and UDPLite in terms of networks throughput.

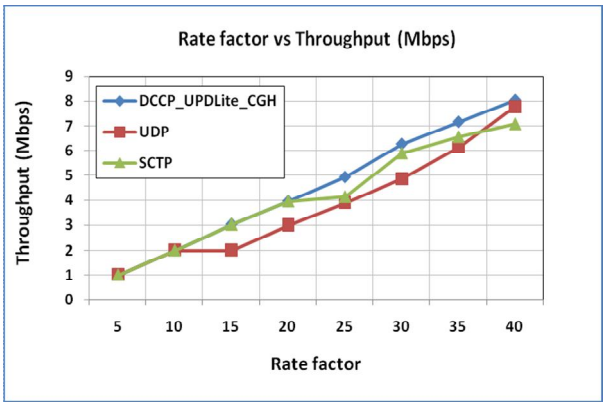

Fig. 7 Rate factor VS Network throughput

In figure 8, the packet loss is in DCCP-UDPLite-CGH is zero because of its efficient congestion control mechanism. In case of SCTP and UDP there is a large number of packet loss due to no congestion control mechanism, while in case of SCTP there is a minor packet loss. The DCCP-UDPLite-CGH outperforms both UDP and SCTP in terms of packet loss.

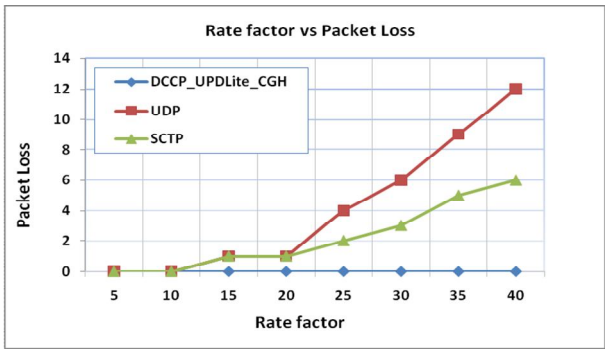

Fig. 8 Rate factor VS Packet Loss

V. CONCLUSION

A novel transport layer protocol DCCP-UDPLite-CGH has been proposed. The proposed protocol inserted a layer between the IP header and DCCP header so it can carry. After implementation it has been observed that DCCP customized generic header (DCCP-UDPLite-CGH) can perform communication using this encapsulation, which allows DCCP to be carried through the current generation of network address translation (NAT) middle boxes without any modification and updation in the network.

The implementation results show that the proposed scheme outperforms both UDP and SCTP in terms of packet loss and network throughput.

VI. ACKNOWLEDGEMENT

This work is partially supported by Ericsson Research Corporate Unit, Luleå Sweden.

REFERENCES

[1] S. Floyd, M.Handley, E.Kholer, "Problem statement for Datagram Congestion Control Protocol", RFC 4336, March 2006.

[2] S. Floyd, M.Handley, E.Kholer, "datagram congestion control protocol", RFC 4340, March 2006.

[3] L-A. Larzon, M. Degermark, S. Pink, "The UDP-Lite Protocol", RFC3828, August 2003

[4] S. Floyd, E.Kholer,"profile for Datagram Congestion Control Protocol Congestion Control ID 2: TCP-like Congestion Control",RFC 4341,March 2006.

[5] S. Floyd, "Congestion Control Principles", RFC 2914, Sep 2000.

[6] S. Floyd, J. Kempf, "Congestion Control for Voice Traffic in the Internet", RFC 3714, march 2004.

[7] S. Floyd, E.Kholer, J.Padhye," DCCP CCID3: TCP-Friendly Rate Control",RFC 4342,March 2006.

[8] W.Simpson," IP in IP Tunneling", RFC 1853, October 1995.

[9] G. Fairhurst, "The datagram congestion control protocol (DCCP) Service codes",RFC 5595,sep 2009

[10] G. Fairhurst, " Datagram Congestion Control Protocol (DCCP): Simultaneous-Open Technique to Facilitate NAT/Middlebox Traversal",RFC 5596,sep 2010

[11] Shahrudin Awang Nor, Suhaidi Hassan SMIEEE,Omar Almomani,"Simulated Performance of VoIP using DCCP CCID2 over Large Delay Link Networks", international conference on applications Protocols and services, Nov 2008, Malaya.

[12] G. Renker, G. Fairhurst, "Sender RTT Estimate option for DCCP", Internet-Draft, Apirl 2011.

[13] G. Renker, G. Fairhurst, "MIB for the UDP-Lite protocol", RFC 5097, Jan 2008.