May 2003

Peer-To-Peer Backup for Personal Area Networks

Boon Thau Loo
University of Pennsylvania, boonloo@cis.upenn.edu

Anthony LaMarca
Intel Seattle Research

Gaetano Borriello
Intel Seattle Research

Peer-To-Peer Backup for Personal Area Networks

Abstract

FlashBack is a peer-to-peer backup algorithm designed for power-constrained devices running in a personal area network (PAN). Backups are performed transparently as local updates initiate the spread of backup data among a subset of the currently available peers. Flashback limits power usage by avoiding flooding and keeping small neighbor sets. Flashback has also been designed to utilize powered infrastructure when possible to further extend device lifetime. We propose our architecture and algorithms, and present initial experimental results that illustrate FlashBack's performance characteristics

Comments

Peer-To-Peer Backup for Personal Area Networks

Boon Thau Loo, University of California, Berkeley
Anthony LaMarca, Gaetano Borriello, Intel Research Seattle

IRS-TR-02-015

May 2003

Peer-To-Peer Backup for Personal Area Networks

Boon Thau Loo
UC Berkeley
421 Soda Hall
Berkeley, CA 94720
(510) 524-5821
boonloo@cs.berkeley.edu

Anthony LaMarca
Intel Seattle Research
1100 NE 45th Street, 6th Floor
Seattle, WA 98105
(206) 633-6555
lamarca@intel-research.net

Gaetano Borriello
Intel Seattle Research
1100 NE 45th Street, 6th Floor
Seattle, WA 98105
(206) 633-6555
borriello@intel-research.net

ABSTRACT

FlashBack is a peer-to-peer backup algorithm designed for power-constrained devices running in a personal area network (PAN). Backups are performed transparently as local updates initiate the spread of backup data among a subset of the currently available peers. Flashback limits power usage by avoiding flooding and keeping small neighbor sets. Flashback has also been designed to utilize powered infrastructure when possible to further extend device lifetime. We propose our architecture and algorithms, and present initial experimental results that illustrate FlashBack's performance characteristics.

1. INTRODUCTION

Trends in the size, speed and cost of computing elements will soon make it feasible to embed computation in practically all manufactured devices. Moreover, the growing popularity of networking technologies like Bluetooth will make it possible to inexpensively add ad-hoc wireless networking capabilities to these devices as well. The result is that the small electronic devices we traditionally carry with us such as key fobs, wrist watches, cell phones and MP3 players, as well as larger devices such as PDAs and laptops will in effect form a pool of trusted, personal networked components. Recent efforts have shown that low-cost, easy-to-build, wireless "personal area networks" or PANs are already feasible [7]. These PANs will soon cease to be the domain of exotic wearable computing rigs, and will instead become a mainstream reality.

While personal area networks hold considerable promise, the heterogeneity, limited power, and frequent disconnection of devices in the PAN environment makes it difficult to develop robust applications. This problem can be greatly reduced by middleware services tailored to the constraints of mobile computing in general and PANs in particular. We introduce *FlashBack*, a solution designed specifically to provide reliable storage for the self-managing, mobile, impoverished devices that will be found in PANs. FlashBack is peer-to-peer, with each device maintaining a limited set of preferred neighbors. Each device in FlashBack allocates a portion of its storage and power to backup the data of its peer devices. Backups are made to nearby devices keeping both cost and battery use at a minimum. FlashBack was designed with the following goals in mind:

