

Enabling Co-located Learning over Mobile Ad Hoc P2P with LightPeers

Bent Guldbjerg Christensen, Mads Darø Kristensen, Frank Allan Hansen, Niels Olof Bouvin Center for Interactive Spaces, ISIS Katrinebjerg
Computer Science Department, Aarhus University, Denmark
Email: {bentor, madsk, fah, bouvin}@daimi.au.dk

Abstract— This paper presents LightPeers – a new mobile P2P framework specifically tailored for use in a nomadic learning environment. A set of key requirements for the framework is identified based on nomadic learning, and these requirements are used as outset for designing and implementing the architecture and protocols. The main contributions of the LightPeers framework are: a mobile P2P framework including a specialized robust messaging protocol resilient to changes in the network topology, an developers API, and a suite of LightPeers applications supporting nomadic learning prototyping key features of LightPeers.

Index Terms— Mobile Peer-to-Peer, Contingency Handling, Multi-user Interaction, Ad-Hoc Networking, Nomadic Learning

I. MOTIVATION

Personal devices such as mobile phones, PDAs, MP3 players, and laptops are widely used for everyday life at work, at home, or at school. Mobile phones are ubiquitous among the younger generation, and this has met with some resistance in educational settings, where use of mobile phones is routinely restricted or outright banned.

The Danish Ministry of Education listed in their vision for learning in elementary schools in 2010 [1] a number of desirable qualities for pupils to acquire, e.g., navigating in increasingly heterogeneous sources of information, collaboration and fellowship, participation and responsibility, and problem solving and knowledge sharing.

Rather than banning mobile phone use, we suggest that pupils should be encouraged to take advantage of their mobile devices in order to enable this vision for 2010.

The following section describes how a resilient mobile ad hoc P2P networking infrastructure could support project oriented group work at an elementary school. This leads on to a formalized user model and a set of requirements for the network architecture and protocols. In Section II the LightPeers framework is presented as such an ad hoc P2P network infrastructure. A number of applications supporting nomadic learning have been developed using this framework, and these are presented in Section III. This is followed by an evaluation of the framework and applications in Section IV. Related work is discussed in Section V, in Section VI future work is outlined, and the paper is concluded in Section VII.

A. Scenario

A group of pupils has been tasked with documenting changes that have undergone their town in the past 100 years, and equipped with a tablet PC and some (possibly their own) camera phones. They venture forth in the town and interview people and photograph landmarks. Using their devices, they can coordinate their actions and keep a diary, as well as share pictures and other multimedia. These activities take place in a session to facilitate shared awareness. Using the session, the group can split up (moving out of network range) and reconvene, whereupon shared state (e.g., notes in the chat application, or photos taken) are automatically distributed and synchronized across the devices in the session. Upon return to their school, the group adds a PC to the session, thus copying all their collected data to it, and use it to create a presentation.

B. User Model and Requirements

This scenario illustrates the fundamental user model of nomadic learning, where pupils equipped with mobile devices can move independently around indoor at the school or outdoor on field trips and enter into groups in an ad hoc manner to share information with each other. The term nomadic learning is used as introduced in [2], where nomadic learning describes pupils (“nomads”) being in transit between many physical places (“oases”) such as classrooms, libraries, museums, urban or natural areas, or at home.

Groups engaged in nomadic learning are typically project groups of four to six pupils, but smaller groups of, e.g., two friends or larger groups of an entire class of 30 pupils are also possible.

Formalizing the user model, a peer is a LightPeers enabled device which in the scenario corresponds to the tablet PC or camera phones used by the pupils to gather, organize, and present information. Sessions are used to divide peers into groups such as described in the scenario and provide environments for robust communication among members.

In Figure 1 an example of three peers with a number of LightPeers applications participating in two sessions S1 and S2 is depicted.

Implementing the formalized user model for nomadic learning requires a resilient mobile network architecture

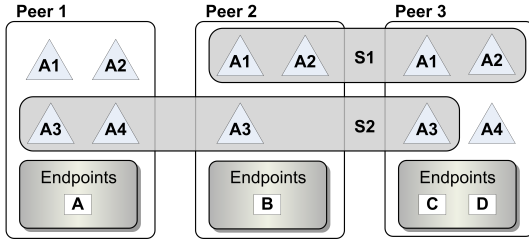


Fig. 1. An example of three peers with a number of LightPeers applications participating in two sessions S1 and S2.

