A Data Synchronization Service for Ad Hoc Groups

Terry Coatta*, Norman C. Hutchinson†, Andrew Warfield‡ and Joseph H. T. Wong§
*Silicon Chalk Inc., Vancouver, British Columbia, Canada
Email: coatta@silicon-chalk.com
†Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada
Email: norm@cs.ubc.ca
‡Computer Laboratory, University of Cambridge, Cambridge, United Kingdom
Email: andrew.warfield@cl.cam.ac.uk
§Department of Computer Science, University of British Columbia, Vancouver, British Columbia, Canada
Email: jwong@cs.ubc.ca

***Abstract*— The emergence of wireless ad hoc networking has opened the way for a new style of distributed computing. Ad hoc networks operate without infrastructure such as routers or access points. Software designed for this environment faces new challenges. In traditional networks, partitioning is typically treated as an exceptional and transient condition. The situation in ad hoc wireless networks is completely reversed. Each ad hoc group is an isolated network, partitioned from all other ad hoc networks. As computers move from one ad hoc group to another, the set of partitions changes. This paper presents the design and implementation of a data synchronization service intended specifically for ad hoc wireless networks. It describes the underlying factors affecting the design, the protocol used to achieve synchronization, and some performance measurements from the actual implementation.**

I. Introduction

One of the fundamental problems in distributed systems is managing state that is distributed across a set of computers. There are a wide variety of systems that address this problem, including: distributed file systems, the network time protocol, replicated databases, and the domain name system. Historically, distributed applications that have scaled to support large numbers of concurrent clients have taken measures to locate shared data in manageable, typically central locations and to employ redundancy and replication to survive component failures [1], [2], [3], [4], [5]. One feature that these systems have in common is that they do not work well in the face of network partitions. Without replication of data, a network partition can mean that data is inaccessible, but even if replicated data is available to all participants, network partitions often result in a suspension of service because the system is unable to coordinate access appropriately. In traditional wired networks though, partitioning is an exceptional case that typically involves only a small number of nodes, and is both infrequent and short-lived.

New wireless networking technologies such as 802.11 and Bluetooth have enabled the emergence of an increasingly mobile community of users. These technologies support the creation of ad hoc networking groups, in which users share resources, interact, and collaborate directly without the support of fixed networking infrastructure. This opens up new horizons for how and where computers are used but also present a drastically different environment, whose properties depart significantly from those of wired networks.

Ad hoc wireless networks present several new challenges to software developers: Bandwidth is not as readily available, and is always shared. Reliability varies considerably depending on the distance and objects that separate hosts. Most crucially though, ad hoc networks present a situation in which *partitioning is the norm*. Collaborative users will often work together in small groups, while the entire network will rarely (if ever) be completely connected. Applications intended for this environment require communications mechanisms that acknowledge the disconnectedness that is intrinsic to ad hoc networks.

This paper presents a data synchronization service for ad hoc wireless networks that aims to address these issues. Section II describes the synchronization needs of a collaborative tool that supports teaching in laptop-enabled classrooms. This leads to a set of requirements in Section III. Section IV contains the design and implementation of the synchronization service. The paper closes with comparisons to other systems that deal with similar problems, and a look at future work on the system.

II. Motivating Problem

The data synchronization service described in this paper was developed as one component of a distributed application that is used in classroom or lecture settings to enhance and assist with education in a face-to-face context. The software facilitates typical classroom interactions such as presentation, note taking, asking questions, and working on problem sets. Architecturally, the application is peer-to-peer, that is, there are no statically defined servers/masters. Hosts dynamically take on client or server roles depending on the manner in which the user is currently interacting with others in the class.

During the early stages of development of this application, we constructed a data sharing service that was tied tightly to the particular application domain. However, as work progressed, it became clear that there were several requirements related to data sharing that were recurring in various guises and would be best served by a generic data sharing service

that was application domain neutral. The data sharing needs of the application broke down into three categories:

1) Resource Availability – Participants in the class need access to information such as whether certain individuals are signed in, or whether the instructor is currently broadcasting course content.
2) Application Data – Participants in the class need to exchange various sorts of information to each other such as display names to be used by the application when identifying someone.
3) Application Configuration – The application itself needs to exchange information about how certain resources are being utilized such as which network channel the audio track for the lecture is being broadcast on.

These issues are not unique to the educational setting. Any application that supports groups of individuals coming together and interacting with each other must address these concerns. The data synchronization service described in this paper has applicability to a wide variety of applications in the general domain of collaborative computing.