- **Power-Efficiency** – Perhaps the biggest single factor influencing the design of middleware for PANs is power-efficiency. Every element of FlashBack has been designed with power-efficiency in mind and communication between devices is minimized as much as possible. FlashBack uses lazy update propagation [8] to reduce the number of updates that need to be sent. FlashBack keeps small neighbor sets minimize the maintenance overhead. In addition to using power sparingly, FlashBack has per-device power budgets to provide explicit control over how much devices can be utilized.
- **Heterogeneity Aware** – In a PAN there may be extreme heterogeneity among the participating devices; PAN may contain a wristwatch with a microcontroller and a 900MHz radio cooperating with PDAs and laptops with 802.11 capabilities. This heterogeneity is both a potential pitfall and an opportunity. To avoid potential starvation or overload FlashBack communicates a device's resources and capabilities to their neighbors and this information is used in our group formation and replication algorithms. In the event that an extremely capable device is present, such as fixed infrastructure (server class devices without power constraints), FlashBack will use this to the maximum advantage, both replicating to the device as well as using it as a proxy to push replicas for less capable devices.
- **Ease of Use** – The most common complaint about computing today is that it is difficult to configure and manage even simple systems. PANs potentially make this worse, introducing a large number of interface-constrained devices. We designed FlashBack to add as little overhead as possible on both users and application developers. FlashBack does not require any explicit actions to invoke data replications. Rather, FlashBack monitors the local data updates and transparently replicates data. In addition, FlashBack's employs very simple replication semantics known as *master data ownership* described in [8]. Devices replicate data for each other, but are unaware of the structure or semantics of the data they are holding for their peers. These replicas are merely used as safeguards for the host device against data-losses. This simple model greatly simplifies the replication, coherency and security mechanisms in our system. While some PAN applications may want to do read and write sharing across devices, we believe that most do not, nor does FlashBack prevent coherent sharing of data by a higher-level data management service.

The contributions of this paper are twofold. Broadly, it offers what we believe is the first investigation into the feasibility of providing peer-to-peer backup services to a collection of power-constrained devices. Specific, we have developed FlashBack, an easily implemented algorithm for doing power-aware backup across a dynamic collection of trusted, heterogeneous devices.

2. FLASHBACK

FlashBack has been designed with a broad and dynamic definition of a personal area network. We believe that users will own a collection of small, trusted computing and communication-enabled devices, some subset of which they have with them at any given

time. In addition, users will also regularly interact with a number of other trusted devices such as car telematics, home automation systems, and traditional desktop computers. As users go through their daily lives, the set of devices in their immediate environment will change as they move from home to car to work and back. We refer to this dynamic collection of devices as the user's PAN. We believe that this dynamic notion accurately reflects the mobile computing context in which users will operate in the future.

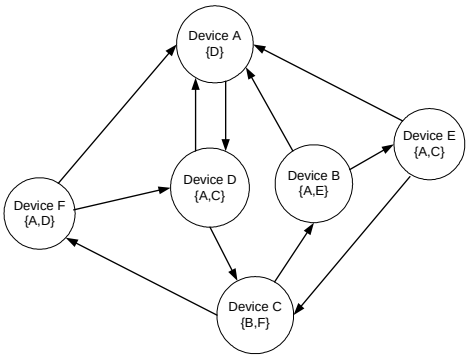

Figure 1: Overview of a PAN: Each device is labeled with its unique device identifier, and has its neighbor set in parenthesis. Note that neighbor links are directional.

We begin our discussion of FlashBack by presenting an overview of its architecture, and then describe its algorithms and initial implementation. We assume that devices within a user's PAN are *trusted*, and a lightweight certificate-based authentication scheme is used to ensure that devices only participate in PANs to which they are assigned. We also assume that point-to-point communication is taking place between devices in the PAN. While the presence of intermediaries performing ad-hoc routing would not affect the functionality of FlashBack, it would affect its efficiency since FlashBack assumes all communications take one hop.

In FlashBack, each device sets aside a portion of its local storage to devote to storing replicas for other devices. Each device also reserves a portion of its power for FlashBack activities such as sending and receiving replicas and status messages. When the power budget is exceeded, the device no longer participates, causing it to effectively leave the PAN from FlashBack's perspective. In FlashBack, each device has a *device identifier* that uniquely identifies itself within its PAN. This device identifier is assigned out-of-band during the installation of FlashBack on the device. If a device fails, its device identifier can be assigned to a new device, and the new device is introduced to the PAN to commence the recovery process discussed in section 2.3.

Each FlashBack device maintains a *neighbor set* (see Figure 1) within the PAN. This neighbor set is chosen based on heuristics discussed in section 2.1, and represents the preferred set of devices to which replicas are pushed. A device with an empty set of neighbors can discover an initial set of neighbors by snooping the network for FlashBack traffic. This initial set of neighbors is refined by occasionally sharing neighbor status messages with its current neighbor set.