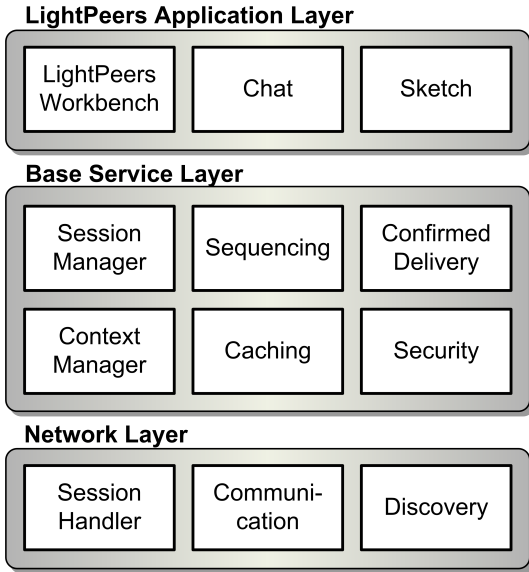


Fig. 2. The three layers of the LightPeers Architecture: Network Layer, Base Service Layer, and Application Layer.

allowing limited devices to enter and leave the network dynamically, while providing robust messaging and sharing of digital material among peers in a session.

How these requirements were approached is described in the following sections.

II. LIGHTPEERS

The LightPeers framework consists of a layered architecture of software components and basic protocols for a peer participating in a LightPeers network.

A. Architecture

The LightPeers architecture, shown in Figure 2, comprises three software layers: the *Network Layer*, the *Base Service Layer*, and the *Application Layer*, each containing a number of components.

The *Network* layer handles basic communication between peers as well as basic P2P services such as discovery. The *Communication* component is responsible for sending and receiving all packets for a peer through the available communication media: e.g., WLAN or ZigBee.

The next layer of the LightPeers framework is the *Base Services*. This layer handles only session data, i.e., packets that belong to a session that the local peer is

participating in. In the *Base Service* layer the *Session Manager* component maintains the state of sessions that the peer is participating in. The state of a session includes the participating peers, the applications registered for the session on each participating peer, the communication features utilized in the session etc. Based on session state information, the *Session Manager* component dispatches incoming packets to the appropriate application in the layer above. The dispatch chain can include a number of communication features such as the ones offered by the *Security*, *Caching*, *Sequencing*, and *Robust Delivery* components. If such features are in use, the *Session Manager* dispatches the messages to these before delivering them to the *Application* layer.

On top of the *Base Service* layer is the *Application* layer. This is where the actual LightPeers applications exist, and an *Application* layer API is provided offering developers convenient access to the functionality of *Base Service* layer components. This API provides functionality to manage sessions and to communicate with other peers over the LightPeers network. For some applications the session management should be an integrated part of the application user interface; this is the case with applications such as multiplayer games. But this approach is not very suitable for a project work session with numerous applications, such as a chat application, a photo share application, a vote application, and more, that must work together within the same session. Therefore, a simple LightPeers Workbench application is bundled with the API providing the users with basic session managing functionality.

B. Routing

The session concept presents some novel ways of thinking about communication in mobile ad hoc networks (MANETs).

In the usage scenarios that LightPeers is designed to support, the peers of a session may shift between being in range and out of range.

The sessions are overlay networks that are each perfectly capable of handling all the intra-session routing. Confining routing of session data to stay within the session preserve already scarce resources such as bandwidth and energy—especially when doing broadcasts. Distributing data within a session is called sessioncasting which is the predominant way of communicating in LightPeers. Sessioncasting is realised as a variant of Epidemic Routing obtain transitive transmission within peers in a session. As described by Vahdat *et al.* [3], Epidemic Routing is used for distributing application messages amongs hosts, called carriers, within connected partitions of ad hoc networks.

C. Protocols

The LightPeers framework is designed to work on top of the link layer of the OSI protocol stack, provided that the link layer supports frame broadcast, e.g., WLAN or ZigBee.

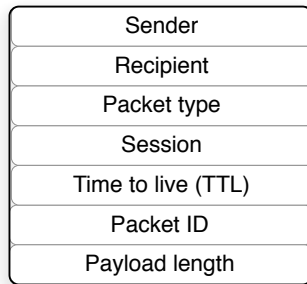


Fig. 3. The base (network layer) header of a LightPeers packet.

As described in Section II-A, LightPeers is divided into independent layers. The *Network* layer is the lowest layer—the layer that corresponds to the network layer of the OSI model. This layer has three main responsibilities illustrated by the three components it consists of: communication, discovery, and session handling.

