Arab J Sci Eng (2016) 41:2771–2785
DOI 10.1007/s13369-015-2012-3

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Reliable Communication Protocol for Applications in Multi-Robot Systems

Shahabuddin Muhammad[1] · **Mayez Al-Mouhamed**[2] · **Nazeeruddin Mohammad**[1]

Received: 9 April 2015 / Accepted: 10 December 2015 / Published online: 23 December 2015

Abstract Multi-robot systems (MRSs) have a wide variety of applications, such as search and rescue in disaster scenarios, where many robots coordinate with each other to accomplish a task. Such MRS applications use infrastructureless environment in which robots rely on inherently unreliable ad hoc network to communicate with each other. Reliable communication among the peers can greatly enhance the performance of a multi-robot system. This paper proposes a reliable communication protocol (RCP) for applications in multi-robot systems. RCP acts as an interface between MRS applications and the underlying communication framework. RCP accepts data from MRS applications and reliably delivers it to other peers. RCP is transparent to MRS applications as well as the underlying communication hardware. To evaluate its performance, we have implemented RCP on seven Stargate micro-controllers that communicate with each other using an ad hoc network. Further, to test the performance of RCP in MRS applications involving higher number of peers, we have also implemented RCP on laptops with Intel *i*7 microprocessors. The obtained results show that RCP achieves reliability while reducing packet delivery time as well as the number of retries needed to deliver a failed packet.

Keywords Peer-to-peer networks · Ad hoc networks · Multi-robot systems

1 Introduction

Multi-robot systems (MRSs) are composed of two or more coordinating autonomous robots to accomplish a common objective or goal. When compared to single-robot systems, MRSs have better task completion times, higher quality results, and increased robustness against individual robot failures [1]. Because of these benefits, MRSs have several military and civil applications. One of the key applications of MRS is to assist humans in urban search and rescue scenarios in dangerous situations (or after large-scale disasters). MRS is also a good candidate for surveillance and environment monitoring.

Multi-robot coordination is the key factor that influences MRS performance [2]. Team-based MRS applications such as search and rescue typically deploy a fixed number of robots. These robots need to coordinate with their peers to achieve the assigned task. The communication among autonomous robots in MRS can happen implicitly or explicitly. In implicit modes of communication, the robots are restricted to derive the internal states of their teammates through passive observations or sensing their surroundings [3], whereas in explicit modes of communication the robots can request and notify the internal states of teammates. Under noisy and dynamic environments, if robots need to succeed in coordinated efforts, the communication must be done explicitly [4]. Another important application that needs explicit communication is a team of robots playing in a soccer field (e.g.,

✉ Shahabuddin Muhammad
smuhammad@pmu.edu.sa

Mayez Al-Mouhamed
mayez@kfupm.edu.sa

Nazeeruddin Mohammad
nmohammad@pmu.edu.sa

[1] Prince Mohammad Bin Fahd University, Al-Khobar, Kingdom of Saudi Arabia

[2] King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia

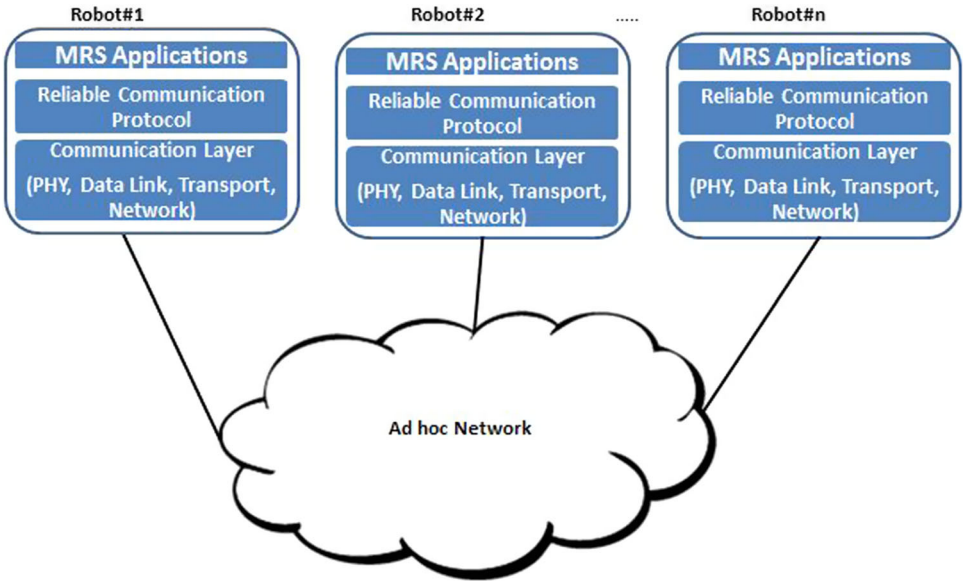

Fig. 1 Reliable communication protocol acts as an interface between application layer protocols and the underlying communication framework

in RoboCup competitions). Each robot needs to inform its current position to other teammates at frequent intervals. Message exchanges continuously take place in such environments, and successful delivery of these messages is essential to achieve the goals of the competition.

The inherent characteristics of all these applications demand an infrastructureless environment that supports reliable communication among coordinating robots. An ad hoc network is a key candidate that provides a communication framework in which individual devices can send their messages to their peers in such applications. The main characteristic of ad hoc networks is their non-reliance on fixed infrastructure. In addition, ad hoc networks also share the same features as peer-to-peer networks, such as decentralization, autonomy, and symmetry of peers [5]. Therefore, ad hoc networks are ideally suitable for applications with multi-robot systems.

As compared to traditional wired networks, wireless channels are noisy and have much higher bit error rates. This results in increased packet loss in a wireless network including ad hoc networks. Therefore, ad hoc networks without any additional support cannot provide reliable communication to MRS applications. This paper presents a reliable and efficient communication protocol for MRS applications in ad hoc networks. We propose a peer-to-peer-based reliable communication protocol (RCP) that works as the middle layer between MRS applications and communication hardware as shown in Fig. 1. RCP receives packets from MRS applications, encapsulates them into RCP data packets (RDPs), and ensures the reliable delivery of RDPs through an underlying wireless network. The RCP layer at the receiving side decapsulates the received RDPs and delivers them to MRS applications. RCP achieves reliability seamlessly without any modification at the application layer as well as without any changes in the communication
protocols.

To investigate the effectiveness of our proposed protocol, we create a testbed environment consisting of seven Stargate micro-controllers that implements a peer-to-peer communication model. Moreover, to test the performance in large MRS scenarios, we have also implemented the proposed protocol in a multi-threaded environment using Intel *i*7 microprocessors. In both implementations, RCP allows each robot to send data without any dependence on other peers.

The rest of the paper is organized as follows. Section 2 summarizes various communication models proposed for MRS in the literature. Section 3 provides some background on why there is a need for mobile, P2P, reliable auction-based communication system. Section 4 presents the architecture, communication algorithm, and implementation details of the proposed protocol RCP. Section 5 explains the experimental results followed by comparisons with other ACK-based approaches. Section 6 presents the results related to the implementation and testing of RCP for large MRS applications. The paper is concluded in Sect. 7.

2 Related Work

A variety of communication systems have been studied for MRS. The proposed communication models for MRS in the literature can be broadly classified as client/server, publish/subscribe, and peer-to-peer (P2P). In the client/server model, one robot acts as a server performing the majority of the processing tasks. The other robots in the MRS act as clients, initiating connections with the server whenever there is a need to request or transfer the coordination

data. In the publish/subscribe model [6], there are publishers which post messages to a group of subscribers that are subscribed to a class. The publishers do not have control over the intended recipients, and they are decoupled with the subscribers. In peer-to-peer (P2P) communication model, any node can communicate randomly with other nodes. P2P model is well suited for MRS applications that need quick coordination among peers without any centralized infrastructure. In general, the performance of MRS communication protocols depends upon the characteristics of network and the purpose of MRS [7]. The following are some examples of proposed/implemented communication strategies for MRS in research literature.

RoboCup Team RFC Stuttgart used a communication system based on a centralized message dispatcher [8]. All agents (robots) establish a TCP connection with a special entity called message dispatcher (MD). All agent communications happen over the MD. In systems where agents are autonomous, MD does not process the messages received from agents except for some filtering. MD sorts the received messages based on priority and dispatches high-priority messages first. Messages are dropped when they become older than a certain predefined threshold. The main benefit of this approach is that agents do not need to have any knowledge about the other agents in the system, and need to know only the MD's address. The main disadvantages of this approach are: (1) Messages are delayed when MD is overloaded; (2) additional communication overhead in maintaining several distributed MDs in order to avoid single point of failure; (3) TCP connections allow continuous stream of data, so agents need to provide their own encoding to differentiate between the messages.