The educational setting shares other challenging characteristics of the general area of mobile computing. Students are inherently mobile "workers." They travel between classes and study groups, and in each location interact with different, but generally overlapping sets of individuals. The ad hoc mode of 802.11 wireless networks meets this need precisely as it allows individuals to come together and create a local network with virtually no effort.

As individuals move from group to group, their computers gain and lose connectivity to other computers. For example, a student's machine may be in contact with another classmate's during a lecture that they share in the morning, lose connectivity during a subsequent period in which they are at different lectures, and then regain connectivity later when the students are both participating in a study group.

This last issue is one that is particularly relevant for collaborative applications. Users can modify shared data at any time. If a student is interacting with others and makes some changes to shared data, the application propagates those changes to the others immediately. However, if the user makes a similar set of changes at a time when he is not participating in that group (and hence no longer part of that group's ad hoc network), it would be undesirable for other members of the group to fail to see those changes when the group came back together. Thus, the application must remember those changes and opportunistically update other members of the group when connectivity is re-established.

In order to be useful in the collaborative setting, the sort of data sharing outlined above needs to happen with minimal latencies and using a reasonable amount of resources. These constraints rule out the use of transaction-based systems which have been the primary mechanism for achieving shared data consistency in distributed applications. The remainder of this paper transforms the general needs outlined in this section into a formal set of requirements, proposes a protocol for satisfying those requirements, and examines an implementation of the protocol.

III. Requirements

The problem that has been presented in the previous section motivates the need for a globally shared data-space whose complete contents are of interest to every host. In order to provide a usable, consistent shared data-space, we define a set of properties that our service must satisfy: Global Structure, Sporadic Connectivity, Data Convergence, and Scalability.

A. Global Structure

A generic data synchronization service cannot rely on semantic knowledge associated with the data as a means for detecting and propagating changes. While it would be possible to create a service that handled totally unstructured data, our experience is that much of the data shared by collaborative applications can be expressed quite naturally in a hierarchical form. Incorporating this hierarchical structure into the data synchronization service provides a means to efficiently detect and propagate changes in the shared data. Thus, the data space is represented as a hierarchical collection of nodes that we refer to as the *directory* (given the structural similarities to other data services such as LDAP and Active Directory). Individual nodes have the following properties:

- Each node has a name and value.
- Each node has a unique parent node, except the root.
- Each node is uniquely identified by its path from the root.

Each host attempts to store a complete replica of the directory. Individual hosts can create, destroy, or modify the values of the nodes within the directory and the synchronization service will leverage this structure to detect divergences between hosts.

B. Sporadic Connectivity

The primary usage pattern for ad hoc networks is to create (relatively) short-lived, unreliable connections. Because of this, the synchronization service must be able to achieve synchronization quickly and incrementally. To achieve this, the synchronization service will have to quickly identify divergent areas of the directories stored on a group of hosts, and efficiently achieve a consistent view of the data in the identified areas.

C. Data Convergence

As a primary goal of the service is to be lightweight, we must not use transactional interactions between hosts to guarantee consistency. Moreover, as the existence of ad hoc groups implies a constantly changing set of partitions, there are bound to be conflicts in the data that even transactions would not avoid. Therefore, we adopt a very simple conflict resolution mechanism. We assume that the machines using the directory have synchronized clocks and use the host's clock to determine whether a given node is up-to-date with respect to another host. The basic synchronization requirement is that once a node's value has not changed for some time period,

all the participating machines will agree on the value of the node. Note that while many systems take considerably greater pains to identify and sometimes even resolve conflicts, we have found this approach to be very useful for a large class of data.

D. Scalability

Finally, it is important to consider the desired scale of groups under which the synchronization service should be expected to operate. The synchronization service's scale is characterized by two factors: the number of hosts in a single ad hoc group attempting to synchronize their directories, and the number of nodes in the group's shared directories.

The service must scale to support large groups of users as might exist together in a classroom or conference environment (perhaps several hundred users). Regarding the size of the directory itself, we anticipate that it should be reasonable to support a volume of data in excess of what will be required in most group collaborative applications; the current implementation is comfortably supports directories containing thirty thousand nodes.

IV. Implementation

The synchronization service has been implemented as a service for Microsoft Windows. Hosts communicate with each other using a UDP broadcast-based protocol. The current implementation is completely functional and is used as a component of a commercial collaborative application. The remainder of this section will present several aspects of the implementation in detail: the structure of the directory, the synchronization protocol, the structure of the synchronization service, and the application API.

