ISSN 2350-1022

International Journal of Recent Research in Mathematics Computer Science and Information Technology
Vol. 3, Issue 1, pp: (39-42), Month: April 2016 – September 2016, Available at: www.paperpublications.org

Command Transfer Protocol (CTP) for Distributed or Parallel Computation

John[1], Rajiv Sharma[2]

[1]M.Tech Scholar, Baba Mastnath University, Asthal Bohar, Rohtak, Haryana-124001
[2]Assistant Professor, CSE Deptt. ,Baba Mastnath University, Asthal Bohar, Rohtak, Haryana-124001

Abstract: **In this paper, an improved version of a new networking protocol CTP for distributed or parallel computations is presented. In common, it is suitable just for fast, reliable and feature full interchange of small messages. CTP is a transport level API which helps in incrementing the speed of interchange. CTP is designed to allow general configurability, enabling its use in a wide range of general purpose and specialized applications. CTP covers a number of layers, from transport layer to application layer, proves that the area of its responsibility starts from relatively low level and goes to a high one.**

Keywords: **CPT (Command Transfer Protocol), SmartBuffers, Command, Confirmations.**

I. INTRODUCTION

Existing network transport protocols such as TCP [1] and UDP [2] have limitations when they are used in new application domains and for new network technologies. For example, multimedia applications sharing a network need congestion control but not necessarily ordered reliable delivery, a combination implemented by neither TCP nor UDP. TCP does not support broadcasting.UDP does, but it is not reliable and the size of UDP datagram is limited by 65467bytes[3].Similarly, the congestion control mechanisms in TCP work well in wired networks but often overreact in wireless networks where packets can be lost due to factors other than congestion. The lack of appropriate guarantees or specific features has led to the widespread development of specialized protocols used in conjunction with or instead of standard transport protocols. The first step in developing a customizable transport protocol is identifying various quality attributes and algorithms. While the list of possible quality attributes is large [4], the ones we address here can be divided roughly into the following categories:

- **Reliability**: Addresses the likelihood that the receiver receives all the data sent by the sender. In addition to accuracy of delivery, network reliability is measured by frequency of failure, the time it takes a link to recover from a failure, and the network's robustness in a catastrophe.
- **Ordering**: Describes guarantees concerning the ordering of data at the receiver relative to the order in which it is sent.
- **Performance**: Performance can be measured in many ways, including transit time and response time .Transit time is the amount of time required for a message to travel from one device to another. Response time is the elapsed time between an inquiry and a response.
- **Timeliness**: Describe the timing characteristics: when the data should be sent and how fast they can be sent.
- **Security**: Addresses confidentiality, integrity, authenticity, and data replay. The strength of the guarantee for each of these attributes depends on the types of attacks to be tolerated.

In general, the chosen quality attributes apply to every message within a session, but it may also be useful to allow individual messages to be given particular attributes. In this paper, we describe our experience building a Command Transport Protocol, CTP, achieves comparable performance to existing protocols such as TCP and UDP on the applications for which they were designed. CTP is a protocol that is to satisfy needs of arbitrary tasks, which, need support of rapid messages interchange and which, can start heavy computations as a reaction for message receipt.

II. COMMAND TRANSFER PROTOCOL (CTP)

CTP is a specialized transport protocol and runs on top of UDP/IP. It meant as a replacement for both TCP and for MPI, PVM, and other high performance computing (HPC) APIs. The majority of existing toolkits for parallel computations use, the so named "message", as a basic abstraction. The basic abstraction used in CTP is "command". Command is an order from somebody to someone to do something (in most cases, workstations in clusters are communicating exactly in this way) or the response for such an order. CTP protocol is used for faster message transfer in heavy computation.