The Cooperative Autonomous Mobile Robots with Advanced Distributed Architecture (CAMBADA) team [9] of the University of Aveiro uses a middleware infrastructure for robot communication. This middleware implements a Reconfigurable and Adaptive Time Division Multiple Access (RA-TDMA) communication protocol with self-configuration capabilities. In order to avoid collisions, TDMA disperses the transmissions of team members in time. The duration of time slots (T_{tup}) is adapted dynamically. Further, they provided mechanisms for self-configuration capabilities to dynamically adapt to the number of active team members [10]. Although this protocol provides a solution for dispersing the robot states, it lacks explicit support for synchronization messages. The synchronization messages are delayed when several rounds of communication are needed between robots. In MRS applications that need a guaranteed delivery, some extra mechanisms need to be implemented. In [11,12], the authors have extended RA-TDMA solution to accommodate synchronization messages among robots. They propose two modes, one for transmission of robot state messages and the other for transmission of synchronization messages. For this, they use UDP multi-cast groups composed of all agents in the team. To tolerate packet losses, all agents multi-cast the responses so that if an agent does not get initial packet it will get all needed information from the next packet. However, this results in unnecessary communication overhead on resource constrained agents and networks.

Several MRS systems use publish/subscribe models for message passing among robots. The Mission Oriented Operating Suite (MOOS) [13] is a popular publish/subscribe-based message passing system in the underwater robotics community. All communications in MOOS are sent through a central server, and clients periodically "pull" messages. The Carnegie Mellon Robot Navigation Toolkit (CARMEN) [14] also uses a central hub to coordinate communications between modules. It uses push-based publish/subscribe model that dispatches the messages to subscribers immediately. This toolkit also provides a configuration option to accommodate direct communications among clients.

The SAE standard Joint Architecture for Unmanned Systems (JAUSs) implements a messaging architecture that enables communication and control of unmanned air/ground/sea systems [15]. JAUS is designed around a routed message passing system in which each message has a specific destination. JAUS uses a hierarchical organization consisting of subsystems, nodes, and components. Components do not communicate directly with each other, but rather through the node manager. Messages that pass outside of a subsystem boundary are routed by the communicator. JAUS does not guarantee message delivery. Standard and custom JAUS services are utilized to support command and control of unmanned air systems (UAS) and unmanned underwater vehicles (UUV) [16].

The lightweight communications and marshaling (LCM) [17] library is designed for message passing and data marshaling in real-time robotic research applications. It uses push-based publish/subscribe model for message passing, which internally uses UDP multi-cast as transport protocol. LCM does not use a centralized mediator for relaying messages; instead it simply broadcasts all messages to the clients. LCM provides only a best-effort packet delivery mechanism and gives preference to the recent messages.

Robotics operating system (ROS) [18] uses a messaging subsystem that provides a publish/subscribe model and a service-oriented model. Their message passing interface is generic, and different transports are supported including shared memory, TCP, and UDP. The communication system offered by standard ROS alone cannot guarantee a reliable communication among peers [19]. Integrated mission planning for heterogeneous robotic agents (IMPERA) [20] extends standard ROS communication framework with a cloud-based communication layer. They propose cloud-based publish/subscribe mechanism that enables reliable communication between robots using the data distribution service

(DDS) [21] as the transport layer. Their experimental results show that the cloud-based DDS overcomes issues related to packet losses that occur in pure centralized approaches [19].

In [22], authors design a multi-robot system consisting of heterogeneous robots to help disabled and elderly in home environments. They used a centralized server (portal) to collect, aggregate, and process data from robots. Portal also generates appropriate control actions and sends the control signals to robot manipulators. Communication between portal and robots is established using one of the variants of 802.11. Portal uses cloud services for processing computationally intensive tasks such as simultaneous localization and mapping (SLAM), image processing, and the kinematics and path planning. They have also proposed a high level of integration with ROS.

In addition to publish/subscribe communication models, several P2P approaches have been proposed in the literature to efficiently achieve reliable communication in MRS. In [23], a TCP/UDP-based communication is proposed where a central node sends an auction to all intended recipients sequentially using TCP/UDP packets. The central node sends an auction to the next recipient only after receiving the acknowledgment from previous recipient. Such sequential communication is not scalable and increases auction completion time. In UDP Broadcast with Token Passing (UBTP) scheme [23], auctions carrying a sequence (SEQ) of recipients are broadcasted by a central node. The SEQ dictates the order in which the recipients should reply. When the first node sends the reply to the central node, it also sends a token to the next node in the sequence. This process continues until all the recipients in the SEQ are exhausted. If for some reason a token has not reached the next intended recipient, then it acknowledges the auction automatically after waiting for a certain time. An auction is successfully concluded when all acknowledgments are received. Otherwise the auction is repeated with a SEQ containing only the IDs of missing nodes. In general, all of these approaches that employ a central node are not suitable for decentralized MRS. In [24], UBTP scheme is extended to support sending auctions by any peer instead of by a single central node, but this scheme also relies on token passing to maintain the acknowledgment order.

UBPP-Back [25] attempts to remove the tokens by broadcasting the acknowledgments. Upon receiving an auction, the first node in the sequence sends ACK immediately, whereas the remaining nodes wait for an ACK from the previous node in the sequence. UBPP-Back is well suited for environments where packet losses are minimal. Its performance degrades significantly when there are packet losses in the system, which is typical of MRS communication under ad hoc networks. Furthermore, broadcasting ACKs increases the network traffic considerably.

In short, there are several recurring themes used in the literature for MRS communication. Publish/subscribe model is the most commonly used, with TCP being the most common transport protocol [26]. The following are a few main drawbacks of these approaches. Client/server communication model is not suitable in MRS as all robots generally have equal processing power. Therefore, dependence on one robot for the server tasks creates a bottleneck and delays the whole coordination process. Further, communication is tightly coupled between client and server, and connection loss may result in data losses. Publish/subscribe type of communication is also not suitable in MRS as the target robots for a message are decided dynamically by the sender, which implies several class formations resulting in unnecessary overhead. Another drawback is that all these architectures have a centralized component. Even if some publish/subscribe architectures support peer-to-peer communications on a component level, a centralized hub serves as a lookup table for ports and addresses of the modules [19]. Cloud-based solutions eliminate the dependence on central component by distributing the service across many servers. Cloud-based solutions do help in a few MRS applications, but there are many other MRS applications where continuous access to cloud infrastructure is not possible. Peer-to-peer (P2P) robot communication in infrastructureless mode provides an alternative for MRS applications.

This paper proposes a P2P communication protocol called reliable communication protocol (RCP) that provides reliable data delivery to MRS applications without relying on any central component. The proposed protocol uses UDP and does not require any changes in the underlying TCP/IP stack. In addition to typical auction-based MRS applications such as [27,28], RCP can also be used in generic MRS applications.

3 Distributed Intelligence for Multi-Robot Systems

A multi-robot cooperative architecture is a general framework for implementing distributed artificial intelligent behaviors. The most commonly used control architecture in multi-robot systems is based on three-level architecture [29–32], which are:

- At the strategic level, the behaviors are typically deliberative. These are characterized by real-time cooperative decision making. This requires intensive communication due to extensive negotiations, which are difficult to meet the real-time decision-making requirements in a soccer game.
- At the tactical level, the behaviors are reactive and deliberative. Behaviors are characterized by peer process cooperation aiming at implementing a chosen strategy

such as tactical playing using a defensive strategy, role (agent) re-assignment, and synchronization.
– At the operational level, the behaviors are basically reactive, which requires closed sense–think–act loops with tight real-time requirements.

Example of multi-robot systems that need the above intelligent architecture can be found in a set of mobile robots playing soccer [33], an expedition of robots moving in a hostile area [34], a team of rescue robots exploring a building after a disaster [35], etc. In this paper, we focus on the application of a team of mobile robots playing soccer. In the following, we briefly describe each layer in the above three-level architecture:

3.1 The Deliberative Component

The deliberative component is a cooperative behavior (CB) [32,36] that can be found in a team of autonomous robots that exchange their beliefs using a series of collective communications. The objective is to cooperate in evaluating the present opportunity under the current state, exchanging information about the current location of own team and opponents, carrying out some geometric evaluation, resolving conflicts in roles and coverage, and finding the best role assignment to implement a recently tuned strategy. Hence, a CB consists of a team of independent robots that cooperate toward achieving a goal while complementing each other and resolving conflicts. At the strategic level, the game strategy allows tuning to a more or less offensive or defensive strategy in response to some emerging conditions. The game strategy varies based on kick off, half time, final objective, game time, and goal difference. In addition to strategy tuning, dynamic role switching provides a flexible approach to rapidly adapt the robot team to unforeseen situations. Hence, distributed coordination is based on a comprehensive view of the environment. Due to the dynamic nature of the game, it must be implemented as a distributed query on the robots because each has partial environment information.

3.2 The Tactical Component

At the tactical level, the game conditions may require cooperative decision to adopt different collective tactics such as choosing a more defensive tactic when the opponent team is noticeably advancing in the game score. A new tactic is implemented by re-assigning roles (tasks or behaviors) to adapt the team to current team tactic and game state using inter-robot negotiations. Role negotiation and task allocation [29,30,37] lead to dynamic role assignment depending upon changes in game state and emerging opportunities. The above can be accomplished by using two supporting components, which are (1) role playing and (2) commitment. Both of these components are based upon intensive communication with information exchange among the team members.

