PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 06/13

MR-UDP: Yet another Reliable User Datagram Protocol, now for Mobile Nodes

Lincoln David Nery e Silva

Markus Endler

Marcos Roriz

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 06/13 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena April, 2013

MR-UDP: Yet another Reliable User Datagram Protocol, now for Mobile Nodes

Lincoln David Nery e Silva Markus Endler

Marcos Roriz

{lnsilva, endler, mroriz}@inf.puc-rio.br

Abstract. This paper describes the main characteristics and functioning of the *Mobile Reliable UDP* (MR-UDP). It extends Reliable UDP and provides reliable connectivity with mobile clients which execute behind Firewalls with NAT, and which may change their IP address and port dynamically.

Keywords: mobile communication, middleware, network protocol.

Resumo. Essa monografia descreve as principais características e funcionalidades do *Mobile Reliable UDP* (MR-UDP). O protocolo é uma extensão do UDP confiável (Reliable UDP) e permite comunicação confiável com clientes móveis localizados atrás de Firewalls com NAT. e que podem trocar dinamicamente seus endereços IP ou porta de conexão.

Palavras-chave: comunicação móvel, middleware, protocolo de rede.

In charge for publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Table of Contents

1 Introduction

The MR-UDP aims at providing reliable communication based on UDP from/to mobile nodes (MNs), with least possible overhead. It extends a Reliable UDP (R-UDP) [1] protocol with mobility-tolerating features, such as the ability to handle intermittent connectivity, Firewall/NAT traversal and robustness to switching of IP addresses or network interfaces (e.g. Cellular to WiFi, and vice-versa).

Compared with the original R-UDP, MR-UDP includes mobility-oriented optimizations and extensions in following aspects: transparent continuation of a MR-UDP connection across IP address or port changes; small number of connection maintenance packets for Firewall/NAT traversal; reduced use of mobile device resources and flexible use of threads. These optimizations are important for wireless connectivity. For example, when a MN enters an area with no - or weak - connectivity, it may suffer a temporal disconnection and when the wireless signal is back, the node may get a new IP address (via DHCP). In this situation, whenever the disconnection time is shorter than a threshold (e.g. 30 seconds), MR-UDP will keep the original logical connection and all buffered UDP packets will be delivered in the original order. MR-UDP is used as the wireless/mobile communication protocol in the SDDL middleware [2].

Despite some tests results that show good performance, the MR-UDP was not designed for continuous, high-performance data transmission, such as multimedia streaming. Instead, it was designed for the maintenance of large numbers of MN connections and reliable delivery of messages to/from these nodes. It supports applications where all MNs periodically send some context data in discrete intervals (e.g. every 10 seconds), rather than continuous streaming, as in multimedia transmissions.

2 Main Characteristics

MR-UDP is implemented in Java, and enhances a publically available Reliable UDP implementation named Simple R-UDP[1]. The programming interface provides ReliableClientSocket and ReliableServerSocket classes, which extend the conventional Java Socket's programming interface, with the well-known Socket, ServerSocket, InputStream, and OutputStream classes. MR-UDP has several features inherited from Reliable UDP, such as: acknowledgment of received data packets; in-order packet delivery with selective retransmission of lost packets, and over-buffering. In addition, it has the following singular characteristics that are described in the following sections.

2.1 Simple flow control

A ReliableClientSocket can be used to transfer large amounts of data, by invoking the send() primitive at high frequency (e.g. every few seconds). At the other end of the connection, the ReliableServerSocket may handle thousands of simultaneously MR-UDP connections with remote peers executing the ReliableClientSocket. Since the underlying protocol is UDP, it does not provide any flow control or delivery guaranties. Reliable UDP implements only a best effort reliable delivery of packets, and hence operating system buffers may suffer overflow when large chunks of data are transmitted frequently, causing a progressive and silent (without warnings) loss of packets. So, in order to avoid that the operating system socket looses packets while exposed to bursts

1 Simple R-UDP: http://sourceforge.net/projects/rudp/

of data packet reception, MR-UDP defines a small, configurable waiting time between consecutive sends, at the sender side. While this flow control is much more simple that TCPs sliding window protocol, it is quite effective in most cases (i.e. using 1ms inter-send wait time), and does not require to keep any state of the logic connection.