The unit of replication in FlashBack is a variable-sized block of unstructured data. Since there is no sharing of data between devices, the replicating devices need not understand the data's format or semantics; rather they treat the blocks as opaque data that may be requested at some future time. This gives devices the opportunity to encrypt data before it is passed to FlashBack, ensuring privacy. It also makes Flashback flexible, allowing it to work with any storage model that can be translated into block reads and writes. In our initial implementation, we layered a *tuple-store* on top on FlashBack, where each block of data holds a tuple. File systems and databases would also be good choices provided care is taken to keep block sizes reasonable. FlashBack provides a simple API with four basic procedure (See Figure 2).

```
DataBlock ← Read (GUID)
GUID ← Write(DataBlock, Priority)
Delete (GUID)
Recover()
```

Figure 2: FlashBack API. GUID uniquely identifies a block of data within its local store.

This API allows for data manipulation (*read*, *write* and *delete* function calls) in data-block granularity. The number of replicas created for each block of data vary based on the *replication factor* that depends on the priority level set by the application creating the data. Higher priority data will be replicated more aggressively. In the event that a device fails or is lost, a new device is introduced to replace the former and starts the recovery process via the *recover* function call.

2.1 PAN Formation and Maintenance

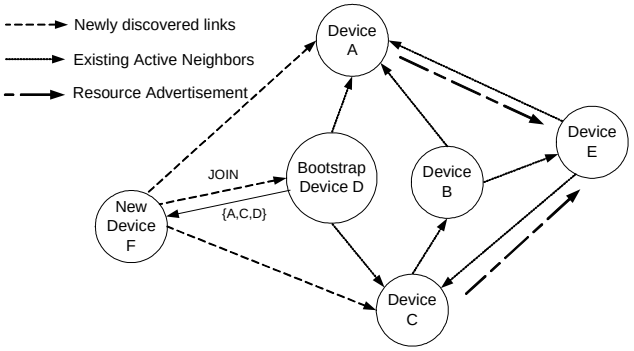

Figure 3: Bootstrapping and PAN Formation. F joins the PAN by contacting D. D responds with a set of recommended neighbors that include its current neighbor set and itself.

To join a PAN, new devices must identify an existing device that is currently part of the PAN. Locating this bootstrap device can be done in two ways. First, a device discovery protocol such as UPNP [21] can be used identify fellow devices in the PAN. Alternately, the wireless network can be snooped for FlashBack traffic, enabling the new device to discovery other active devices. This latter approach works especially well when a device is joining an established PAN with a number of active devices.

Once a bootstrap device is identified, a *Join* message is sent from the new device to the bootstrap device. Figure 3 shows an example where F sends a Join message to D. The Join message includes the device identifier as well as a *PAN certificate*. The PAN certificate is simply an identifier that uniquely identifies the PAN in which a device is participating. The bootstrap device will accept the new device into the PAN only if it has an identical identifier. This simple authentication scheme ensures that devices only participate in their designated PANs.

Upon receiving the Join message and authenticating the new device into the PAN, the bootstrap device returns a list of neighbors including itself to the new device, together with their advertised resources. In Figure 3, D returns the recommended neighbors {A,C,D} together with their advertised metrics. The new

device will select a subset from the recommended peers to be its initial neighbors. Note that neighbor links are directional. E.g., D considers C its neighbor but not vice versa.

Devices will periodically send keep-alive messages to their neighbors who respond with their currently available power and storage resources. Requesting another device to be a neighbor does not require a special message since it is implicit from the first keep-alive message being sent. The lack of a response alerts a device that a neighbor may have departed and its entry is then deleted from its neighbor set.

2.1.1 Utility Function

Ideally, a device should choose neighbors that have sufficient storage and power resources to support all the device's replicas, and have a high probability of being in the device's proximity if it should require recovery. *FlashBack* was written to accept a parameterized utility function that approximates this notion of a good neighbor. The higher the computed utility value, the more a device is valued as a neighbor.

