World Academy of Science, Engineering and Technology
International Journal of Electronics and Communication Engineering
Vol:1, No:5, 2007

Managing Handheld Devices in Ad-Hoc Collaborative Computing Environments

Alaa Alwani, Hassan Artail, Haidar Safa, Ayman Abi Abdallah, Bashir Basha, Tarek Abdel Khalek

***Abstract*—**The noticeable advance in the area of computer technology has paved the way for the invention of powerful mobile devices. However, limited storage, short battery life, and relatively low computational power define the major problems of such devices. Due to the ever increasing computational requirements, such devices may fail to process needed tasks under certain constraints. One of the proposed solutions to this drawback is the introduction of Collaborative Computing, a new concept dealing with the distribution of computational tasks amongst several handhelds. This paper introduces the basics of Collaborative Computing, and proposes a new protocol that aims at managing and optimizing computing tasks in Ad-Hoc Collaborative Computing Environments.

***Keywords*—**Handheld devices, Collaborative Computing, main process, Collaboration Table.

I. Introduction and Related Work

SINCE their debut back in 1993, handheld devices have undergone drastic improvement in processing power, storage space, and battery lifetime [1]. Those changes were met with an increase in the complexity of most PDA applications, thus preserving time and computation constraints. Collaborative Computing (CC), originally inspired from parallel computing, was the most dominant approach deployed in an attempt to evade those constraints. CC applies in two distinct ways. The first addresses the lack of communication between individuals by building a collaborative network and the second is based on resource sharing among collaborating computing devices.

An example is the Pervasive Collaborative Computing Environment (PCCE) [2] project that offers an environment for supporting scientific collaborations. This environment houses various tools needed in collaborations such as a synchronous/asynchronous messaging, video conferencing, and file sharing/transfer. Such a project constitutes a high level of collaboration between individuals via the use of computers and wireless technologies. The lower level of collaboration on the other hand, exists when a given task is divided into smaller parts, distributed amongst a set of collaborators, processed, and then reassembled to yield the solved original problem. In addition, another form of this collaboration can be viewed when a specific device lacks the means to perform a computational job, such as text translation, and seeks the help of other devices to insure job completion. Several approaches have been proposed to provide aspects related to low-level collaboration and include the ones discussed in [3], [4], [5], [6], [7], and [8].

A compiler-directed remote task execution system was proposed in [3] for battery power management on mobile devices. Remote process execution was proven in [4] to save battery power through experiments that were conducted on portable computers. In [5] a model for distributed memory management between handhelds and workstations was proposed. In this model, inactive memory items (programs and data) are moved temporarily to other devices in order to make space for large programs. A power-aware cost-based distributed computation model is presented in [6] for the purpose of fairly allocating computational tasks among mobile devices in ad-hoc wireless networks. The use of parallel processing is suggested in [7] to overcome the shortage of processing power in handheld devices. In the implementation, Java message passing was employed in the mobile ad-hoc network for task communication. A protocol was suggested in [8] for remote execution of processes based on the use of multicast for discovering computational resources.

II. The Protocol

A. Scope

This paper proposes the Collaborative Computing Protocol (CCP), which aims at managing the low-level collaboration between devices for implementation at the data link layer.

In this work, we classify the interacting devices into two classes: master devices that seek help from others in order to solve a certain computational task and slave devices that are willing and ready to offer the requested help. Communication between master and slaves takes place in an Ad-Hoc environment in which the master uses broadcast for slaves discovery. In order to come up with the basic block of this collaboration protocol, a set of assumptions governs its scope. First of all, the protocol is considered to work between handhelds of homogenous nature, specifically communicating using 802.11 [9] and are synchronized in time. Second, the medium is assumed to be reliable, eliminating the burden of error checking and packet resending. In addition, security issues are of no concerns at the moment since we assume that no malicious use of this protocol will be deployed to gratify

Manuscript received March 30, 2005.

Alaa Alwani, Dr. Haidar Safa, Ayman Abi Abdallah, Bashir Basha, and Tarek Abdel Khalek are with the Computer Science department at the American University of Beirut (e-mail: {aba09, hs33, asa28, mib06, tsa10 }@aub.edu.lb).