A. Shared Data Space Structure

As described in the requirements section, the shared data-space takes the form of a tree. There is a single, global tree, which may be defined as the union of all trees stored on all hosts using the synchronization service. A database on each local host stores the hierarchy of shared data as a collection of nodes. Each node has two parts: a set of administrative properties that describe the node and are used primarily to represent the tree structure and to support synchronization and an embedded value, which is a string and some associated properties. The structure of a node is illustrated in Figure 1.

The administrative properties of a node are as follows:

- *Node ID:* the identity or path of the node.
- *Reception Time:* the time at which this value was last updated in the local database.
- *Main Hash:* a hash value that characterizes the state of this node and all of its children.
- *Child Hash:* a hash value that characterizes the state of the children of the node only.
- *Child Nodes:* a set of the children of the node.

The properties of a node's value are as follows:

- *String Value:* is the string data, used by applications, stored within this node.

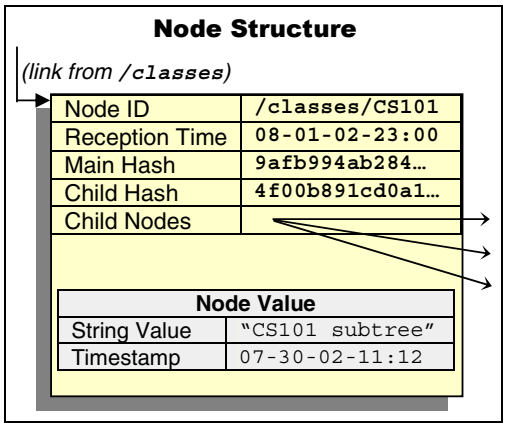

Fig. 1. Structure of an Individual Node

- *Timestamp:* stores the time at which this value was created. This value is used to select the dominating node in the case of a conflict.

The hash values stored in each node deserve some additional explanation. The purpose of the hash function is to reduce a relatively large state space to a finite set of integers in such a manner that if two elements of the state space have the same hash value then it is extremely likely that they are equal. The hash values may be used to quickly compare subtrees of the directory for equality. Two hashes are maintained so that it can quickly be determined whether the difference in two sub-trees is due do a difference in the root node of the two sub-trees (main hashes differ, but child hashes are equal) or a difference in their children (main hashes and child hashes differ).

B. Synchronization Service

The design of the synchronization service requires that several goals be achieved. At the service's base, network interactions must be handled as efficiently as possible. A large portion of the work within the service involves interacting with other hosts in the ad hoc group. Meanwhile, the service must provide a reasonable top-level interface to local applications that desire to interact with the shared data-space.

Making efficient use of the available wireless bandwidth is a difficult task, given that there is not a designated server to handle all requests. As each host, being peer-to-peer, acts as both a client and a server, there is a risk that a request from one host will result in a barrage of replies from every host within the current ad hoc group. The service has been designed specifically to accommodate adaptation as messages from other hosts are seen on the network. This is achieved by maintaining large in-application queues and aggressively optimizing their contents.

The overall structure of the service is shown in Figure 2. The network interactions section of the diagram handles synchronization efforts over the network. The implementation uses three queues: inbound, scheduling, and outbound.

The inbound queue contains messages that have been received from the network but have not yet been processed by the protocol engine. Outbound messages from the protocol engine

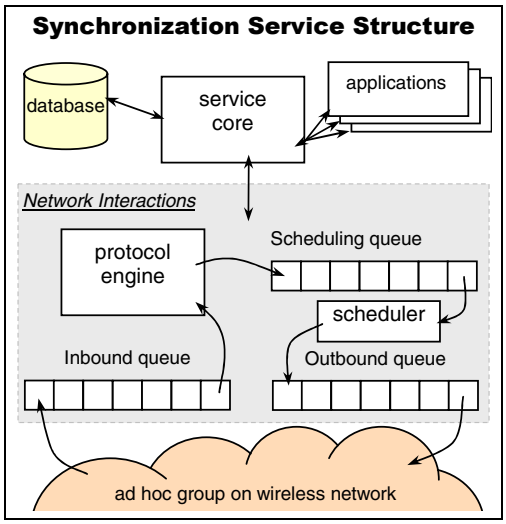

Fig. 2. Synchronization Service Structure

are placed into the scheduling queue and assigned transmission delay values. As delay values expire, messages are moved from the scheduling queue to the outbound queue for transmission on the network.

The scheduler for outbound messages allows time to advance for items on the scheduling queue only when the inbound and outbound queues are both empty. As the processing on inbound messages often results in the removal of messages from the scheduling queue, this avoids the transmission of stale and redundant messages to other hosts.

C. Synchronization Protocol

The synchronization protocol is responsible for communicating directory divergences between members of the group. As groups may contain any number of members and members may enter and leave the group at any time without warning, the protocol must require a minimal coupling between hosts.