Given that the main resource constraint is power, a simple utility function based solely on a neighbor's available power should provide decent overall behavior. In Section 3 we show that while performing reasonably well, a utility function based only on power fails to take advantage of either co-locality or the benefits offered by devices with large amounts of available storage. Hence, we also include a more balanced utility function that blends the following factors:

- **Available Power** – Devices with more power available have a higher chance of lasting longer in the PAN, and can support more replicas for other devices within the PAN.
- **Available Storage** – Devices with more storage available are favored due to the maintenance efficiency offered by replicating many data blocks on a single device and batching keep-alive messages. To the same effect, if a device already has some replicas on a neighbor, that neighbor should be positively favored as well.
- **Pair-wise Locality** – This final factor keeps track of whether two devices are frequently active in the same PAN at the same time. This metric is especially useful in tracking if devices tend to "cluster" together, such as embedded devices within a car or a briefcase. We estimate pair-wise locality by counting the number of keep-alive messages exchanged with each device over a moving window of time.

- **PowerBudget(RD) :** The advertised current power budget of RD.
- **StorageFactor(RD) :** The ratio of the current storage size of RD allocated for the PAN, to the total size of data in LD.
- **ReplicaFactor(RD) :** 1 + (the ratio of the size of replicas stored in RD on behalf of LD, to the total size of data in LD)
- **LocalityFactor(RD) :** $(2 - e^{-numMsgs})$, where numMsgs is the number of keep-alive messages exchanged with LD.

Figure 4: Elements of the Balanced Utility Function. The balanced utility function is a product of above 4 properties of a remote device. LD stands for Local Device, while RD stands for Remote Device.

Given that the utility function has to be positively correlated with all the factors above, the actual function that we have in our implementation is the product of the properties of the remote device as shown in Figure 4.

2.1.2 Neighbor Selection

The *aggregate utility* of a device is the sum of all the utility values of all its current neighbors. In order to keep the FlashBack maintenance costs low, the maximum number of neighbors N_{max} is limited to the number of replicas that will be created for any block of data.

Periodically, each device will contact one of its neighbors randomly and request neighbor recommendations. That neighbor will return its current set of neighbors to the requestor. Based on the new recommendations, a device may choose to add new neighbors (at the expense of its low utility neighbors if N_{max} is reached). This process of improving the neighbors is known as *rediscovery*.

Performing rediscovery too frequently will drain the resources of the devices, while performing rediscovery too infrequently would mean that information about resource-rich devices will be propagated too slowly. In FlashBack, we use a simple adaptive approach in which a device with few neighbors will perform rediscovery more aggressively (based on a simple linear function) to ensure that it can build up to the maximum set of neighbors quickly.

2.2 Replicas

Periodically, FlashBack identifies locally written data and pushes their replicas lazily into the PAN. The process operates as follows: the device replicating the data (called the *local device*) sends a *replicate* message containing the data to be replicated to one of its neighbors *(*called the *remote device)*. A greedy algorithm is used in selecting the remote device for replication. Among the neighbors with sufficient resources, the highest utility neighbor is picked as the candidate remote device to push each replica to. If no suitable neighbor is found among the neighbor set, replication is simply deferred. Upon receiving a replica from a neighbor, the remote device stores the replica, and returns an acknowledgment to the local device.

To keep track of its replicas, FlashBack maintains, in persistent storage, a *replica table* that has an entry for each local data block. Each entry in the replica table maintains a list of *replica states* for each replica created. The replica state contains a 2-bit status (Active, Inactive, or Expired), the identifier of the device that currently stores the replica as well as a time at which the replica was created. A replica state is added to the list whenever the local device receives an acknowledgement from the remote device about the successful creation of a replica.

2.2.1 Multiple Replicas for Redundancy

Replication can be performed multiple times for redundancy, and the replication factor is decided on a per-data-block basis based on the priority assigned by the application at the time of the write. Having redundant replicas improves reliability and increases the probability that a replica is present in the PAN during the recovery process. FlashBack maps each priority level to a replication factor.

FlashBack provides soft reliability guarantees as follows: at any point of time, the number of replicas for each data block is at most its replication factor. Periodically, FlashBack scans the replica table to identify data blocks where the designated replication factor is not reached, and creates replicas for these blocks. Creation of new replicas is tuned to be lazier (based on a linear scale) as the number of replicas approaches the replication factor. This achieves a damping effect on the creation of replicas, and places higher priority on data blocks that currently have no replicas.