CTP provides following functionality to improve message transmission:

- **DelAfterError** :- if set, then information about sending this packet will be removed for sent packets storage after the receiver has been informed that its arrival was not confirmed (after error description has been generated).So, this error will be delivered only once, and if data has not arrived, it will be lost.
- **NoResend:-** if set, this packet will not be resent even if it was not confirmed.
- **UniqueCommand :-** if set, then confirmation of this packet's command will confirm all packets with the same command's number that was sent to the given recipient. It is not allowed to use this option for large messages to protect from integrity corruption.
- **Broadcast:** - if set, then this packet is to be broadcasted. Option itself does not have any influence on the networking. As usual, for broadcasting, the user has to specify the recipient's IP address, like: 255.255.255.255. But this option provides two important things which are necessary for broadcasting with CTP. First of all, sent message has to get its ID and has to be stored in session information storage in entry, which corresponds to "broadcasting". Secondly, this message will get the ability to be confirmed by an arbitrary workstation, because it is impossible to know beforehand, which will get it. The command is considered to be confirmed if at least one workstation confirms it. It is not recommended (but possible) to use it for large commands.

III. CTP'S IMPLEMENTATION

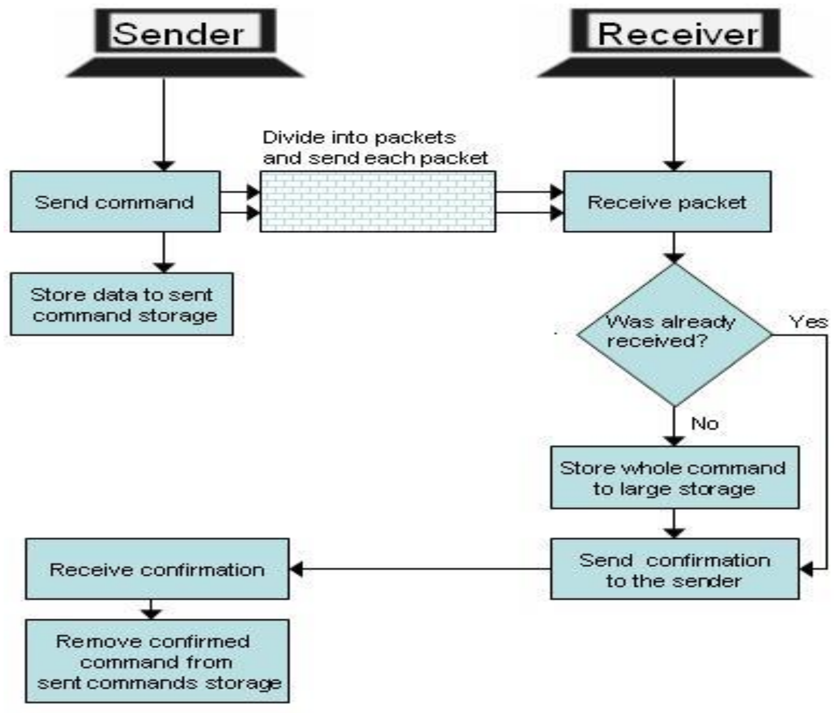
Sender
Receiver
Divide into packets and send each packet
Send command
Receive packet
Store data to sent command storage
Was already received?
Yes
No
Store whole command to large storage
Receive confirmation
Send confirmation to the sender
Remove confirmed command from sent commands storage

Figure 1: Working of CTP

All important transfer parameters, as IP addresses and ports of sender and recipient, are stored in the UDP header. Each packet can be fully identified by its sender, its receiver, and ID. Sender provides uniqueness of ID for each recipient in the following way: initial value of ID for next packet that will be sent is taken as pseudorandom number. After sending each packet, it is to be incremented. The very first packet sent to the recipient, is to be marked with special option to allow recipient to learn the value of the starting ID.

- The command to be send is divided into packets and each packet is send. All the data sent is stored in command storage.
- The recipient receives a packet and check if the same packet has been already received. If such a packet was already received, that means that the sender failed to get the confirmation, so confirmation has to be sent again and this packet's receiving procedure can be skipped.
- Confirmation has to be, exactly, "sent again", not "resent". Confirmations are not stored in the sent packets storage, but are generated when needed.
- If such a packet has not been received earlier, then information about its arrival needs to be stored.
- If got packet represent the normal command, then the server informs the application about data arrival (creates delivery and puts it to the deliveries storage). If it is a part of a large command, then the server stores it to the large command storage. If it is the last remainder part of the message, then delivery also is to be generated.
- After the packet has found its place, confirmation has to be sent, and the recipient begins to wait for the next packet.
- When the sender receives any confirmation, it is to delete the corresponding record from the sent packets storage. The mechanism, like a physical system, aspires to minimize its potential energy, to free all storages as soon as possible.