To be able to perform these duties the *Network* layer expects the packets sent in the MANET to have a certain structure, i.e., the packets need to include a standard LightPeers header as shown in Figure 3.

The header contains the sender and recipient addresses, which are obviously needed when routing packets through the network. A LightPeers peer is identified by the MAC address of its main network interface.

The next field is the packet type which contains one of the following packet types: PING, SESSION SEQ MARKER, SESSION UPDATE, JOIN REQ, RESEND REQ, and DATA:

A PING packet is periodically emitted by all peers, but not forwarded. These packets are sent by the *Discovery* component to enable the discovery of neighboring peers. The payload of a PING packet is a list of sessions that the sending peer is participating in. This payload is used by the *Session* components.

The SESSION SEQ MARKER packet is sent to session peers nearby, but not forwarded and contains the current vector stamp of packets received from other peers in the session. This information is used by the *Robust Delivery* component to update session peers coming back into network range.

The SESSION UPDATE packet contains information about the session such as newly joined peers or peers that have left, and also which applications each peer participate with in the session. This packet type is sent out whenever a change in the session occurs.

A JOIN REQ packet is used when a peer is requesting to join an existing session. The packet is sent to an existing member which then will sessioncast a SESSION UPDATE packet if the request is approved.

RESEND REQ packets are used as part of the Robust Delivery communication feature to request a packet that was lost in transmission.

DATA packets are only sent within sessions. These packets contain any data that should be passed on to the *Service* layer and possibly further up the stack to the

Application layer. Such packets often have more headers after the base header such as a session header and an application header.

After the packet types comes the UUID¹ of the session that the packet belongs to.

The time-to-live (TTL) field specifies how many hops the packet should be forwarded. This field will usually have a very low value as sessions are not expected to span a large number of hops.

The packet ID is a sequence number increased by the peer each time a packet is sent. The pair (sender, packet ID) thus uniquely identifies a packet in the MANET. The unique ID of individual packets is needed when broadcasting packets to ensure that a packet is not forwarded twice.

The final field in the header is the payload length that specifies the length of the data above the header.

A number of other headers are used at the next layer of the protocol stack. All messages sent within a session have a session header containing a message sequence number. These sequence numbers are individual, i.e., it takes a (sender, seq. no.)-pair to uniquely identify a message within a session. The sequence number is initially set to zero when a new session is created or joined, and each time a message is sent within the session, the sequence number is incremented and stamped on the message. Another important header is the application header, which does the job of directing messages to the appropriate application on a peer. The application header consists of two entries: an application ID uniquely identifying the application, and an optional MIME type that describes the type of the content. This information is needed to be able to serve the data to the correct application. If the specified application is not available, another application capable of working with the specified MIME type can be invoked.

D. Caching of session data

The caching feature is one of the most important parts of the LightPeers design. Using caching of session data is what facilitates peers to be out of network coverage for a period of time, and then get updated when they are within range of the other peers again.

As with any other communication feature, caching can be offered on a per session or per application basis; and it is the responsibility of the *Session Manager* to apply the caching service to the appropriate messages. But, in contrast to the other communication features, the caching service needs not run on all peers in the session. Only a few resource strong peers are chosen to maintain the cache, and these peers collect the data and pro-actively keep all other nodes updated.

The responsibilities of the caching peers are twofold: Firstly, they must collect and store any data that belongs to the session or the specified set of applications within the session; Secondly, they must be able to serve the cached

¹Universally Unique Identifier as defined in RFC 4122.

data to peers that have for some reason not been kept up to date.

Caching is realized by recording the sequence numbers of messages sent by each individual peer in the session. Every peer registers the sequence number of the last message it has sent to the session together with the last sequence number seen from other peers in the session. These values constitute a vector stamp that shows the peer's current view of the in-session communication.

A caching peer can not be expected to have unlimited storage available for the caching service. The cached data is therefore discarded in FIFO order when the cache occupies more than a fixed amount of storage. This amount can be set either as a configuration parameter globally for the LightPeers framework, or individually when the caching service is started in a specific session.

Serving the data is done in two ways: re-actively, where a peer asks the cache for a specific set of messages, and pro-actively, where the caching peer reacts to input from the context manager. When pro-active, the caching service will listen to discovery-events from the context manager, alerting it to session peers entering the vicinity of the caching peer. When that happens, the caching peer contacts the newly seen peer, sending it a vector stamp showing what the cache contains. If the caching peer has some messages unknown to the new peer, it may request these messages from the cache.