In order to achieve this, the synchronization protocol has a single message type. A message contains one or more challenges, where a challenge is simply an assertion regarding the contents of a single node in the tree. All hosts periodically send a challenge containing their tree's root node. As the hash values in the root node describe the entire tree, trees with different contents are almost guaranteed to have different root hash values.

When a challenge is received, two actions are taken. First, the value of the challenge node is compared to the value of the corresponding node in the local tree. If the node values are different, the timestamps are compared and the newer value is selected for the local tree. If the local node possesses the newer value, it is scheduled for transmission in order to update the host that sent the stale value. If the local tree's value is replaced, hash values are recalculated from that node back up to the root, reflecting the change across the tree.

Once the node values have been compared, the node hashes are compared to ensure that the children of the two versions of the tree are consistent. If the hashes do not match exactly, the immediate children of the node are sent as challenges. In this manner, challenges quickly descend through the tree and divergent nodes are advertised to other hosts in the ad hoc group.

As the protocol is broadcast-based, each host in the group receives every challenge message that is sent. In order to prevent the entire group from replying at once, outbound messages are scheduled with a random delay. This delay is based on the last update time of a given node; hosts with more up-to-date values will tend to respond more quickly. This delay also allows hosts to avoid redundant broadcasts by removing messages from the scheduling queue as duplicate (or newer) node messages are received.

D. Service API

The service provides a simple interface for applications. This interface is composed of four primary methods: *getDirectoryContents*, which returns a read-only copy of the contents of the directory; *setNodeValue*, which adds the given node to the directory; *addListener* which attaches the specified function to the service to receive notifications of updates; and *removeListener*, which removes the specified listener function from the notification list.

When an application adds a node or modifies the value of a node, that node is immediately transmitted to the ad hoc group. In this manner, updates are incorporated within the connected hosts of the ad hoc group very rapidly, without necessitating their discovery through hash value comparison between hosts.

V. Current State of the Synchronization Service

The synchronization service has gone through three major overhauls since its inception and has provided a core infrastructure component for our collaborative classroom application. A full description of the evolution and performance improvements over the past two years is beyond the scope of this paper and will be left to a future work. In summary though, we have found that the broadcast-based replication mechanism is very sensitive to even apparently small changes in the protocol.

The service has realized marked performance improvement since it was first written and currently runs stably in ad hoc networks with upwards of one hundred wireless participants.

VI. Comparison to Other Systems

As noted in the introduction, there are a large number of systems that address the issue of data replication and distribution. We believe that the characteristics that distinguish our work are: shared read and write access to the data, the peer-to-peer nature of the protocol, the use of broadcast to optimize synchronization, and the low overhead associated with its implementation. We present here a selection of existing systems that provide data replication and note how they differ from ours. Note here that we are examining the current set of peer-to-peer research efforts which allow data to be updated or modified. These applications are markedly different from popular peer-to-peer file sharing applications, for which data

is generally immutable, coarse-grained, and not synchronized across the entire membership [6].

Distributed Hash Tables such as Tapestry [7] and Chord [8] attempt to spread data over a set of nodes to improve access latencies while taking advantage of group resources. These systems depend on a stable membership and incur large costs for membership changes. DHTs attempt to spread data across a set of nodes while maintaining fast access, while we replicate data on each node to survive a constantly changing membership.

LDAP and related systems [9] provide a hierarchical data store that bears some resemblance to our directory. However, these systems typically follow a more traditional approach to availability, replicating central servers and using transactional mechanisms [10] to ensure consistency.

The Bayou [11], [12] project is quite similar in goal to our own work. While the protocol is peer-to-peer in nature, there are elements of the conflict resolution protocol that require a master, although the master can be dynamically selected. In addition, Bayou requires hosts to keep more complex state information than does our protocol. Bayou messages must be retained to enable conflict resolution. Finally, Bayou is not broadcast-based, and so is less suited to data synchronization within an ad hoc group.

The Clique [13] project at HP Labs Grenoble has aimed to build a peer-to-peer replicated file system and its approach bears similarity to our own. Clique is different from the work described here in several ways: their one-to-many messaging is based on IP multicast rather than wireless broadcast, differences between nodes in their system are detected using a high-level metadata journal rather than a tree of hashes, and our directory is much more akin to a distributed data structure than a file system; operations in our system are of a considerably finer granularity.

Finally, LIME [14] (Linda in Mobile Environments) shares the goal of providing a data-structure-based middleware for ad hoc computing. Their system is tuple-based and specifically addresses location by, for instance, allowing large portions of a shared data structure to be available only when the mobile client is in a specific physical location. While we are very excited about the idea of a data-structure based middleware for collaborative applications, we feel that our approach may be more pragmatic, at least in the domain of specific applications.