Dr. Hassan Artail is with the Electrical and Computer Engineering Department at the American University of Beirut (e-mail: ha27@aub.edu.lb).

P.O.Box: 11-0236, Riad El-Solh, Beirut 1107 2020, Lebanon.

evil intentions. Finally, the master's code that is fed to the protocol in order to be divided and distributed is assumed to be containing parallelizing constructs. One part of this code, which is relatively bigger than the others, will be executed at the master and we shall call it the *main process* and will wait for all other processes to terminate upon completion. Hence, the code of the program to be divided is composed of a relatively big chunk to be executed as the *main process* on the master device, and a number of subtasks to be distributed on the collaborators. We assume that these subtasks are non dependent in order to minimize communication costs, and reserve some more complex ones to future work.

B. Relation to other protocols

In order to be correctly placed in the TCP/IP stack, relations between CCP and other protocols have to be analyzed. First, we have to keep in mind that CCP is a protocol used for managing the collaboration between handheld devices therefore it has to define a number of control messages that are communicated using 802.11. To ensure efficiency, CCP should be placed at the lowest possible layer in the stack. The first idea was to include CCP at the internet layer where it can communicate with the Internet Protocol using IP addresses for message exchange. However, we realized that in CCP the principle motivation that led to the creation of the IP protocol, namely unifying the addressing scheme, seems to be of less importance. For instance, the user does not need to reference the address of the devices since the service offered is based on a broadcast discovery of collaborators, and then it is the responsibility of the operating system to divide and send tasks to other devices. Also, routing decisions and forwarding packets are not needed since we are working in a single-hop Ad-hoc environment where messages are broadcasted from master to slaves and then sent from slaves to master. Finally, the model being studied assumes an interaction between homogenous devices (using 802.11), thus, one of the motivations behind using the Internet Protocol is lost. Hence, building the Collaborative Computing protocol does not require the use of the IP layer as an essential one, and therefore can survive on the data link layer without the need of IP addressing and by depending on hardware addressing only. The advantage of this approach is that it spares developers from worrying about the IP layer and its complexity (Fig. 1).

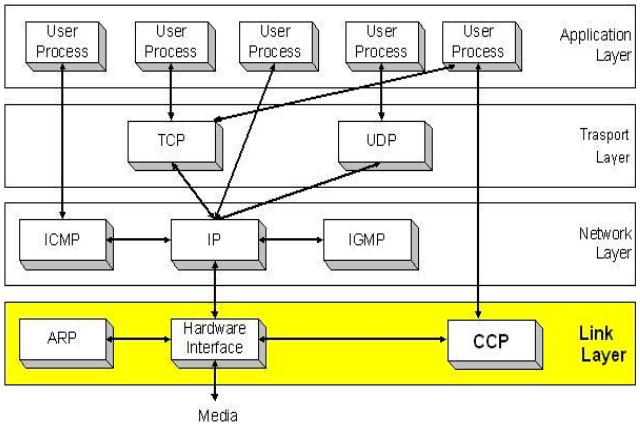

Fig. 1 Protocol relations. The location of CCP in accordance to other protocols.

C. Protocol Description

CCP defines a number of messages that share a common header, which includes six fields: Type, Time To Respond (TTR), checksum, Local Process ID (LPID), Remote Process ID (RPID), and sender Timestamp. CCP first begins by executing the *main process* of the specified job on the master side. After that, a CCP_DISCOVER message, in which the LPID field of the header is set to the *main process* ID, and the RPID is set to 0's, is broadcasted to inspect the availability of any neighboring devices.

The protocol permits any handheld to specify whether it would like to accept incoming discovery messages. The ones that are willing to help will reply with a CCP_OFFER message, whereas others will discard the message. A device that doesn't reply within the TTR time, set in the message, will not be considered for collaboration. If no slaves reply, then the master device will prompt the user to decide whether he wants to execute the job locally or cancel it. In case of replies, the offer message is accompanied by a payload describing available software and hardware resources. The CCP_OFFER sets the LPID to the ID of a newly created dummy process that allocates a portion of the slave's resources to be offered. These resources are calculated on the slave device as to maintain its basic functionalities and not affect any of the currently running processes. The offer message also sets the RPID to the ID of the master's *main process*. If the slave did not receive any message from the master within the TTR time, it destroys the previously created dummy process thus freeing its resources.