E. Sequencing

Another useful feature in a communication channel is sequencing. This feature tries to deliver all packets in the session in the correct order. It is important to notice that this does not impose a strict ordering on all messages in the channel, only on messages sent from a specific peer.

Sequencing can be done in a number of ways and with varying strictness. The sequencing done in the LightPeers framework is probably the least strict way of doing sequencing; a best effort sequencing. When sequencing is enabled, all messages belonging to the sequenced channel are handed to the sequencer before being passed along to the application layer. When the sequencer receives a message, it checks the sequence number of the message and takes one of the following actions:

- If the sequence number is exactly one larger than the previously known sequence number of a message from that peer, it is delivered immediately.
- If the sequence number is more than one larger it is buffered for a configurable amount of time before being delivered, and any messages that arrive within this period of time are either buffered or delivered according to their position in the sequence.
- If the sequence number is smaller than the last sequence number one of two things may be done:
 - If there are some buffered messages, the message may be buffered or delivered immediately depending on its sequence number.
 - If no buffered messages exist, the message will be delivered out of order. Optionally, the

sequencer can be configured to discard such messages

F. Robust Delivery

Robust delivery provides a semi reliable communication channel. This should not be confused with complete reliability since it does not guarantee that all messages are delivered to all peers within a session. What the Robust Delivery communication feature guarantees is, that a message is delivered to all peers within the current *session segment*, i.e., the message is received by all peers within sessioncasting range.

Robust Delivery uses an optimistic retransmission scheme in the sense that the sending peer does not expect acknowledgments from the receiving peer if every packet was received properly and in order. Only a *missing* packet will make the receiving peer requests it resent with a negative acknowledgement.

The Robust Delivery makes use of both the Sequencing and Caching communication features since a missing packet is discovered if a packet is received out of sequence. The out of sequence packet is stored by the Caching feature and a RESEND REQ packet for the missing packet is sent.

In the two cases where either the last packet in a sequence is lost or a session peer has been out of network range and comes back, the receiving peer cannot determine that packets are missing. Therefore, SESSION SEQ MARKER packets are periodically broad-casted from each peer to inform neighboring session peers of the sequence numbers of sent and received session packets. So, when a peer comes back into network range it receives a SESSION SEQ MARKER and the Robust delivery communication feature will then begin requesting the missing packets.

A RESEND REQ for a missing packet can be answered by any session peer in range having the missing packet cached and otherwise the request is forwarded.

In this way Robust delivery is delay tolerant and capable of working in highly ad hoc network topologies while providing a best effort delivery of sessioncasted messages, using retransmissions to counter the effects of packet collisions and other packet loss. For examples of reliable multicast algorithms, see [4], [5]. Such multicast algorithms could be used in LightPeers, representing sessions as multicast groups. Such an approach has been rejected for a number of reasons, primarily to keep the framework simple, and secondarily because these approaches create a large amount of traffic in the network wasting valuable resources.

III. PROTOTYPE IMPLEMENTATIONS

A prototype implementation of the LightPeers framework has been developed in C# for the .NET CF 2.0 platform and is currently running on the Fujitsu-Siemens Pocket Loox 720, Tablet PCs with Windows XP, ordinary laptops with Windows XP, as well as Linux on Mono

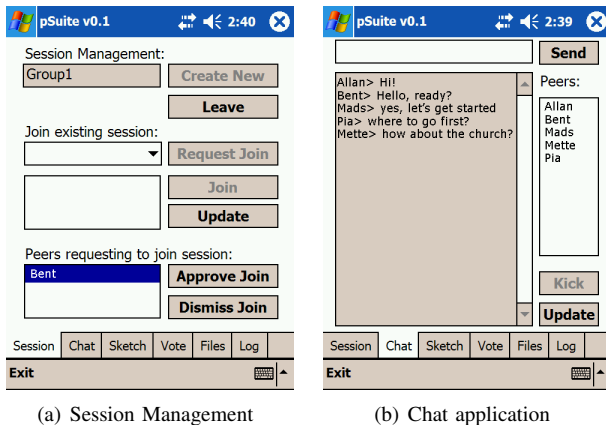


Fig. 4. The Session Management provides mechanisms for self-moderating groups and the Chat application supports simple text messaging between session peers.

and Macs with Mac OS X.² The current prototype uses WLAN and the UDP protocol for broadcasting, however the framework design is independent of the IP protocol.