2.2 Use of UUID

Each node has a unique identifier (UUID) that is generated only once, when the MR-UDP ReliableClientSocket or ReliableServerSocket object is created for the first time at a given mobile node (MN). This 128 bits identifier is used by the ReliableServerSocket to identify the logical connection with each MN, independently of the IP address and port number being currently used by this MN. Thus, the mobile node may switch networks and receive a new IP address, but will still be recognized as the same MN by the ReliableServerSocket. The UUDI also enables to resynchronize the state of the communication (by retransmitting buffered packets which were not confirmed) on both connection endpoints. To implement the unique MN identification, MR-UDP maintains an up-to-date map that records the association between each UUID and its current IP address and port, expressed as the function MN-UIID → "MN-IPAddress:Port". However, the creation and use of a UUID is not mandatory: i.e. if not used, the MR-UDP protocol will work as the original R-UDP, without being resilient to IP address and port changes.

2.3 Types of protocol messages

MR-UDP implements the following types of messages (a.k.a. segments) with the corresponding functions:

- **UID Segment –** carries the UUID of the sending node. When using the UUID identification, this segment is periodically sent, which makes it works also as a heart-beat control message to keep a connection open to a peer node behind a firewall/NAT. This segment is a genuine increment of MR-UDP as an extension of the original R-UDP protocol;

- **SYN Segment –** is used to initiate a new - or reset an existing - connection and to synchronize the packet sequence numbers. It also contains negotiable parameters and optional flags;

- **DAT Segment –** carries a data packet;

- **ACK Segment –** is used to acknowledge in-order received packets. It has an ACK sequence number and contains also the sequence number of the next expected data packet;

- **EAK Segment** - is used to acknowledge out-of-order received packets. It carries a list of sequence numbers of the received packets;

- **RST Segment –** is used to reset the connection by closing and reopening it. When received, a peer node must not schedule any new packet for transmissions, but only try to deliver the not yet acknowledged packets;

- **FIN Segment –** is used to close a connection;

- **NUL Segment –** in the original R-UDP protocol this message is used to check if the node at the other end of the connection is still alive (i.e. AreYouAlive? Message). When received, and if the connection is still active, the peer node must immediately

acknowledge it. In MR-UDP, this segment is used only if a MN does not use UUID for its identification.

2.4 Retransmission of undelivered messages

MR-UDP maintains a buffer of all "not yet acknowledged" packets for each connection. For all packets in this buffer, it tries to re-send them a certain number of times, and wait for the corresponding acknowledgement. When the configured threshold of re-tries is reached, the node considers that the connection has been dropped, and sends a FINSegment.

2.5 Data segmentation

Large messages are split into blocks of data, each of which is sent in a separate DATSegment. The block size is configurable, but in experiments we noticed that block size of 384KB is a good choice for the mobile network scenario. It minimizes the chances of dropping packets (due to overload the networking component of the operating system), and the costs of packet retransmission, while keeping the overhead of acknowledgements manageable and providing a good transmission bandwidth despite use of the inter-send wait time, MR-UDP's simple flow control mechanism. The total size of the MR-UDP buffer is also configurable by a desired number of blocks that must be retained.

2.6 Connection management: disconnection detection and Firewall/NAT traversal

Any endpoint of a MR-UDP connection detects a disconnection when: (a) it does not receive any segments from the remote endpoint for some time interval; (b) it sends a NULSegment to the remote endpoint and does not receive an ACKSegment back within some time interval (c) it has reached the re-transmission threshold for non-acknowledged sent packets (in the connection's buffer).