Upon receiving the offer message, the master will select convenient slaves depending on their offers. In addition it will create a *Collaboration Table* (CT) (Fig. 2) that serves to map the selected slaves to their assigned tasks. The master's *main process* ID occupies an entry in the table as LPID and the processes on the slaves that are intended for collaboration are identified by their unique tuple <RPID, MAC Address>. If on a single master device we have more than one task to be divided, it is the LPID entry that will uniquely identify the group of collaborators working on this job. The first row of this group identifies the collaborator that holds the first subtask of the main program and so on. For instance, Fig. 2 illustrates a CT at the master side upon receiving offer messages from multiple slaves willing to help in different tasks. As we notice two slaves have offered their services to help in accomplishing the task with *main process* 411. These latter created dummy processes with ids, respectively, 233 and 9985. The CT is also useful in assembling the final results received from the collaborators, especially when we have simultaneous collaborations.

Having constructed the CT at the master device, this latter

	LPID	RPID	MAC Address
0	411	233	0C-08-50-FE-76-6B
1	411	9985	0B-29-74-FE-76-6C
2	412	2456	0A-06-79-FE-76-6D
3	412	279	0F-08-75-FE-76-6A
4	413	982	0F-08-75-FE-76-6A

Fig. 2 The Collaboration Table (CT). The 3-tuple <LPID, RPID, MAC> defines a unique master-slave collaboration. In this example we have three collaborations taking place. Notice that the slave with MAC Address 0F-08-75-FE-76-6A is participating in two of them.

sends a CCP_ACCEPT message as an acknowledgment to the selected slaves. The LPID in this message is set to the ID of the *main process* whereas the RPID is set to the ID of the dummy process waiting at the slave to be populated by the assigned task. Thus, the master's task is now divided into many subparts to be computed independently. The master also takes part of the collaboration and handles the bigger portion, the *main process*, hence, favoring slave devices over itself.

Upon receiving the accept message, a similar CT table is constructed on the slave to identify the possibly numerous masters that currently require the collaboration of this device. After sending the accept message, the master will transmit the assigned task to its corresponding slave using a reliable protocol such as TCP. The slave now possesses a task that can be independently computed without any coordination with the master. Therefore the slaves accomplish these tasks and, upon success, send a CCP_DONE message to the master notifying it about its completion. Following the done message, the result is also sent using a reliable protocol. The slaves will not delete any accomplished task before receiving a CCP_FIN message from the master implying that it has successfully received the result. If, on the other hand, the TTR time of the CCP_DONE message has expired, the slave will resend the result to the master and waits again for a CCP_FIN message. Resending the message is necessary since it is possible that the slave device goes out of range of the master device temporarily. The number of resends is left as an implementation issue leaving room for optimizations. Upon receiving the CCP_FIN message the slave permanently deletes its assigned task and updates its CT, thus freeing the corresponding resources. When the master receives all subtasks, it identifies each subtask by consulting the CT, specifically the <RPID, MAC Address> tuple.

While the handheld devices are computing their assigned tasks, the master may decide to abort the whole job. This results in sending a CCP_RESET message to all its slaves to avoid any unnecessary computations. Once receiving this message the slaves will silently destroy the collaborating process and free all the corresponding resources. On the other hand, a failure or intentional termination of the process on the slave side during the computation will also result in sending a CCP_RESET message to the master informing him about the lost task. Consulting its CT the master will know which task has been lost from the LPID of the CCP_RESET message as well as the MAC Address of the sender. The master then rescans the neighborhood by sending a new CCP_DISCOVER with same header as the one before. If any new collaborators are detected, the failed subtask is resent completely to a new device or divided further and distributed to new collaborators. If on the other hand, no new devices are detected, the master will queue this subtask, to be computed locally.