The most interesting part of the LightPeers framework is not the framework in itself, but rather the applications running on top of it and the novel facilities provided by the applications to the user. The communication between LightPeers and the applications is socket based, so from an application developer's perspective the access to LightPeers is provided through the LightPeers API library that handles: application registration, session management, sending and reception of application packets, and more. This division of framework and applications is deliberately made to avoid constraining the application developers to a certain language and programming environment, but only having the support for socket communication as the only requirement for utilizing LightPeers. In this section a small suite of basic LightPeers applications is presented.

A. Session Management

Session Management is an important part of every application in LightPeers since it provides means to: create, join, and leave sessions as seen in Figure 4(a). Furthermore, Session Management allow existing session peers to invite and expel other peers. In order for a peer to join an existing session, a request to join is sent to one of the existing session members, who can then dismiss or approve the request. The approval can be performed manually or automatically where the latter corresponds to an open session. If the new peer is approved, the approving peer session-casts a session update including information about the newly joined peer. As peers in a session are required to only process packets from approved peers, the approval mechanism bar peers from joining at will. Protecting the approval mechanism from malicious peers, e.g., changing the sender address of its packets to that of an existing session peer, is a challenge

²The framework and applications can be downloaded from: <http://www.daimi.au.dk/~bentor/LightPeers/>

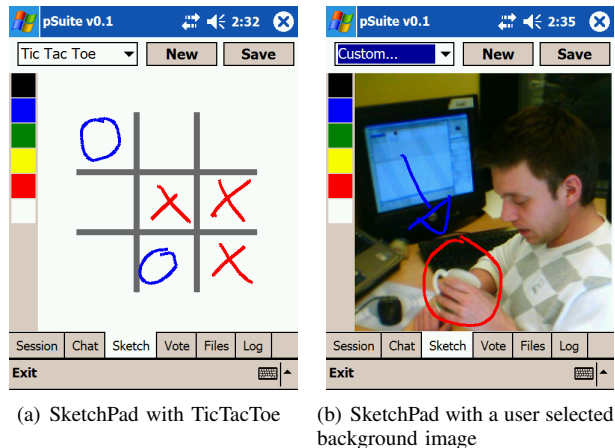


Fig. 5. The SketchPad application supports collaboratively sketching out ideas on a shared sketchpad, playing simple board games such as Tic Tac Toe, or letting the user choose an arbitrary image to be annotated.

for the security component. Existing session peers can also be expelled from the session if one or more of the other session members decides to do so and issues a session update. Thus, LightPeers provides an infrastructure for ad hoc group forming and self-moderation, and does not per se restrict the user to conform with certain social rules, but rather relies on the users to bring their social understanding and behaviour from their everyday life into their use of LightPeers emphasizing fellowship, participation, and responsibility.

In the JXTA P2P framework [6] peer groups can have a membership policy ranging from open to protected where certain peer credentials are required, which is similar to the Session Management mechanisms in LightPeers, however JXTA peer groups do not directly have support for expelling and inviting of peers.

B. Chat

A fundamental application of networked devices is sending text messages between users, e.g., SMS messages between mobile phone users. The Chat application provides a shared communication channel which can be used for exchanging messages or function as a shared log for session activities similar to the functionality of Internet Relay Chat (IRC) networks³. The Chat application implemented on LightPeers can be seen in Figure 4(b).

C. Sketchpad

A step further than simple text messaging is to have the equivalent of a sketchpad for freehand drawing to allow sketching figures or concepts not easily described in text. The Sketchpad application as shown in Figure 5(a) supports exactly this situation where a session peer can make a freehand drawing which is simultaneously shared among all session peers. All peers are allowed to draw on the sketchpad and annotate the figures concurrently supporting collaborative construction of digital material.

³The IRC RFC (1459).

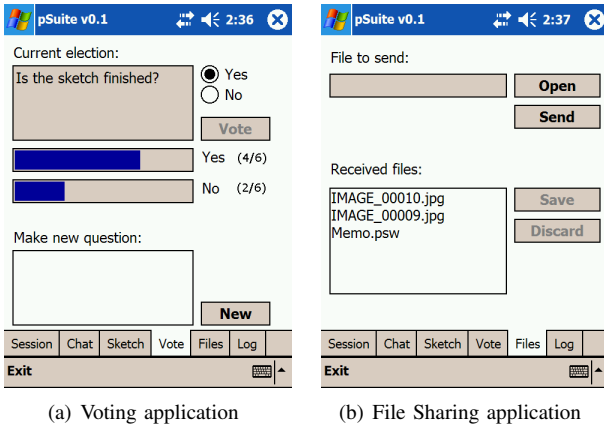


Fig. 6. The Voting application provides a tool for decision making among peers in a session and the FileSharing application supports simple sharing of files.