When using the UUID identification some disconnections that would happen can be totally hidden for the application that uses MR-UDP. When a MN experiences a temporary disconnection from its network and gets a new IP address upon the reconnection, it immediately sends a UIDSegment with its new address. This enables the ReliableServerSocket to learn the node's new location and update the UUID-IPAddress: port map. Thus, in this situation the conventional Reliable UDP connection would be broken, but the MR-UDP continues to work as if the connection were never broken.

Also by using the UIDSegment (or the original NULSegment), a MN behind a Firewall/NAT can keep a connection active, and allows other nodes to reach it. In some sense, this mechanism replaces piggybacking of near real-time messages or control information, as used in other approaches, and makes possible to reach the MN without the need to wait for a message/signal originated from it.

2.7 Data serialization and compression

Since MR-UDP is targeted at mobile nodes, it is very important to minimize the serialization time and the compressed data size. The original R-UDP protocol uses Java built-in serialization mechanism to do these tasks. While this approach works for desktop and other robust domains it is not appropriate for MNs. The problem is that the built-in mechanism uses reflection to pack and unpack segments and does not uses a compression algorithm to write and read its data, thus, writing and reading an enormous

volume of redundant data. In MR-UDP, we used a serialization library called Kyro [2] that uses instrumentation to speed up the serialization time and reduce the segment message size. Through the Kyro instrumentation, we could reduce the segment message size by up to 99% and the compression time by approximately 85%.

2.8 System overview

Figure 1 illustrates some of MR-UDP's features and behavior. On the left side, a ReliableServerSocket is handling several connections (ovals) with remote clients by multiplexing them over its local port X. A pool of threads does this handling. More specifically, it shows two connections - and the associated packet buffers - for clients with UUDIs U1 and U2. Notice that a hashTable, the UUID-IPAddress:port map (purple box) associates each client to its current Internet address and port. If some of this information changes for any known client/UUID, for example, if the corresponding MN is connected to a new network, then the ReliableServerSocket just updates this map. On the right side, client U1 is serializing and segmenting a message object for transmission into packets and putting these packets, one by one, into the OutBuffer. After data packet with sequence number 13, D(13), arrives at the ReliableServerSocket, a corresponding acknowledge packet, A(13), is sent and received by the ReliableClientSocket. Client U2, on the other hand, is reassembling a message object - to be delivered to the application - from its constituent packets D(2) through D(6). After acknowledging packet with sequence number 3, data packet D(4) is lost, but D(5) and D(6) arrive. Client U2 then sends an EAK Segment, EAK(5,6) packet, which causes the ReliableServerSocket to re-transmit data packet D(4). In the meantime, U2 sends another UID segment to keep the connection through a Firewall open. The Figure also suggests that MR-UDP just uses a single port (blue dot), associated with the UDP socket.