VII. Future Work

We have identified several interesting areas of extension to the existing work. We would like to further improve the efficiency of the directory, perhaps by focusing synchronization efforts on more recently modified data. We are exploring the possibility of introducing a weak transactional semantic where an interdependent set of nodes under a common parent will not be made visible until all are present. An area of particular interest to the authors lies in expanding the existing service to allow applications to specify their interest in specific subtrees of the global directory. Finally, the directory could be expanded to provide an event-based communication system. As updates to the tree are synchronized with other nodes very quickly, hosts may be able to use it as a publish-and-subscribe event system for asynchronous communications in ad hoc groups.

VIII. Conclusions

Our efforts to build a collaborative application targeted for ad hoc wireless networks identified the need to replicate data across ad hoc groups. Traditional approaches to replication are not well suited to this type of environment due to their reliance on master/slave architectures and transactions. Furthermore, as ad hoc groups form and dissolve over time, replicated data must be carried from its original source to other hosts for which a direct communication path may never exist.

These high-level requirements were transformed into a rigorously defined data structure and accompanying specifications of the expected semantics of its replication both within an ad hoc group and across multiple ad hoc groups.

An implementation of this data structure and the synchronization protocol is currently being used as part of the fundamental infrastructure of our collaborative application. In that capacity, it is performing very well, and satisfies the application's need to identify the presence of participants, share data amongst participants, and configure the set of applications that constitute the ad hoc group.

Our experience in building this one type of collaborative application suggests that the type of data sharing required is not unique to this application, but is, in fact, common to all applications that are intended to support collaboration in dynamic environments.

References

[1] A. Black, N. Hutchinson, E. Jul, and H. Levy, "Object structure in the Emerald system," in *Conference proceedings on Object-oriented Programming Systems, Languages and Applications*. ACM Press, 1986, pp. 78–86.

[2] E. D. Lazowska, H. M. Levy, G. T. Almes, M. J. Fischer, R. J. Fowler, and S. C. Vestal, "The architecture of the Eden system," in *Proceedings of the eighth ACM Symposium on Operating Systems Principles*. ACM Press, 1981, pp. 148–159.

[3] E. Levy and A. Silberschatz, "Distributed file systems: concepts and examples," *ACM Computing Surveys*, vol. 22, no. 4, pp. 321–374, 1990.

[4] B. Liskov, "Distributed programming in Argus," *Communications of the ACM*, vol. 31, no. 3, pp. 300–312, 1988.

[5] P. D. O'Brien, D. C. Halbert, and M. F. Kilian, "The Trellis programming environment," in *Conference proceedings on Object-oriented Programming Systems, Languages and Applications*, ser. ACM SIGPLAN Notices, vol. 22, no. 12. ACM Press, 1987, pp. 91–102.

[6] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," in *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*. ACM Press, 2003, pp. 314–329.

[7] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," UC Berkeley, Tech. Rep. UCB/CSD-01-1141, April 2001.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM Press, 2001, pp. 149–160.

[9] T. Howes and M. Smith, *RFC1823: The LDAP Application Program Interface*. IETF Network Working Group, August 1995.

[10] J. Gray, "Notes on database operating systems," in *Operating Systems: An Advanced Course*, ser. Lecture Notes in Computer Science, R. Bayer, R. Graham, and G. Seegmuller, Eds. Springer-Verlag, 1978, vol. 60, pp. 393–481.

[11] A. J. Demers, K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and B. B. Welch, "The Bayou architecture: Support for data sharing among mobile users," in *Proceedings IEEE Workshop on Mobile Computing Systems & Applications*, Santa Cruz, California, 8-9 1994, pp. 2–7.

[12] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer, "Bayou: replicated database services for world-wide applications," in *Proceedings of the 7th workshop on ACM SIGOPS European workshop*. ACM Press, 1996, pp. 275–280.

[13] B. Richard, D. M. Nioclais, and D. Chalon, "Clique: A transparent, peer-to-peer replicated file system," in *Proceedings of the 4th International Conference on Mobile Data Management*, ser. Lecture Notes in Computer Science, vol. 2574. Springer-Verlag, 2003, pp. 351–355.

[14] A. L. Murphy, G. P. Picco, and G. Roman, "LIME: A middleware for physical and logical mobility," in *Proceedings of the 21st International Conference on Distributed Computing Systems*. Phoenix, AZ, USA: IEEE Computer Society, April 16-19 2001, pp. 524–233.