The main role playing primitives are: cooperative positioning (CP), role negotiation (RN), goal clearance (GC), striker-defender (SD), etc. CP re-assigns roles based on current position of robot, ball, own team, and opponent's team. RN coordinates the analysis of game state and collectively determines whether the team should defend or attack as a function of remaining time and goal difference, and assignment of new roles. GC coordinates moving the goalie which is closer to the ball than any other teammates and passes the ball to some partner. SD assigns the robot closest to the ball as a striker and the others as defenders after examining the distance to ball and the position of the opponent team members as seen by the own teammates [2,38,39].

The commitment is a temporary alliance among a few team members toward the achievement of a team objective like scoring a goal. Examples of temporary commitments are the ball-passing (BP) and dribbling-helper (DH) behaviors. The dribbling with a helper allows an attacker to dribble the ball while coordinating its movements with another attacker in preparation for kicking toward the goal or ball passing. As an example of the above negotiation mechanisms, we describe the behavior–communication aspects of a commitment for ball passing. A robot becomes a kicker when it has control of the ball. When the goal is not visible due to surrounding opponents, the kicker may detect the need to establish a relational commitment with a partner to carry out the ball-passing technique. The kicker broadcasts an auction announcing the detection of an opportunity for scoring and asking its own teammates to make a bid based on their instantaneous game conditions like availability in some field area and a good visibility of the goal. The bids are received and analyzed by the kicker which may return a grant message to the winner, which becomes the partner, and the other bidders become free. In the case where no such a partner can be found the attacker behavior may decode another action like dribbling the ball in some favorable direction. If a partner is found for the ball passing, the kicker re-evaluates the scene and may change its state to prepare for the ball passing. For this, frequent synchronization with the partner is done through peer-to-peer auction (multi-cast) messaging. However, the dynamic game conditions, like visibility of the goal at the kicker and partner, may change into one of the following scenarios:

– The goal becomes quite visible and the kicker must kick the ball and end the previously established relational commitment, which frees the partner.
– The kicker coordinates and synchronizes the ball passing with the partner if the goal is quite visible from its current location. In this case, the partner state becomes the ball intercepting, which may lead to dynamically kick the ball.

– While intercepting the ball, the partner finds the goal weakly visible. This may trigger the need to establish another relational commitment and restart all over again.

3.3 The Reactive Component

The reactive component is based on a set of sense–think–act behaviors, each is represented by a specific finite state machine [30,32,35,40,41]. A typical reactive behavior consists of grabbing an image of the environment, analyzing the image to identify a number of state parameters including self-localization, and using the parameters to traverse a decision tree, which results into an action. Examples of reactive behaviors that are frequently used in RoboCup competitions (humanoid league) [42] are searching-ball, walking-to-ball, stand-behind-ball, kick-ball, etc. In the walking-to-ball, if an attacker is away from the ball, traversing the decision tree leads to "continue moving toward the ball" action until the robot is very close to the ball. Note that soccer game imposes hard real-time requirements such as moving toward the ball, ball capturing, and ball dribbling which all require fast response based on motion, vision and action, but without communicating with teammates (at this level).

3.4 Hybrid Architecture

Research in the last two decades showed that the above three-level cooperative architecture is unlikely to be scalable due to the intensive collective communication and the difficulties to meet the real-time requirements in cooperative decision making. In addition, unpredictable events may offset the benefit of costly planned behaviors, especially in dynamic situations. For this, most of the proposed architectures mainly use a hybrid approach combining of the both reactive and deliberative components. The reactive component improves response time, while the deliberative component supports some intelligent behaviors.

Both the deliberative and tactical components can be built on the top of a specific collective communication layer. The application domain of the proposed approach is the soccer game. The basic communication style consists of a set of well-defined queries. A query is issued by a peer node about a specific topic, which requires imperative reply from all or a subset of nodes including the return of some local topic-related information. The requesting node combines the received information and uses it to traverse its own decision tree. The node may announce the outcome to the concerned nodes seeking their own perception of the outcome. This process can be repeated by other teammates until a consensus is reached. For the above reasons, we propose a P2P reliable, light weight, auction-based communication framework to efficiently support the implementation of a distributed negotiation mechanism.

4 Reliable Communication Protocol (RCP)

This section describes our proposed strategy, RCP, which provides reliable and efficient communication for MRS applications. To achieve efficiency, RCP employs UDP as a transport layer protocol because it has lower overhead as compared to TCP. As observed in [24] TCP performed 4–5 times slower than different variants of UDP-based auction schemes in multi-robot applications. Reliability in RCP is achieved through a combination of ACK sequencing and packet retransmission. Every RCP packet is associated with an acknowledgment sequence that dictates the order in which ACKs should be sent. This reduces the number of collisions by allowing only one ACK to be sent at a time. Furthermore, RCP also uses broadcast/unicast retry mechanism to deal with missing ACKs.

RCP implementation has two main parts: dispatcher and receiver. Each peer participating in the ad hoc network performs these two tasks which run in parallel as threads. The pseudo-code for dispatcher and receiver threads is shown in Algorithm 1. We briefly describe the working of each thread in the following subsections.

4.1 Dispatcher Thread

The dispatcher thread broadcasts RCP data packet (RDP) and waits for the acknowledgments from other nodes in the network to ascertain the successful delivery of the packet. In addition to notifying the sender the successful delivery of an RDP, an ACK can also carry RDP response. For example, an ACK contains auction response in auction-based MRS applications. Each RDP is sent with a sequence ($seq[1 \ldots n-1]$), which defines the order in which recipient nodes should send acknowledgments (ACKs). Figure 2 shows RCP packet format, and Fig. 3 shows a scenario consisting of seven Stargate devices in the network. Node 1 broadcasts an RDP with the sequence "3, 4, 2, 5, 6, 7", so the node with ID 3 acknowledges the RDP first followed by the nodes with IDs 4, 2, 5, 6, and 7 respectively. In general, each node waits for $(s-1) * T_{\text{interACK}}$, where s is the position of the node's ID in the sequence and T_{interACK} is the minimum time between two acknowledgments to avoid collisions. In other words, the node in the first position will acknowledge immediately and other nodes wait for T_{interACK}, $2 * T_{\text{interACK}}$, etc. A dispatcher waits for T_{timeout} to receive all the acknowledgments. If all the acknowledgments are not received in T_{timeout}, then the dispatcher rebroadcasts the RDP with a new sequence containing only the IDs of the missing nodes. The dispatcher retries only for a fixed number of times (n_r) before marking the RDP delivery unsuccessful. This sequencing option can be switched off (using NoSeq mode) in which nodes can acknowledge an RDP without any order. This removes the time restrictions on other nodes to send acknowledgments.

Dispatcher Thread
$pkt\_id \leftarrow 1$; $n \leftarrow$ total number of nodes;
$retransmit \leftarrow false$;
$recd\_acks[pkt\_id, ack1, ack2, \cdots, ackn] \leftarrow 0$;
$seq[1...n-1] \leftarrow 0$;
while *RCP data packet is available* **do**
  generate random ack sequence;
  bcast_pkt(pkt_id);
  wait until $T_{timeout}$ expires;
  if $total\_acks\_recd < n-1$ **then**
    if $num\_retries \leq n_r$ **then**
      refill ack sequence randomly;
      $num\_retries \leftarrow num\_retries + 1$;
    else
      $pkt\_id \leftarrow pkt\_id + 1$;
    end
  else
    $pkt\_id \leftarrow pkt\_id + 1$;
  end
end
Receiver Thread
while $true$ **do**
  Listen on port 2222 and wait for incoming packets;
  if $src\_id == my\_id$ **then**
    continue;
  end
  if $pkt\_type == RDP$ **then**
    if *no sequence used* **then**
      if *ack already sent* **then**
        continue;
      end
      send ack;
    else
      $\exists i \in \{1 \cdots n-1\}$ s.t. $my\_id \in seq[i] \Rightarrow s = i$;
      send ack after $(s-1) * T_{interACK}$;
    end
  else
    if *delayed/duplicate ack received* **then**
      return;
    end
    update $recd\_acks$;
  end
end

Algorithm 1: RCP Dispatcher and Receiver Threads

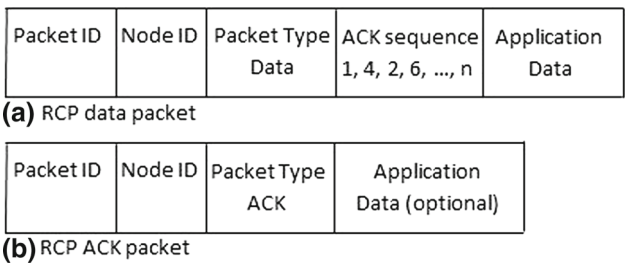

Fig. 2 RCP data and ACK packets

In this case, once an RDP is received, nodes do not wait for $(s-1) * T_{\text{interACK}}$. Rather they send ACKs immediately.

In the case of missing ACKs, RCP may choose to resend the RDP by either unicasting (Uc mode) or broadcasting (Bc mode) it. In Uc mode, RDP is resent to only those nodes whose acknowledgments were not received. The rationale behind this approach is that the majority of nodes reply in the first attempt, and it may be more conservative to send fewer unicast RDPs to the missing nodes than a broadcast RDP to everyone. If RDP is broadcasted, every node needs to process the packet to find out whether it needs to send an ACK, whereas unicast RDPs are sent to only selected nodes. This results in a reduced processing overhead for those nodes whose ACKs were successfully received by the dispatcher in the previous attempt.