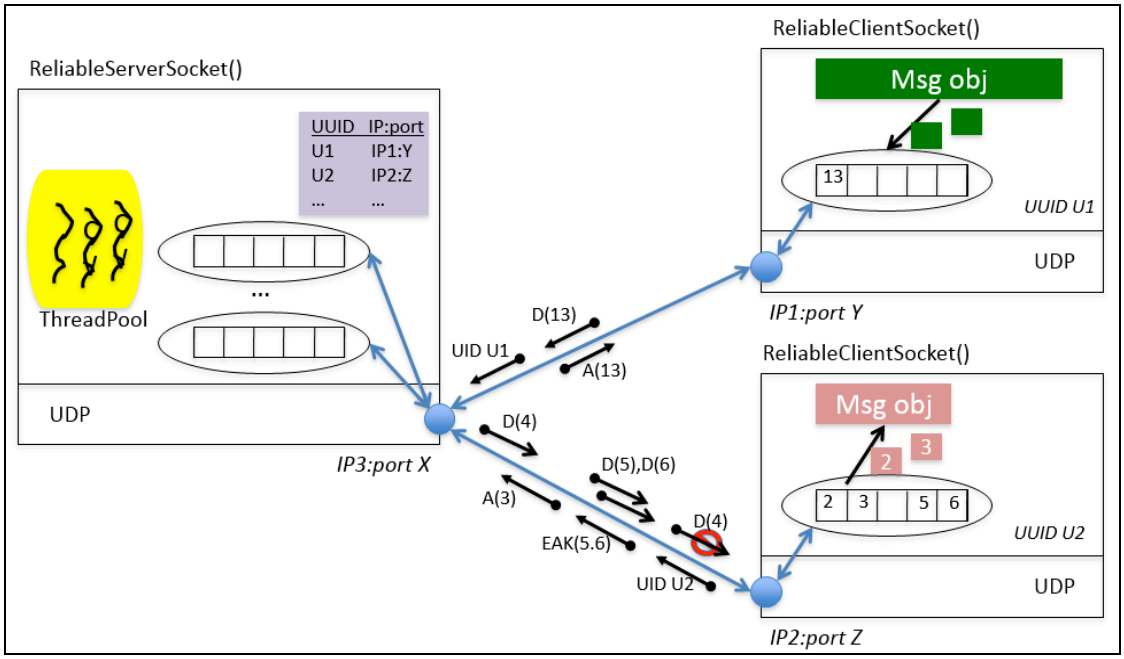

Figure 1. Schematic view of MR-UDP in action with two clients

2 http://code.google.com/p/kryo

3 Some Implementation Details

Since we wanted the MR-UDP protocol to efficiently handle communication in mobile networks, it was necessary to implement several optimizations and extensions to the original R-UDP.

The first one that deserves notice is the use of a pool of threads. In MR-UDP, the pool of threads has small, configurable number of worker threads, which are automatically allocated as new connections are being handled (by the ReliableServerSocket). By this, it is able to manage thousands of simultaneous connections. In the original R-UDP implementation that we used as the starting point of our implementation, every connection used 15 threads! But since threads consume much system resources, this implementation was not scalable, and the result was that many connections were dropped. As all threads in the original R-UDP were used for timeouts (in a class called Timer), in MR-UDP we just implemented a more efficient Timer using the pool of threads. Our tests have shown that with just 3 worker threads, MR-UDP incurs in much less resource consumption, and is able to handle 5-10 times more connections.

When using a transport protocol over an unreliable network, it is important to enable that higher-level protocol layers (such as ClientLib of SDDL) be notified of some states of message delivery or connectivity. For this reason, in MR-UDP we made some changes to the message send and acknowledge APIs. We used the Observer design pattern that enables high-level protocol layers to observe and be notified when specific events happen within MR-UDP. For example, we made MR-UDP inform which was the ACK number of a message sent, and enriched the internal socket listener to inform the number of every ACK received. In this manner, the software layer above MR-UDP can be informed if a message was correctly delivered at its destination, or if a message was not delivered due to some disconnection.

To keep applications compatible, we made ReliableClientSocket (and ReliableServerSocket) extend the default Java socket classes, which means, that any application can use these sockets without significant modifications. Basically, MR-UDP's reliable socket will be created using another constructor (that will inform the UUID), and nothing else needs to be changed.

Another noteworthy characteristic is that MR-UDP implements the Front Controller design pattern, allowing the ReliableServerSocket to use a single UDP port for all connections, i.e. through which all inbound and outbound message traffic is multiplexed/de-multiplexed. For this, as already mentioned, MR-UDP maintains an up-to-date map of the MN's UUID and its current pair (IPAddress: Port).

The Simple R-UDP [4] uses Java built-in serialization mechanism, which is based on reflection iterations over the segment. While the approach is elegant, as it can be written clearly and recursively, it is a slow mechanism due to the high overhead of inspecting the segment reification (metadata) to pack and unpack messages. To speed up the serialization process, in MR-UDP, we used Kyro [5], a serialization library that provides an API for fast and efficient object graph serialization in Java. This library inspects the serialization class only once to build a map, which contains a read and write strategy. The strategy contains general hints on how the data should be written. Using the Kyro library, we instrumented our entire serialization process and reduced the message size by up to 99%. For example, a 1112 bytes message could be compressed to only 9 bytes. Using Kyro we expected the serialization time to be high, since it requires pre- and post- processing in the serialization task. Surprisingly however, we also noticed that it is very efficient in packing and unpacking messages. For example, in some cases, the total time for a serialization was reduced by 85% (on average, each segment took only 7ms to be serialized, instead of the 44ms using Java's built-in serialization).