A mechanism to test the availability of the slaves within the transmission range of the master is receiving *alive* messages, CCP_ALIVE, from the slaves periodically. If the master doesn't receive an *alive* message from a specific slave having an entry in the CT, it is assumed that the slave went down or out of range, and the same error handling mechanism used previously is deployed.

D. Message Format

CCP messages (Fig. 3) share a common header as stated in the previous section. This part summarizes the function of the header fields as well as the purpose of each message type.

1) *Header Fields:*

- *Type*: Specifies the type of the CCP message.
- *TTR*: Time To Respond. Time in milliseconds. A bit corresponds to 10 milliseconds.
- *Checksum*: Error checking mechanism.
- *LPID*: Local Process ID. Identifies the process running on the source device (master or slave).
- *RPID*: Remote Process ID. Identifies the process running on the destination device (master or slave).
- *Timestamp*: a 32-bit timestamp of the sender device used for synchronization purposes.

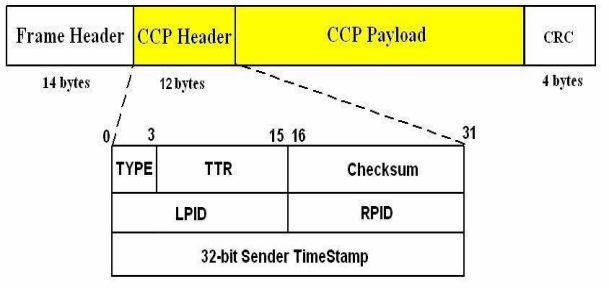

Fig. 3 The CCP Message Format. The message header that will be sent using the collaboration.

2) *Message Types:*

- *CCP_DISCOVER*: discovery message broadcasted by the master to detect any neighboring handheld devices. Payload: empty.
- *CCP_OFFER*: offer message sent from slaves to master as a response to a CCP_DISCOVER message. The message contains information that the slave is willing to offer. Payload: software and hardware information.
- *CCP_ACCEPT*: accept message sent by the master to a subset of the offering slaves. Payload: empty.
- *CCP_DONE*: done message sent by the slaves to inform the master that they have accomplished their assigned tasks. Payload: empty.
- *CCP_FIN*: finish message sent by master to slaves as an acknowledgment to a done message. Payload: empty.
- *CCP_RESET*: error message sent by master or slave to indicate a failure. If it is sent by the master, the slave should halt its assigned task. On the other hand, if it is sent by the slave, the master should send the task to another neighbor or execute it locally. Payload: empty.
- *CCP_ALIVE*: alive message sent by the master to test if a slave is up and in transmission range. Payload: empty.

III. Simulation and Results

To illustrate the protocol, we had to implement a simple application that summarizes the interaction between handhelds. The application was only intended to illustrate the

protocol and not implement it, hence providing a simulation of it on the application layer. The problem chosen for this purpose is a string permutation problem in which a set of 8 characters are permuted to generate all possible strings. The simulation doesn't support error handling however; it assumes that the medium is reliable and that no device will be intentionally shut down. To make it more realistic we assume that computing one eighth of all the combinations is an atomic operation and cannot be furthermore divided, thus simulating cases where applications are not fully parallelizable.