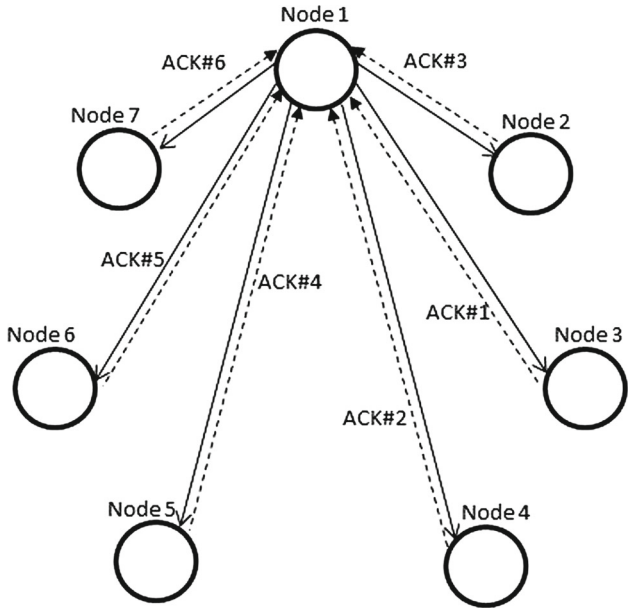

Fig. 3 Schematic diagram of the communication between Stargate devices: node *1* broadcasts RCP data packet (*solid line*) with sequence: *3*, *4*, *2*, *5*, *6*, *7*. All the other nodes acknowledge the reception (*dashed line*) in their respective turns

4.2 Receiver Thread

The receiver thread continuously listens for packets. If the received packet is a data packet, it determines whether its ID is in the sequence. If its ID is missing, it simply ignores the packet. Otherwise it will send an ACK in one of the following two ways.

- In sequence mode, the receiver thread determines its order (s) in the sequence and calculates the waiting time ($(s-1) * T_{\text{interACK}}$) to send the acknowledgment. For this purpose, the receiver starts a new thread which waits for the required time and sends the acknowledgment packet after the timer expires.
- In non-sequence mode, ACK is sent immediately without any delay.

If the received packet is an acknowledgment for the latest RDP sent by this node, then the receiver processes the

packet and updates the record recd_acks as shown in Algorithm 1. All duplicate ACKs received for obsolete RDPs are discarded.

4.3 RCP Traffic Analysis

Let p_b and P_e be a single bit and packet error probabilities, respectively. Let S be the total packet size in bits, n be the total number of nodes (Stargate devices) present in the network, and N_p be the total number of packets generated by the system and r be the number of retries. We look into three cases for analyzing the total traffic generated by the system. In the first case, we assume ideal channel. That is, no error in the packet transmission or reception. In the second case, we assume minimum probability of error in the system in case of non-ideal channel. That is, out of $n-1$ receivers, only one ACK is missing. So auction needs to be retransmitted to only one node. In the last case, we assume maximum probability of error in the system due to non-ideal channel. In this case, none of the ACKs are received by the auctioneer in the first and second tries, so auction needs to be retransmitted the third time. In the case of ideal channel (case 1) and channel with minimum error (case 2), the total number of packets in the system will be the same in unicast-based and broadcast-based retries. That is, $N_p = n^2$ in case 1 and $N_p = n(n+1)$ in case 2. But in case 3, the total number of packets is $[n + r(n-1)]n$ in unicast-based retries and is $(n+r)n$ in broadcast-based retries. That is, for larger MRS, broadcast-based retries introduce lesser traffic in the network than unicast-based retries.

In the case of non-ideal channel, probability of packet error P_e can be calculated as $P_e = 1 - ((1-p_b)^S)$. Since r is the number of retransmissions required to deliver a packet successfully, $P[r = i] = (1 - P_e)P_e^i$ for $i = 0, 1, 2$. Now, the total number of packets sent for unicast-based retries can be calculated as $N_p = 1 \cdot P[0] + (n-1) \cdot P[1] + (n-1)P[2] + (n-1)$ and $N_p = 1 \cdot P[0] + 1 \cdot P[1] + 1 \cdot P[2] + (n-1)$ in worst and best cases, respectively. Similarly, the total number of packets sent for broadcast-based retries is $N_p = 1 \cdot P[0] + 1 \cdot P[1] + 1 \cdot P[2] + (n-1)$.

5 Stargate Experiments and Results

To evaluate the proposed approach, we have created an ad hoc network with seven Stargate devices. A Stargate device is a single-board computer that consists of an Intel 32-bit 400 MHz Xscale processor, 96 MB of memory, and a daughter card with 802.11b Wireless LAN support. RCP is implemented in Java, and the byte code is transferred to the Stargate devices. For our experiments, it is assumed that a device always has RDPs and it broadcasts packets sequentially independent of other nodes in the ad hoc network. This implies that multiple outstanding RDPs (MOR) are present in the system. That is, in a network of 7 Stargate devices, at any given time there will be on average 6–7 outstanding RDPs. The parameters used in Stargate experiments are given in Table 1.

Table 1 Parameters used in Stargate experimental setup

Parameters	Values
Total RDPs per node (Stargate)	500
Number of nodes	7
n_r	2
T_{timeout}	50, 75, 100 ms
T_{interACK}	10 ms

RCP has been tested for four different options depending on whether acknowledgments are sent in a sequence or not, and whether an unsuccessful delivery of an RDP is retransmitted via multiple unicast messages or a single broadcast message:

(1) SeqBc: There is an acknowledgment sequence in each RDP and unsuccessful RDPs are retransmitted using a broadcast message.
(2) SeqUc: This is similar to SeqBc except that unsuccessful RDPs are retransmitted using unicast messages.
(3) NoSeqBc: In this case as in SeqBc, unsuccessful RDPs are retransmitted using broadcast, but acknowledgments for an RDP are sent immediately, that is, without any sequence.
(4) NoSeqUc: This is similar to NoSeqBc except unsuccessful RDPs are retransmitted using unicast messages.

RCP with the above options is compared in terms of average packet delivery time (T_{avg}), total time to deliver 3500 RDPs (T_{total}), total time to deliver 70 % of RDPs ($T_{70\,\%}$), and percentage of RDPs delivered with zero, one, and two retries. The experimental results for the above cases are summarized in Table 2. The detailed results are plotted in Figs. 4, 5, 6, 7, 8 and 9.

Figure 4 represents four cases of RCP in terms of packet delivery time. It plots the percentage of successfully delivered RDPs for various time intervals. It can be seen in the figure that with RDP delivery timeout of 50 ms, both the options that use sequence in ACKs (SeqBc and SeqUc) delivered few RDPs in the beginning (that is, before 75 ms) because each node waits for its sequence before sending an ACK. This implies that the last node in the sequence will wait for $(n-1) * T_{\text{interACK}}$ before sending an ACK. With smaller RDP delivery timeout, a dispatcher considers an RDP to be undelivered sooner than the time it takes for a node to wait for its turn to send an ACK. This is also evident from Fig. 5

Table 2 Summary of RCP results for Stargate with T_{timeout} of 50, 75, and 100 ms

RCP modes	SeqBc	SeqUc	NoSeqBc	NoSeqUc
T_{timeout}	(50, 75, 100) ms	(50, 75, 100) ms	(50, 75, 100) ms	(50, 75, 100) ms
T_{avg} ms	126, 108, 98	135, 100, 97	52, 53, 57	49, 52, 54
$T_{70\,\%}$ ms	140, 120, 100	155, 110, 100	60, 50, 50	55, 50, 50
T_{total} ms	81,856, 71,128, 63,189	82,129, 63,368, 61,628	40,722, 42,735, 46,359	36,826, 40,180, 42,579
Percentage of $\text{Total}_{\text{RDPs}}(n_r = 0)$	2, 37, 76	1, 44, 78	73, 81, 83	74, 82, 85
Percentage of $\text{Total}_{\text{RDPs}}(n_r = 1)$	66, 56, 22	51, 54, 21	24, 17, 15	25, 17, 15
Percentage of $\text{Total}_{\text{RDPs}}(n_r = 2)$	33, 7, 2	48, 2, 1	3, 2, 2	1, 1, 0
Total packets sent	29,365, 26,950, 25,410	52,010, 36,540, 29,260	25,550, 25,235, 25,165	25,445, 25,165, 25,025

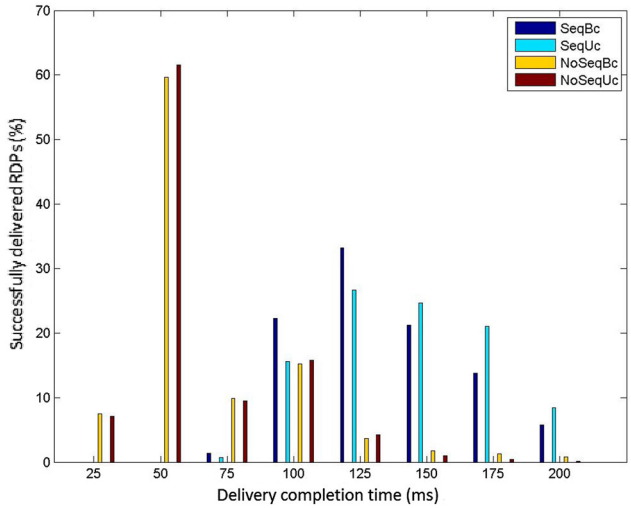

Fig. 4 Distribution of RDP delivery time for different RCP modes with timeout 50 ms