Finally, we also did some re-factoring to make MR-UDP Android-compatible, since some Java constructs were causing problems in some cases (e.g. causing deadlocks) when running MR-UDP in VM Dalvik.

4 Possible Future Improvements

MR-UDP can still be further improved along several lines, which include:

- Include Error detecting code (a.k.a CRC) into each data packet so that corrupted packets are discarded at the receiver and later re-transmitted by the sender. This is particularly important for the ability of re-assembling serialized objects from its constituent packets.
- Implementation of Quality of Service (QoS) policies for communication, (e.g. minimum throughput, other message delivery policies; priority lanes; segment compression levels; data encryption, etc.)
- Port to other Object based languages and mobile platforms, such as Objective C, for iOS, or C++. This line also requires investigating techniques to express the data carried by the segments in an external data representation (XDR), such a XML and JSON.

5 Other Reliable UDP Implementations

There are also some other implementations of RUDP. One of the most complete one is a .Net implementation called R-UDP[3]. The main features of this implementation are a KeepAlive message, ordered delivery of packets, Rendezvous mode and NAT traversal, Piggy backing of multiple ACKs in a packet, sliding window using slot technique and pluggable Congestion protocol and strategy (currently, with two TCP congestion protocols). Moreover, it also uses threads to take advantage parallel processing on multi-core architectures. This Microsoft version is used in its MediaRoom product for IPTV service delivery over multicast networks.

Another one is a Java implementation the Simple Reliable UDP [4], implemented by Adrian Granados, and which is available as open-source under BSD License and was used as the basis of our implementation. As it name suggests, it includes only the basic in-order, reliable message delivery features, but does not provide any congestion/flow control and other optimizations. As mentioned before, it also uses several threads for each connection, which compromises the protocol's performance when a large number of simultaneous connections are required.

6 Conclusion

The MR-UDP is one of the fundamental building blocks of the SDDL middleware, where it is encapsulated in the ClientLib. MR-UDP has been thoroughly tested and has delivered good performance in all its deployments so far. A ReliableServerSocket can well support up to 10.000 connections with RelableClientSockets at MNs which send their position (GPS coordinates, about 100 bytes) every 6 seconds. And with this load the Round-Trip-Delay for an Unicast message plus its corresponding acknowledgement take only 50 ms, as discussed in [2]. This scenario would not be supported by the original R-UDP, since the default Java heap size would be reached with less than 600

connection due the high use of resources, specially the number of threads. Among MR-UDP's main advantages are a small footprint, the use of a thread pool, its efficient data compression mechanism, and the use of UUID as means of unique identification of mobile clients, independently of their current Internet address. MR-UDP can be downloaded from `http://www.lac-rio.com/mr-udp`

References

[1] Boya, T., Krivoruchka T., CISCO; **Reliable UDP Protocol**, IETF Internet Draft, February 99. URL: http://tools.ietf.org/html/draft-ietf-sigtran-reliable-udp-00

[2] L. David, R. Vasconcelos, L. Alves, R. Andre, G. Baptista, M. Endler, **A Communication Middleware Supporting Large scale Real-time Mobile Collaboration**, IEEE 21st International WETICE, Track on Adaptive and Reconfigurable Service-oriented and component-based Applications and Architectures (AROSA), pp. 54-59, Toulouse, June 2012

[3] **Reliable UDP**, Beta 8.0, http://rudp.codeplex.com (last visited: April 2013) A. Granados, Simple Reliable UDP, http://sourceforge.net/projects/rudp/, 2009.

[4] A. Granados, **Simple Reliable UDP**, http://Sourceforge.net/projects/rudp/, 2009.

[5] **Kryo 2.21** - http://code.google.com/p/kryo/ (last visited: April 2013).