The simulation is carried out by specifying the number of neighboring slaves to the master device. Two methods were implemented to solve the problem of string permutation. The first one insures that the load is balanced between master and slaves. The second on the other hand assumes that the slave devices have limited capabilities and can only generate a certain number of permutations. In our case of 8 characters the total number of permutations should be 8! = 40320 entries. In the second method of simulation mentioned above we assume that a slave can only generate 5040 entries which is one eighth of the total number of required strings. Table 1 illustrates the time required to compute several fractions of the total entries on a single device whereas Fig. 4 illustrates the performance of the protocol in terms of the number of slaves. First, let us compute the performance gain in terms of speedup factor where load balancing takes place. In the case of a single slave, both master and slave will have to compute half of the entries. According to table 1, 40320 entries are computed in 26100 ms whereas with one helping device and with load balancing it took 13426 ms (see Figure 4). This implies a speedup factor of 1.94 and a 49.6% which points to a good performance in the case of two devices working in parallel. On the other hand if load balancing was not used due to limits in the slave resources, it would take 22029 ms to generate all the results thus yielding a speedup of 16%, not bad in case we have only a single slave with minimal resources. However, Fig. 4 shows that in the case of load balancing there is a problem in which additional slaves yield the same or slightly worse performance when going from 3 to 6 slaves. This is because in the cases of 3 to 6 slaves the master device is a bottleneck solving always one fourth of the entries and the increasing number of slaves is worsening performance since communication costs are increasing. With 6 and 7 slaves, the two methods converge since every device is solving the minimum number of entries. In the case of 7 slaves, the results are generated in 3501 ms and when considering that a single device takes 3154 ms to generate the 5040 entries, one can infer that there is about 11% of time consumed by communication activities. However, this cost is considered a negligible overhead in comparison with the performance gain. As a conclusion, choosing the distribution method depends on the number of collaborators, data size, and communication costs.

IV. Continuing and Future Work

Improving the performance of the protocol by minimizing communication costs and optimizing distribution methods is still under study. Future works should include security and error handling mechanisms as well as more accurate and expressive simulations to finally implement and test the protocol in real situations.

TABLE I
RESULTS ON A SINGLE DEVICE

Fraction of total entries	Number of generated strings	Runtime in milliseconds
1/8	5040	3154
1/4	10080	6209
3/8	15120	9423
1/2	20160	12548
5/8	25200	15553
3/4	30240	18456
7/8	35280	21811
1	40320	26100

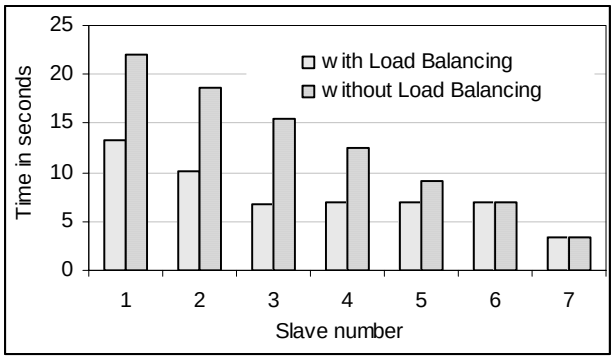

Fig. 4 Running time using CCP.

References

[1] G. O. Young, "Introduction to Personal Digital Assistants", J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.
[2] D. Agarwal, C. McParland, and M. Perry, "Supporting Collaborative Computing and Interaction," Proceedings of the Grace Hopper Celebration of Women in Computing 2002 Conference, October 9-12, 2002, Vancouver, Canada.
[3] U. Kremer, J. Hicks, and J. Rehg, "Compiler-directed remote task execution for power management," *Workshop on Compilers and Operating Systems for Low Power*, 2000.
[4] Rudenko, P. Reither, G. Popek, and G. Kuenning, "Saving portable computer battery power through remote process execution," *Mobile Computing and Communications Review*, 2(1), 1998.
[5] Sathiaseelan and T. Radzik, "Using remote memory paging for handheld devices in a pervasive computing environment," *Asian Journal of Information Technology* 2(1): 08-12, 2003.
[6] L. Shang, R. Dick, and N. Jha, "An economics-based power-aware protocol for computation distribution in mobile ad-hoc networks," *IEEE Transactions on Mobile Computing*, 3(1): 33-45, 2004.
[7] R. Shepherd, J. Story, and S. Mansoor, "Parallel computation in mobile systems using Bluetooth Scatternets and Java," School of Informatics, University of Wales, Bangor.
[8] S. Patwardhan and S. Pichumani, "Ether: a remote execution service for mobile devices," School of Computing, University of Utah.
[9] IEEE Computer Society LAN MAN Standards Committee.Wireless LAN Medium Access Control (MAC) and PhysicalLayer (PHY) Specifications. New York, New York, 1997. IEEEStd. 802.11–1997.