IV. PROTOCOL HIGHLIGHTS

This section gives an overview of CTP suite, including those that implement reliable delivery, transmission control, intelligent resending, message ordering and many more.

4.1 Storages:

It stores description of each workstation the current one communicates with. Among description, next packet ID, interchange timeout, and description of packets received from this recipient, are meant here. Messages can be resent.

If confirmation of packet's arrival will not be received during timeout, then the packet is to be resent. A new entry is added to session information storage when the first message is going to be sent or was received from workstation, or is unknown yet. To send the command, packets are to be arranged. Some memory is to be allocated and filled with packet headers and data. The fact is that it will not be freed and unallocated just after sending, but stored to the sent commands storage. A record can be removed from sent commands storage only after all its packets arrival has been confirmed. To store large message, special storage is used, when receiving it part by part. The special storage stores the total amount, the vector of parts receiving status, and a buffer for compiling.

4.2 SmartBuffers:

Smart buffers save CTP's implementation from redundant memory allocations. It reserves the place for the packet's header once per definite size of the data (maximum amount of data in a single packet) on the fly. So, the user just puts data to the smart buffer. Then sending function inserts headers, and packets are ready to go out.

4.3 Confirmations:

Confirmation is a packet with empty body (header only), which has only three differences from headers of the packet having been confirmed. In confirmations header:

- Packet's size is set to header's size.
- In command's number, highest bit is set.

- Message size is set to zero.

It can be considered as an inefficient solution to confirm each packet with separate confirmation, but it was done to provide more features by using options.

4.4 Timestamp:

Timestamp is needed when a network error occurred, for example, for building log files. The time when the error's description has been delivered may differ greatly from the time when it had taken place. To avoid such mistakes, timestamp was decided to be included .Timestamp for the current moment can be always retrieved with the help of the static member function with the accuracy of thousandth of a second.

V. CONCLUSION

In this paper, we have described an approach for building transport services, as well as a concrete realization of the approach in the form of CTP. The ability to customize transport protocols can provide important flexibility when it comes to supporting new applications and new network technologies. In the family of transport protocols, various attributes such as reliable transmission and congestion control are implemented as separate micro protocols, which are then combined in different ways to provide customized semantics. The goal of CTP is not to be yet another transport protocol or yet another TCP extension, but a prototype of a completely customizable transport protocol that can be configured to serve any application domain in any execution environment. Extensions to TCP have also been developed to improve its performance and applicability for specific application or execution domains. Examples of such extensions include selective acknowledgments [7] and support for transaction-oriented services [8].This protocol has been implemented on a small network for the solution of nonlinear optimization problems, i.e. network flow problems. CTP's implementation doesn't use a critical amount of resources. Of course, existing protocols, for example RTP [5] and SCTP [6], can provide many application benefits. However, each of these protocols had to be constructed from scratch, and is not easy to modify to support other, different application needs .We plan to study a specification language for controller decision rules description. We shall also concentrate on the design of a decentralized environment for high performance peer to peer distributed computing. Other future work will concentrate in three areas. First, we intend to use CTP as an experimentation and prototyping platform to implement and measure different transport-related algorithms .Second, we plan to extend CTP to support customizable multicast and group communication. Finally, we will explore further performance optimizations in the CTP.

ACKNOWLEDGEMENT

The authors wish to thank the referees, who provided valuable comments that substantially improved the paper.

REFERENCES

[1] J. Postel, “Transmission control protocol,” RFC 793, 1981.

[2] ——, “User datagram protocol,” RFC 768, 1980.

[3] H.H. *Jones A., Ohlund J.* Network Programming for Windows - Microsoft Press. 2000.

[4] S. Iren, P. Amer, and P. Conrad, “The transport layer: Tutorial and survey,” ACM Computing Surveys, vol. 31, no. 4, pp. 360–405, Dec 1999 .

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP: A transport protocol for real-time applications,” RFC 1889, 1996.

[6] R. R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. J. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, “Stream Control Transmission Protocol,” Internet Draft, 2000.

[7] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP selective acknowledgment options,” RFC 2081, 1996.

[8] R. Braden, “T/TCP—TCP extensions for transactions,” RFC 1644, 1994.