After sketching ideas out the content of the sketchpad can be saved for future use.

The Sketchpad has a series of standard background layouts, e.g., blackboard or whiteboard, but also game boards like Tic Tac Toe is included with the application. The users can also create and apply custom backgrounds, for instance by taking a picture and let it be the new sketchpad background to be collaboratively annotated as seen in Figure 5(b). The custom background is transferred to the other peers which allow the sketchpad application to be used as a platform for simple multiplayer board games as well.

Support for multi-user shared real-time environments including freehand drawing can be encountered in earlier (CSCW) systems, e.g., DOLPHIN [7] connecting remote workplaces and providing a shared work environment. However, co-located multi-user construction of digital materials on mobile devices as supported by LightPeers is still a novel research area.

D. Voting

In a group of people working or playing together questions can occur that requires decisions to be made on account of the whole group. This situation could be solved by, e.g., electing one person to be in charge or by taken a vote among the group members. The Vote application as shown in Figure 6(a) gives session peers the opportunity to conduct a simple election among the session members. For instance, the Vote application could be used in the situation of deciding on if a peer should be approved for joining the session or not, or if the shared picture is good enough for the project assignment.

E. File Sharing

Traditional P2P applications are maybe best-known for providing file sharing among peers, e.g., Napster and Gnutella [8] [9], file sharing is not the main focus for LightPeers, but the functionality is nevertheless useful for the application domain intended for LightPeers, e.g.,

sharing sound or video recordings of interviews made by pupils on field trips. The FileSharing application seen in Figure 6(b) provides the core functionality of sharing files between session members.

IV. EVALUATION

The LightPeers framework is specifically designed to support applications for nomadic learning. Even though it is useful to evaluate and compare frameworks with respect to the raw performance of their architectures and protocols, e.g., typical P2P quality attributes such as scalability, reliability, connection bandwidth, peer discovery, node addressing, and so on [10], we have chosen to focus on evaluating how well the design of LightPeers supports development and execution of applications for nomadic learning. Thus, in our evaluation of the framework we have not only measured its performance, but also considered how well the framework and its applications could support tasks within the use-domain, e.g., pupils joining and leaving groups, and producing, sharing and annotating material on a range of diverse mobile devices.

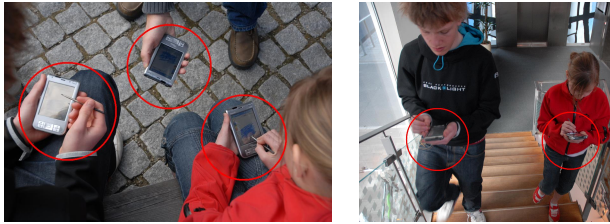
In order to make these assessments we have found the following evaluation criteria, originally proposed for evaluating pervasive computing architectures [11], quite useful: *completeness*, *complexity*, *performance*, and *utility*. Completeness is an assessment of whether the framework and architecture are powerful and flexible enough to support development of interesting programs, i.e., is the provided set of framework primitives complete enough to support different applications? Complexity refers to how hard it is to write code and develop applications with the framework. Ideally, the framework should make it easier to develop and deploy nomadic learning applications, than developing these application from scratch. On the other hand, a large and complex framework will typically have a steep learning curve, and if too complex, it may eventually be abandoned in favor of a simpler one. Performance refers to the system performance of applications implemented with the framework. The question is whether or not the architecture performs well enough to support real workloads of actual applications. Utility describes the general usefulness of the framework. This criteria determines whether others can built applications with the framework.

A. Completeness

A number of different applications have been built using framework primitives to support nomadic learning as presented in Section III. The applications include: Chat, Vote, SketchPad, FileShare, Ping, and Session Manager.

B. Complexity

The framework provides an API for program developers encapsulating the framework primitives into high-level language classes and methods. Besides session managing the core functionality involves basic sending and receiving application data packets. Bundled with the API is a set



(a) Creating a new session

(b) On the road

Fig. 7. Pupils in the field using LightPeers for nomadic learning

of example applications and a skeleton application implementing the Session Management to serve as starting points for developers. The API, the skeleton application, and the minimal conceptual model make the learning curve for framework deployment less steep compared to more complex frameworks. However, implementing corresponding applications on top of LightPeers and other related frameworks, e.g., JXTA, has not yet been formally evaluated.

C. Performance

A series of simple quantitative performance test were conducted to measure the latency between peers in a session. A Ping application was developed using the LightPeers API, so the measurements would be on application data being sent among session peers. The Ping application allows the user to choose which session peer to ping. The payload of the Ping packet contains a timestamp from the sending peer which is send back from the destination peer in a pong packet.