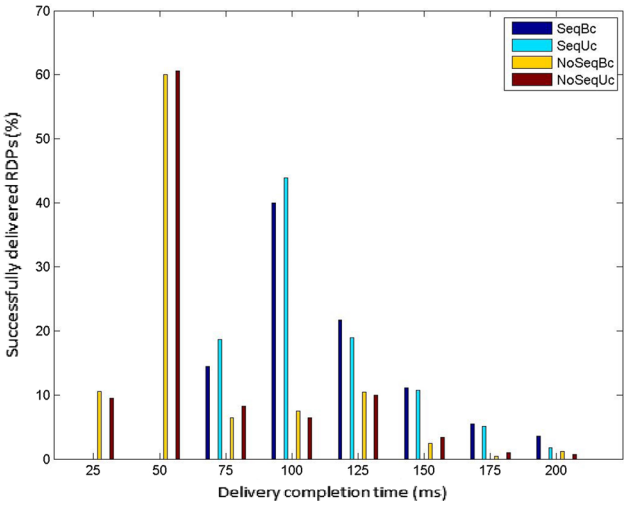

Fig. 6 Distribution of RDP delivery time for different RCP modes with timeout 75 ms

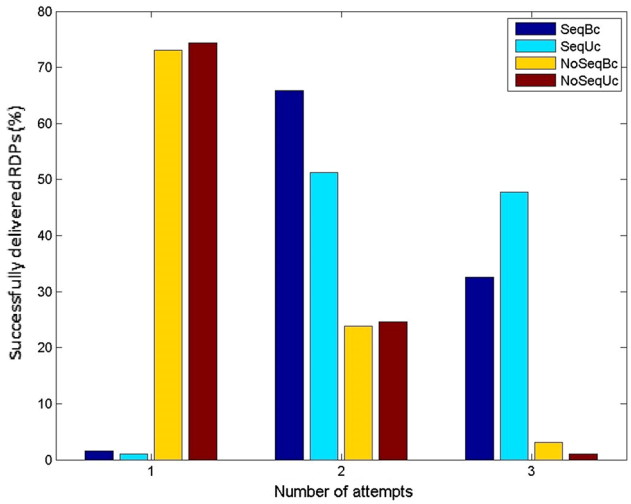

Fig. 5 Number of retries needed for different RCP modes with timeout 50 ms

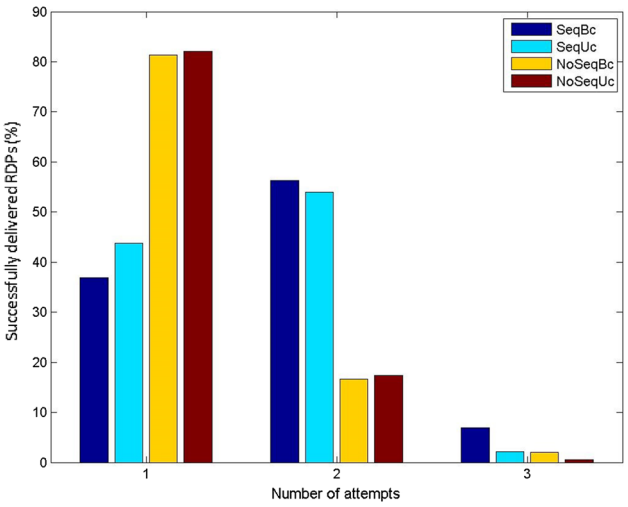

Fig. 7 Number of retries needed for different RCP modes with timeout 75 ms

that most of the RDPs are not delivered in first attempt. The majority of the sequence-based RDPs are delivered in second and third attempts. When the RDP delivery timeout is increased to 75 and 100 ms (Figs. 6, 8), a dispatcher waits a little longer and most of the ACKs are received in a sequence within 75–100 ms. This is the reason that in sequence-based

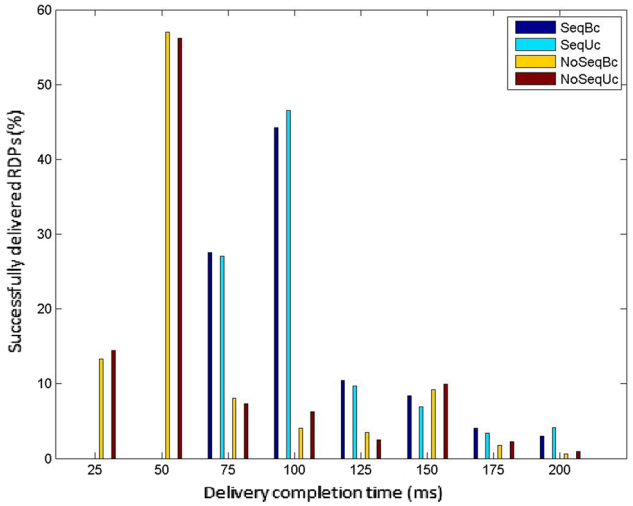

Fig. 8 Distribution of RDP delivery time for different RCP modes with timeout 100 ms

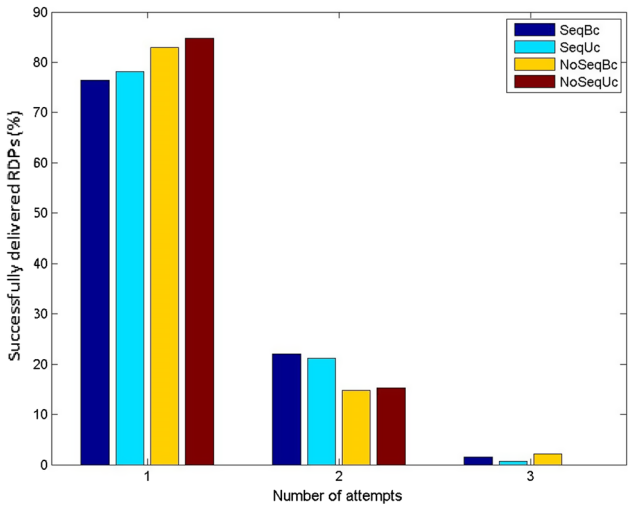

Fig. 9 Number of retries needed for different RCP modes with timeout 100 ms

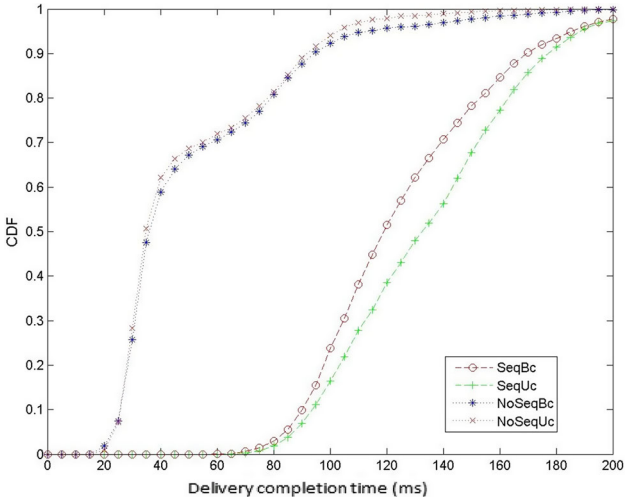

Fig. 10 CDF of time needed to successfully deliver RDP for different RCP modes with timeout 50 ms

approaches almost 50 % of RDPs are delivered in first attempt when timeout is increased to 75 ms (Fig. 7) and the majority of RDPs are delivered in first attempt with 100 ms timeout (Fig. 9). This can also be seen in Table 2 that T_{avg} and T_{total} have decreased with the increase in timeout in sequence-based approaches.

The approaches with no sequence (NoSeqBc and NoSeqUc) deliver most of their RDPs quickly because there is no wait time involved and each node acknowledges immediately. This results in reduced average RDP delivery time (T_{avg}) and total RDP delivery time (T_{total}) as given in Table 2. Furthermore, it is interesting to note that increasing the timeout in non-sequence-based approaches has a negative impact on T_{avg} and T_{total}. This is due to the fact that T_{avg} is proportional to $n_r * T_{\text{timeout}}$, so with higher timeouts a dispatcher has to wait a little longer before retransmitting an RDP, thereby increasing T_{avg} and T_{total}.

It is also clear from the table that resending a failed RDP via unicast or broadcast in both cases (sequence or non-sequence-based ACKs) has only a slight effect in their RDP delivery times where unicast-based retries slightly outperform the broadcast-based retries. Figure 10 shows an interesting scenario where SeqBc performs better than SeqUc. This is because the majority of the sequence-based RDPs could not be delivered in smaller timeout of 50 ms. Therefore, it is better to resend an RDP via a single broadcast instead of many unicast RDPs. The results in Sect. 6.3 will further elaborate this point.

The total number of packets sent by the system is given in Table 2. These results match the analytical values obtained in Sect. 4.3. It clearly indicates that in smaller multi-robot systems there are very few missing ACKs due to which broadcast-based retries introduce similar number of packets in the system as unicast-based retries. Only when many ACKs are missing (as in the case SeqUc with timeout 50 ms), unicast-based retries produce more packets.

In short, the current experimental setup with 7 Stargate devices suggests that using ACK sequence and retransmitting RDPs via broadcast result in performance degradation. We can deduce that in small MRS communication using ad hoc networks (1) it is better not to use sequence because adding an additional sequence requirement in ACK causes extra delay which outweighs the benefits of organizing ACKS to avoid collision; (2) it is better to resend the undelivered RDPs via unicast messages as RDP delivery is usually failed due to missing ACKs from few nodes. Therefore, it is recommended to send a unicast message to these nodes instead of sending a broadcast to all the nodes of the network.

Table 3 Comparison of RCP with similar ACK-based approaches in multi-robot systems

Options	T_{avg} (ms)	$T_{70\%}$ (ms)	T_{total}	Percentage of $Total_{RDPs}$ (ms)	Avg attempts ($n_r = 0$)	Power consumption (mW)
RCP-SeqBc	98	100	63,189	76	1.28	20.73
RCP-SeqUc	97	100	61,628	78	1.23	20.52
RCP-NoSeqBc	57	50	46,359	83	1.19	20.06
RCP-NoSeqUc	54	50	42,579	85	1.15	11.42
UDB-P2P	99	100	63,211	75	1.31	20.77
UBTP-P2P	108	110	69,355	69	1.4	22.85
UBPP-Back	175	205	185,590	53.9	1.5	37.05

5.1 Comparison with Other ACK-Based Approaches

This section compares the proposed approach with other ACK-based approaches for achieving reliability in multi-robot systems. The compared approaches are UDP Broadcast with Token Passing (UBTP) where ACKs are synchronized by tokens [23], UDP P2P Distributed Broadcast (UDB-P2P) [24] where ACKs are synchronized by a sequence number, and UDP Broadcast Peer-to-Peer with Broadcasted ACKs (UBPP-Back) [25] where ACKs are synchronized by broadcasting them. For a fair comparison, a peer-to-peer version of UBTP (UBTP-P2P) is used where all nodes have equal opportunity to send packets instead of a single node.

All the above-mentioned approaches are tested with the same parameters as given in Table 1 except for $T_{timeout}$, which is set to 100 ms. The results are compared in terms of average packet delivery time (T_{avg}), total time to successfully deliver 3500 packets (T_{total}), total time to deliver 70 % of packets ($T_{70\%}$), percentage of packets delivered in one attempt, average number of retries needed to deliver a packet, and average power consumed per packet delivery. The obtained results are summarized in Table 3.

The approaches compared in the table can be grouped together based on whether the ACKs are received with or without a sequence. The results clearly indicate that adding a sequence deteriorates the performance. All the sequence-based approaches (RCP-SeqBc, RCP-SeqUc, UDB-P2P, UBTP-P2P, and UBPP-Back) have worse packet delivery time as well as a higher number of attempts to deliver the packet. Therefore, employing sequence in order to reduce contention is not suitable in such applications. Among the sequence-based approaches, reliance on any external mechanism to achieve sequence in ACKs further deteriorates the performance. It can be seen in both UBTP-P2P and UBPP-Back. As UBTP-P2P relies on tokens to achieve sequence in ACKs, its performance is worse than UDB-P2P, RCP-SeqUc, and RCP-SeqBc. Also, UBPP-Back performs worst among sequence-based approaches as broadcasting acknowledgments introduced extra overhead in the system. On the other hand, RCP approaches where sequence is not used (RCP-NoSeqBc and RCP-NoSeqUc) perform better than any sequence-based approach as the burden of managing sequenced delivery of ACKs outweighs the benefits of reduced contention in orderly delivered ACKs.

6 RCP Performance in Large Multi-Robot Systems

In this section, we evaluate the performance of RCP when the total number of nodes in a team increases beyond seven. As we do not have many Stargate devices to conduct this experiment, we have implemented a different experimental setup. This setup consists of three laptops with Intel Core *i*7 2.8 GHz processors connected through an ad hoc network. Each laptop runs multiple threads where each thread implements a dispatcher which represents a peer of an MRS. All dispatchers are assigned unique IDs and independently send RDPs without any correlation with the RDPs originating from other dispatchers (even if they are running on the same laptop). Each laptop also runs a receiver thread which processes all incoming packets from local and remote dispatchers. The receiver thread is given higher priority over any other thread running on the same laptop.

Using this experimental setup, we have analyzed the behavior of RCP for SeqBc, SeqUc, NoSeqBc, and NoSeqUc. We increased the number of peers from 1 to 16 per node (laptop). As we used 3 laptops in these experiments, the total number of peers varies from 3 to 48. Approximately 500 total RDPs were generated by the system.

Increasing the number of peers has a twofold effect: one on the total number of RDPs and the other on the total number of acknowledgments generated by the system. In order to get a better understanding of these changes, we have conducted two sets of experiments, one with restricted multiple outstanding RDPs (RMOR) and another with unrestricted multiple outstanding RDPs (UMOR).

6.1 Restricted Multiple Outstanding RDPs (RMOR)

In these experiments, the maximum number of concurrent RDPs is limited to 9. That is, at any moment of time there will be at most 9 multiple outstanding RDPs in the system.

Table 4 Parameters used in large MRS scenarios

Parameters	Values
Total RDPs per node (laptop)	500
Number of nodes	3
Number of dispatchers	3–48
n_r	2
T_{timeout}	100 ms
T_{interACK}	1 ms

Table 4 summarizes the list of parameters used in this experimental setup. Notice that T_{interACK} is reduced to 1 ms in these experiments. This is because in the Stargate experiments described in Sect. 5 a typical RDP delivery took at least 50 ms and we had 7 Stargate micro-controllers, so we chose 10 ms inter-ACK time. Otherwise, if smaller timeouts were chosen, timers would have expired before receiving ACKs for any RDP. In the current experimental setup, a typical RDP delivery completes in around 9–10 ms, so even a less inter-ACK time (1 ms) is appropriate to differentiate between sequence versus non-sequence-based approaches.

Figure 11 shows the percentage of successfully delivered RDPs in the first attempt: % of $\text{Total}_{\text{RDPs}}(n_r = 0)$. It is clear from the figure that with the increase in number of dispatchers, every dispatcher has to process more ACKs for each RDP, which increases the likelihood of a missing ACK. Hence, fewer RDPs get delivered in the first attempt. All the remaining RDPs will be scheduled for second and third attempts, thereby increasing the overall average delivery time as shown in Fig. 12. With the increase in number of dispatchers, the performance in all RCP modes degraded in terms of both T_{avg} and % of $\text{Total}_{\text{RDPs}}$ $(n_r = 0)$. This effect is even worse in unicast-based approaches as sending individual messages to many nodes takes longer than sending a single broadcast message. Among all these approaches, SeqBc performed better. To further verify the results, we executed several RMOR scenarios with a varying number of dispatchers and RDP delivery timeout values (T_{timeout}) and found similar results in all the experiments.

Figure 13 shows the total number of packets processed (which includes all sent and received packets) in all RCP modes. The number of packets processed increases with the increase in the number of dispatchers. As expected, for large number of dispatchers, broadcast-based modes result in a higher amount of packets processed by the system when compared to unicast-based modes regardless whether the sequence is used or not.

From these experiments, we conclude that in applications involving large multi-robot systems, sequenced ACKs with broadcasting failed RDPs (SeqBc) results in the best RDP delivery time. However, broadcasting increases the number of packets processed by the system.