For a single Ping packet the Ping times were $41.8ms$ for 1 hop and $80.7ms$ for 2 hops which corresponds to a latency of $\sim 21ms$ for a single packet to be transmitted from an application through the framework and received on the other peer by the framework and delivered to the end application. The Ping times are averaged over 10 measurements.

A usability test of the LightPeers framework and applications was performed with pupils given a group assignment that involved using various aspects of the framework. In Figure 7(a) three pupils are creating a new session for the group work in the field and sketching out a todo list on how to approach the assignment. After agreeing on the approach they split up with two members entering an office building to make an interview (seen in Figure 7(b)) while the third member was waiting outside making plans for their next stop. The third member in the session successfully received the interview material as the session members joined up again.

Feedback from the usability test included new ideas for simple multiplayer games, as well as queries as to getting LightPeers on their own personal mobile phones.

D. Utility

So far, the application developers have also been involved in developing the core framework, thus already

having an extended insight in the intentions of the API and example applications. The framework is planned to be used by students in a course on P2P networks as the basis for their own projects. This will provide an evaluation of the framework from an application developer's perspective.

V. RELATED WORK

In [12] Yang *et al.* present ASOS a self-organized network of nodes to support distributed and reliable storage overlay for end-to-end data flows under disruption. The general delay-tolerant network approach of ASOS is similar to that of LightPeers with the sessioncasting mechanism, however with ASOS the objective is to support delivery between two peers whereas the objective with LightPeers is to support delivery from one peer to a group of session peers. ASOS supports advanced probabilistic data management among ASOS peers compared to LightPeers where peers are replicating as much session data as their cache store allow. The Session Management in LightPeers supports dynamically creating sessions and allowing peers to join and leave, whereas the ASOS group of peers are pre-configured with a multicast address.

Section I presented an educational setting as a use case for the LightPeers system. There are other systems for learning in the field, notable the Ambient Wood [13], where children explore a woodland environment by using handheld devices and interacting with sensors and speakers. A number of systems have been developed to explore mobile P2P networking, among them Proem [14] and JXME [15].

In [14], Gerd Kortuem *et al.* make a persuasive argument for the applications of portable P2P systems, as well as the challenges inherent in building such systems. The Proem system is an open system based on three core protocols for presence establishment, data exchange, and community building, all utilizing the same connectionless, asynchronous transport protocol. Based on these core protocols, a sophisticated event driven architecture featuring "peerlets" running in a peerlet engine is built, enabling users to collaborate and share applications across devices. As the basic Proem transport protocol is XML based, the whole architecture is language agnostic, although the current system is Java based.

The JXTA project⁴ has in the past years created a comprehensive framework for the development of P2P systems. While predominately Java-based, the specification of the framework is based on the exchange of XML messages, and a number of ports to other languages are available. The JXTA framework is a rich and elaborate framework, that supports the development of many types of P2P applications, but of special interest in this context is the JXTA Micro Edition (or JXME), which is an implementation for J2ME. While not as feature rich as other implementations, the system has recently progressed nicely, and can now operate without relying on proxies

⁴<http://www.jxta.org/>

on more capable peers. This new addition is a significant change and removes a major obstacle in a general adoption of JXME.

VI. FUTURE WORK

As described in Section II-C LightPeers assume that peers are able to broadcast frames on the link layer. Peers that do not have this broadcast ability, such as peers that connect through a Bluetooth device, must cooperate with stronger peers who are running a Bluetooth bridging service. Through this bridging service, Bluetooth devices gain the ability to broadcast packets in the network. Designing and building this bridging service is a highly prioritized step in the future design of LightPeers.

One communication feature not described in detail in this paper is the *Security* feature. This feature has not yet been implemented, but is planned to be using the Diffie-Hellman [16] key exchange protocol. When a new peer enters a session, it receives a shared secret key from one of the existing session peers using the Diffie-Hellman protocol. When each session peer has the secret key all subsequent messages can be encrypted using this key.

The LightPeers framework is currently implemented in C# running on .NET CF 2.0 and Mono, but a Python version of the framework is planned as future work to support a greater variety of platforms such as mobile phones, e.g., the Nokia N80.