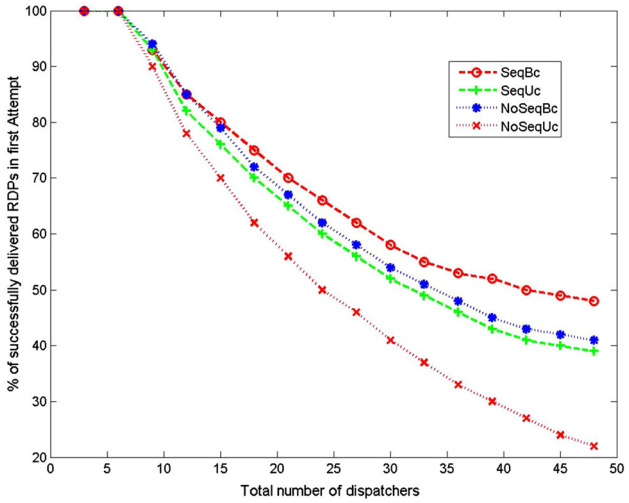

Fig. 11 Percentage of successfully delivered RDPs in first attempt in RMOR

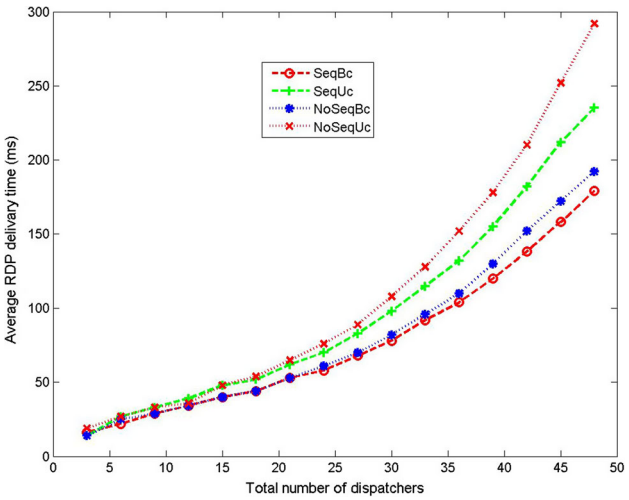

Fig. 12 T_{avg} in RMOR

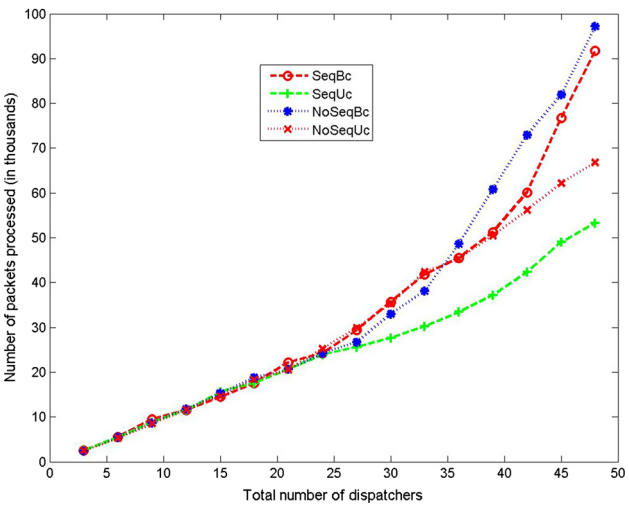

Fig. 13 Number of packets processed in RMOR

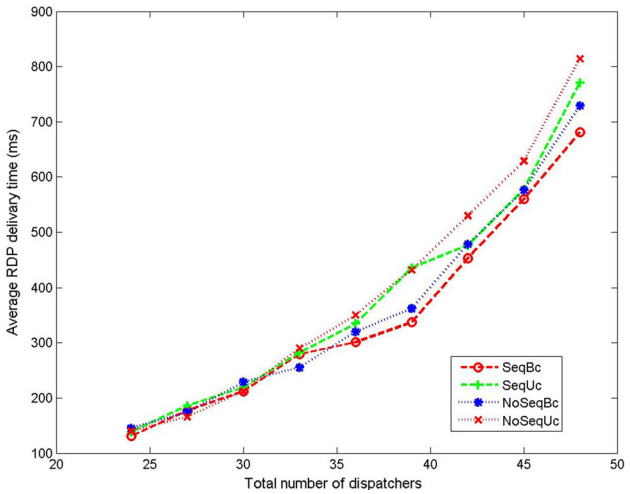

Fig. 14 T_{avg} for all attempts in UMOR (inter-ACK time = 1 ms)

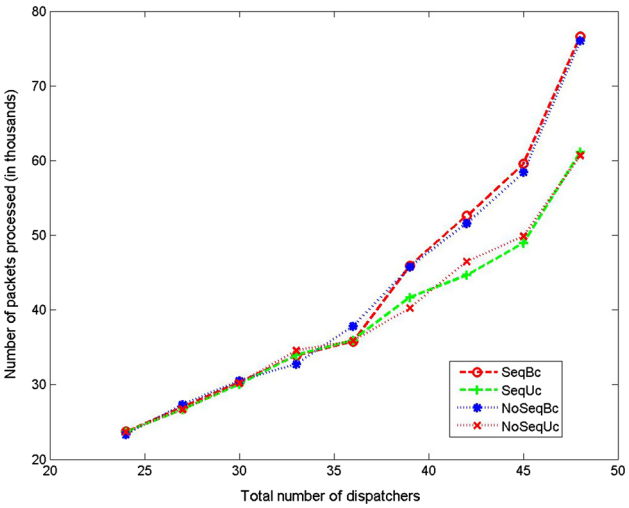

Fig. 15 Number of packets processed in UMOR

6.2 Unrestricted Multiple Outstanding RDPs (UMOR)

We have conducted another set of experiments to evaluate the effect of a higher number of multiple outstanding RDPs. That is, with the number of multiple outstanding RDPs equal to the number of dispatchers at any time. Although MRS applications have a limited number of peers at a given time, we are interested in the performance of RCP under heavy loads. We conducted these experiments with the same value of T_{timeout} as in RMOR. We observed that with this T_{timeout} almost none of the RDPs were delivered. To analyze it further we looked at the RDPs delivered successfully in the first attempt and noticed that the minimum RDP delivery time was more than 100 ms. Hence, we increased T_{timeout} to 500 ms in these set of experiments. Figure 14 plots average RDP delivery time for various number of dispatchers. It can be seen from the figure that SeqBc performs better than other RCP options in terms of RDP delivery time. However, in these experiments the difference in RDP delivery time among all four options is marginal.

Figure 15 shows the total number of packets processed for the various modes of RCP. As expected, irrespective of the sequencing in ACKs, the schemes in which unsuccessfully delivered RDPs are broadcasted resulted in a higher amount of processed packets by the system.

6.3 Unicast Versus Broadcast RDP Retransmission

In order to evaluate the performance of unicast versus broadcast RDP retransmission, we have created two distinct testing scenarios. In the first scenario, the majority of the nodes acknowledge a received RDP in the first attempt, and in the second scenario only a few nodes acknowledge an RDP in the first attempt. In both scenarios, none of the RDPs are delivered in the first try. We are interested in evaluating whether unicast or broadcast should be used when resending an RDP.

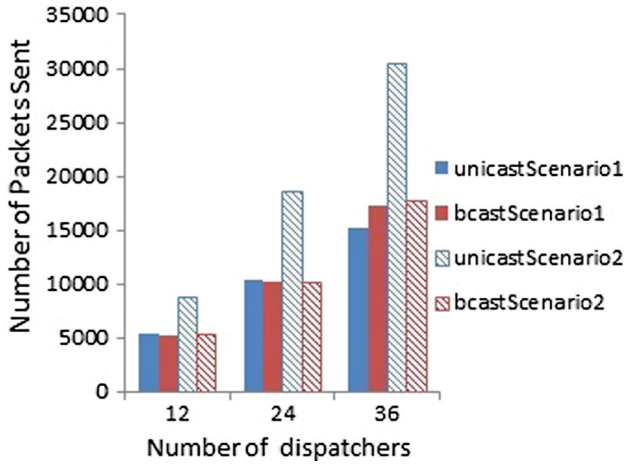

Fig. 16 Number of packets sent in two test scenarios for unicast versus broadcast RDP retransmission

It can be seen from Figs. 16 and 17 that in scenario 1 in which most of the nodes acknowledge an RDP, unicasting a failed RDP results in better performance in terms of the total number of packets received as well as average RDP delivery time. This is because it is simple to send a unicast message to the missing node instead of broadcasting an RDP to all nodes of the system.

On the other hand, in scenario 2 in which most of the dispatchers do not respond to a received RDP, resending the RDP using a single broadcast message puts less burden on the dispatcher, thereby increasing the overall performance of the system in terms of the total number of packets sent as well as the average RDP delivery time.

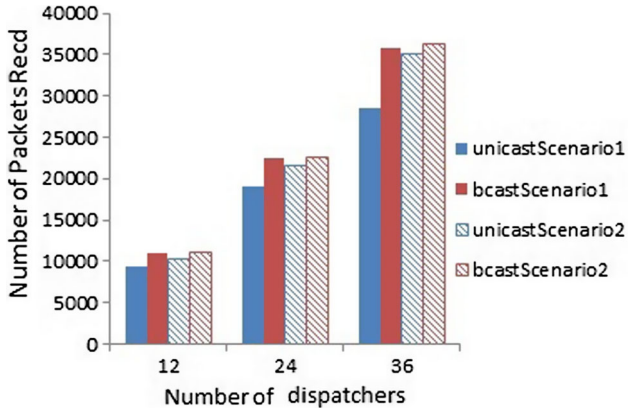

Fig. 17 Number of packets received in two test scenarios for unicast versus broadcast RDP retransmission

7 Conclusion

In this paper, we presented the design, implementation, and experimental results of a reliable peer-to-peer protocol called RCP. RCP uses broadcasts for communication and requires acknowledgments with packet retransmission to ensure reliable delivery. RCP is implemented and tested on Stargate micro-controllers and as a multi-threaded application on laptops. The results from different sets of experiments gave different perspectives about RCP. Experimental results suggest that, in multi-robot systems with fewer robots, sequenced ACKs have a negative effect on the performance. In small MRS, avoiding sequence makes the algorithm simpler, quicker, and efficient. On the other hand, in bigger MRS, sequenced ACKs reduce average RDP delivery time.