VII. CONCLUSION

We have presented the LightPeers framework. A user model grounded in a nomadic learning scenario was developed and formalized. From the user model some key requirements for the framework were identified. The requirements derived from the scenario and user model for LightPeers resulted in the following contributions:

a) *a three-layered architecture portable to a variety of platforms*: currently the prototype is running on a variety of mobile devices.

b) *a robust messaging mechanism between peers in a session*: the session concept and sessioncast mechanism provide an overlay network among session peers and a robust delivery protocol, and

c) *a number of applications for nomadic learning*: with the LightPeers API developers have access to session management mechanisms and handling transmission of application data.

The work on LightPeers continues with implementing secure sessions where all communication within a session is encrypted. More performance evaluation and optimization is planned and finally, exploring application domains beside nomadic learning, e.g., mobile urban gaming for testing and evaluating the LightPeers framework is an exciting step in front of us.

REFERENCES

[1] U. T. D. M. of Education), "Vision 2010 - en udviklingsamtale med skolen," 2000, ISBN 87-603-1708-6, Undervisningsministeriets forlag.

[2] C. Brodersen, B. G. Christensen, K. Grønbaek, C. Dindler, and B. Sundararajah, "ebag: a ubiquitous web infrastructure for nomadic learning," in *WWW '05: Proceedings of the 14th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2005, pp. 298–306.

[3] A. Vahdat and D. Becker, "Epidemic routing for partially connected ad hoc networks," 2000. [Online]. Available: citeseer.ist.psu.edu/vahdat00epidemic.html

[4] S. K. S. Gupta and P. K. Srimani, "An adaptive protocol for reliable multicast in mobile multi-hop radio networks," in *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA '99. Second IEEE Workshop on*, 1999, pp. 111–122. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=749283

[5] T. Gopalsamy, M. Singhal, D. Panda, and P. Sadayappan, "A reliable multicast algorithm for mobile ad hoc networks," in *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, 2002, pp. 563–570. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1022306

[6] Bondolo, Jaltman, Tra, and Yeager, "JXTA v2.0 Protocol Specification," 2004, <http://spec.jxta.org>.

[7] N. A. Streitz, J. Geißler, J. M. Haake, and J. Hol, "Dolphin: integrated meeting support across local and remote desktop environments and liveboards," in *Proceedings of the conference on Computer Supported Cooperative Work*, Chapel Hill, USA, Oct. 1994, pp. 345–358.

[8] S. Fanning, "Napster," 1999, <http://www.napster.com>.

[9] J. Frankel and T. Pepper, "Gnutella," 2002, <http://www.gnutella.com>.

[10] J. Walkerdine, L. Melville, and I. Sommerville, "Dependability properties of p2p architectures," in *P2P '02: Proceedings of the Second International Conference on Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, 2002, p. 173.

[11] R. Grimm, "One.world: Experiences with a pervasive computing architecture," *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 22–30, 2004.

[12] G. Yang, L.-J. Chen, T. Sun, B. Zhou, and M. Gerla, "Ad-hoc storage overlay system (asos): A delay-tolerant approach in manets," in *Mobile Adhoc and Sensor Systems (MASS), 2006 IEEE International Conference*.

[13] M. J. Weal, D. T. Michaelides, M. K. Thompson, and D. C. DeRoure, "The ambient wood journals: replaying the experience," in *Proceedings of the 14th ACM Hypertext Conference*, L. Carr and L. Hardman, Eds. Nottingham, UK: ACM Press, Aug. 2003, pp. 20–27.

[14] G. Kortuem, J. Schneider, D. Preuit, T. G. C. Thompson, S. Fickas, and Z. Segall, "When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad hoc networks," in *P2P '01: Proceedings of the First International Conference on Peer-to-Peer Computing (P2P'01)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 75.

[15] Hamada, Kuldeep, and Tra, "JXTA-J2ME 2.0," 2005, <http://jxme.jxta.org/>.

[16] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644–654, 1976. [Online]. Available: <http://citeseer.ist.psu.edu/340126.html>

Bent Guldbjerg Christensen is currently a Ph.D. candidate in the Center for Interactive Spaces, ISIS Katrinebjerg, at Aarhus University, Denmark. He received his MS degree in computer science from Aarhus University, Denmark in 2003.

Mads Darø Kristensen is currently a Ph.D. candidate at Aarhus University, Denmark. He received his MS degree in computer science from Aarhus University, Denmark in 2006.

Allan Hansen is Post. Doc. at Aarhus University. He received his PhD and MS degrees in computer science from Aarhus University in 2006 and 2003, respectively.

Niels Olof Bouvin is Associate Professor at Aarhus University. He received his PhD and MS degrees in computer science from Aarhus University in 2001 and 1999, respectively.