The choice of unicasting or broadcasting an unsuccessful RDP depends on the number of missing ACKs. When this number is higher, it is efficient to rebroadcast the RDP, which could be the case in bigger MRS. However, in smaller MRS only few ACKs may be missing and hence unicasting the RDP is preferable.

The results from this study encourage us to develop an adaptive RDP delivery mechanism, which dynamically selects appropriate options depending on various conditions. In future, we are planning to implement dynamic RCP and analyze its effectiveness in multi-robot systems.

Acknowledgments Authors would like to thank King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, KSA, for providing computing support and the Stargate embedded system for the experimental part of this work. Authors also thank Prince Mohammad Bin Fahd University (PMU), KSA, for providing computing facilities.

References

1. Song, T.; Yan, X.; Liang, A.; Chen, K.; Guan, H.: A distributed bidirectional auction algorithm for multirobot coordination. In: International Conference on Research Challenges in Computer Science, pp. 145–148 (2009)
2. Gerkey, B.P.; Mataric, M.J.: Sold!: auction methods for multirobot coordination. IEEE Trans. Robot. Autom. **18**(5), 758–768 (2002)
3. Bonnet, F., et al.: Discovering and assessing fine-grained metrics in robot networks protocols. In: IEEE 33rd International Symposium on Reliable Distributed Systems Workshops (SRDSW) (2014)
4. Parker, C.A.; Zhang, H.: A practical implementation of random peep-to-peer communication for a multiple-robot system. In: IEE Conference on Robotics and Automation, Italy (2007)
5. Ayed, H.K.; Jaidi, F.; Doghri, I.: Fairness and access control for mobile P2P auctions over MANETs. J. Theor. Appl. Electron. Commer. Res. **7**(3), 11–27 (2012)
6. Eugster, P.T.; Felber, P.A.; Guerraoui, R.; Kermarrec, A.M.: The many faces of publish/subscribe. ACM Comput. Surv. (CSUR) **35**(2), 114–131 (2003)
7. Castro, M.: Scalable Communication Protocols in Mobile Networks. Report of the MiNEMA Ph.D. Workshop (2009)
8. Zweigle, O.; Kappeler, U.; Haussermann, K.; Levi, P.: Event based distributed real-time communication architecture for multi-agent systems. In: 5th International Conference on IEEE Computer Sciences and Convergence Information Technology (ICCIT), pp. 503–510 (2010)
9. Neves, A.J.R., et al.: CAMBADA 2011: Team Description Paper (2011)
10. Santos, F.; Almeida, L.; Lopes, L.: Self-configuration of an Adaptive TDMA wireless communication protocol for teams of mobile robots. In: IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp. 1197–1204 (2008)
11. dos Reis, J.C.G.: Distributed Communications System for Multi-Robot Systems. Master Degree Thesis (2012)
12. Reis, J.C.; Lima, P.U.; Garcia, J.: Efficient distributed communications for multi-robot systems. In: RoboCup 2013: Robot World Cup XVII, pp. 280–291. Springer, Berlin (2014)
13. Newman, P.M.: MOOS—Mission Orientated Operating Suite. Massachusetts Institute of Technology, Technical Report 2299/08 (2008)
14. Montemerlo, M.; Roy, N.; Thrun, S.: Perspectives on standardization in mobile robot programming: the Carnegie Mellon navigation (carmen) toolkit. In: Proceedings of the IEEEIRSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2436–2441 (2003)
15. Rowe, S.; Wagner, C.R.: An introduction to the joint architecture for unmanned systems (JAUS). Ann Arbor **1001**, 48108 (2008)
16. Incze, M.L., et al.: Communication and collaboration of heterogeneous unmanned systems using the Joint Architecture for Unmanned Systems (JAUS) standards. In: IEEE OCEANS, Genova (2015)
17. Huang, A.S.; Olson, E.; Moore, D.C.: LCM: Lightweight communications and marshalling. In: Proceedings of the IEEE/RSJ International Conference Intelligent Robots Systems, Taipei, Taiwan, pp. 4057–4062 (2010)
18. Quigley, M.; Conley, K.; Gerkey, B.P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
19. Hartanto, R.; Eich, M.: Reliable, cloud-based communication for multi-robot systems. In: IEEE International Conference on Technologies for Practical Robot Applications (TePRA) (2014)
20. Eich, M.; Hartanto, R.; Kasperski, S.; Natarajan, S.; Wollenberg, J.: Towards coordinated multi robot missions for lunar sample collection in an unknown environment. J. Field Robot. **31**(1), 35–74 (2014)
21. Object Management Group.: Data Distribution Service for Real-time Systems. OMG, Technical Report DDS 1.4 formal/15-04-10 (2015)

22. Benavidez, P., et al.: Design of a home multi-robot system for the elderly and disabled. In: 10th IEEE System of Systems Engineering Conference (SoSE) (2015)
23. Al-Mouhamed, M.; Siddiqi, U.F.: Performance evaluation of auctions WLAN for RoboCup multi-robot cooperation. In: The 7th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), pp. 610–615 (2009)
24. Al-Mouhamed, M.A.; Khan, I.A.; Firdous, S.N.: A reliable peer-to-peer protocol for mobile ad-hoc wireless networks. In: 9th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), pp. 32–37 (2011)
25. Mohammad, N.; Muhammad, S.; Al-Mouhamed, M.: Design and implementation of reliable auctioning algorithms for multi-robot systems. In: 2nd International Conference on Advances in Computing, Communications and Informatics (ICACCI), Mysore, India (2013)
26. Huang, A.S.; Olson, E.; Moore, D.: Lightweight Communications and Marshalling for Low Latency Interprocess Communication. Massachusetts Institute of Technology, Technical Report MIT-CSAIL-TR-2009-041 (2009)
27. Dias, M.B.; Zlot, R.; Kalra, N.; Stentz, A.: Market-Based Multirobot Coordination: A Survey and Analysis. Carnegie Mellon University Pittsburgh, Pennsylvania (2005)
28. Dias, M.B.; Zlot, R.; Kalra, N.; Stentz, A.: Market-based multirobot coordination: a survey and analysis. Proc. IEEE **94**(7), 1257–1270 (2006)
29. der Vecht, B.V.; Lima, P.: Formulation and implementation of relational behaviours for multi-robot cooperative systems. In: Proceedings of 8th RoboCup International Symposium, Lisbon (2004)
30. Manohar, V.; Crandall, J.: Programming robots to express emotions: interaction paradigms, communication modalities, and context. IEEE Trans. Hum. Mach. Syst. **44**(3), 362–373 (2014)
31. Barsalou, L.W.; Breazeal, C.; Smith, L.B.: Cognition as coordinated non-cognition. Cogn. Process. **8**(2), 79–91 (2007)
32. Khalastchi, E.; Kalech, M.; Rokach, L.: Multi-layered model based diagnosis in robots. In: 23rd International Workshop on Principles of Diagnosis, UK (2012)
33. Santos, F.; Almeida, L.; Pedreiras, P.; Lopes, L.S.: A real-time distributed software infrastructure for cooperating mobile autonomous robots. In: International Conference on Advanced Robotics (ICAR) (2009)
34. Zeiger, F.; Kraemer, N.; Schilling, K.: Commanding mobile robots via wireless ad-hoc networks—a comparison of four ad-hoc routing protocol implementations. In: International Conference on Robotics and Automation (ICRA), vol. 19–23, pp. 590–595 (2008)
35. Li, X.; Cai, W.; Turner, S.J.: Efficient neighbor searching for agent-based simulation on GPU. In: IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), pp. 87–96 (2014)
36. Behnke, S.; Stuckler, J.; Schreiber, M.; Schulz, H.; Bohnert, M.; Meier, K.: Hierarchical reactive control for a team of humanoid soccer robots. In: 7th IEEE-RAS International Conference on Humanoid Robots, pp. 622–629 (2007)
37. Calderon, C.A.; Mohan, R.; Zhou, C.: Robot Soccer, Chapter Distributed Architecture for Dynamic Role Behaviour in Humanoid Soccer Robots. IN-TECH, pp. 121–138 (2010)
38. Spaan, M.T.J.; Gonçalves, N.; Sequeira, J.: Multirobot coordination by auctioning POMDPs. In: International Conference on Robotics and Automation (ICRA), pp. 1446–1451 (2010)
39. Urmson, C.; Simmons, R.; Nesnas, I.: A generic framework for robotic navigation. In: Proceedings of the IEEE Aerospace Conference, vol. 5, pp. 2463–2470 (2003)
40. Mantz, F.; Jonker, P.; Caarls, W.: Behavior-Based Vision on a 4 Legged Soccer Robot. Lecture Notes in Computer Science, vol. 4020, p. 480 (2006)
41. De Haas, T.J.; Laue, T.; Röfer, T.: A scripting-based approach to robot behavior engineering using hierarchical generators. In: 2012 IEEE International Conference on Robotics and Automation, pp. 4736–4741 (ICRA) (2012)
42. Mayez, A.; Abu-Arafah, A.: Design of a library of motion functions for a humanoid robot for a football game. In: IEEE/ACS International Conference on Computer Systems and Applications (AICCSA) (2010)