2.2.2 Replica State Management

FlashBack manages replicas using a time-based management scheme. When a new replica is created, a commitment is made by the remote device to store the replica for a *lease* period. During this period, the local device assumes that the replica is available unless the remote device fails. The lease of any replica can be in any of the three states seen in Figure 5.

A newly created replica is always *active*. The local device will periodically send keep-alive messages to the remote device holding the replicas. A replica transitions from active to *inactive* state during the *disconnection* phase, when the remote device storing the replicas no longer responds to keep-alive messages sent by the device. Devices that have disconnected and left the PAN leave with their replicas intact. Flashback avoids being too greedy in making more replicas when a neighbor exits a PAN, as such aggressive replication will not work well in a resource-constrained environment where devices could be entering and leaving frequently and network connectivity cannot be guaranteed. Inactive replicas are retained on devices that depart the PAN, and will 'decay' over time due to the lack of updates reaching them. However, their replicas are still considered useful for future recovery purposes as the device may rejoin the PAN, and will be kept for the duration of the lease period.

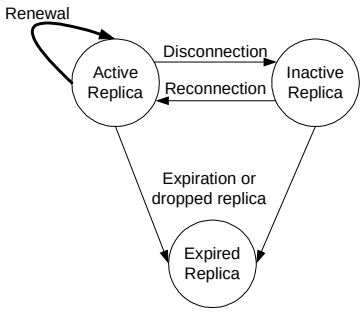

Figure 5: State Diagram for Replica Management

A replica is reactivated during the *reconnection* phase, when two devices detect the presence of each other within the PAN. In the *expired* state, the lease has expired and the remote device can drop the data and reclaim space for new data.

Replica renewal is done periodically by sending a *renew* message to the remote device storing the replica. When half the lifetime of a replica has transpired, a device will attempt to renew the duration of the replica on the remote device. If a replica's lease cannot be renewed be due to disconnection, it expires and the replica can be dropped. The owner of the original data is aware of the expiration and will attempt to place a new replica elsewhere. In the event that a replica is valid but expired, the replica's lease can be renewed upon reconnection.

The lease duration should be longer than any planned disconnection to ensure that replicas are not dropped during normal usage patterns. Long lease durations also increases power efficiency by reducing the number of messages sent to extend leases. Setting lease duration to be too large, however, make the system less responsive by increasing the time taken to move replicas to resource rich devices.

2.2.3 Deletes and Updates

Deletes are performed by sending a message to the respective devices in the PAN to drop the block. If a replica is unavailable at delete time, it will eventually be dropped when the lease on the replica expires. Updates are performed as a lazy deletion of the old block and lazy creation of a new block. Periodically, each device scans the replica table for potential updates. A replica is due for an update when the timestamp of the base data is newer than the timestamp of the replica. Active replicas are updated simply by sending the new data to replace the existing replicas. This guarantees that whenever possible, replicas are updated in their former locations and allows the possibility of sending deltas. At the same time, new blocks are created lazily in the background to ensure that there are sufficient copies of the new block. When inactive replicas are reactivated during the reconnection phase, if there are sufficient up-to-date replicas, the inactive replicas will be dropped. Otherwise, these replicas are reactivated and updates propagated.

2.3 Recovery Process

During the recovery process, a new device is introduced into the PAN to replace a failed device. It initially sends a *recovery* message to its immediate neighbors who will then flood the message throughout the PAN. While flooding is power inefficient, recoveries should be rare and represent a situation in which power drain is not a central concern. The flooded recovery message contains the unique device identifier of the device it is replacing, as well as the network address of the new device. The recovery message is buffered in devices for time T_{REC} and gossiped to new neighbors who have not seen the message before. A longer T_{REC} will ensure better recall as devices are allowed to recover their replicas for a longer time. However, this means that many devices will have to buffer recovery messages and propagate this information to new devices for a longer time, increasing the PAN overhead.

When a device receives a recovery message, it scans its local storage and sends all the relevant replicas that it has directly to the recovering device. Since there can be multiple data blocks with different timestamps, the recovering device retains the newest ones. An open issue yet to be resolved is when a device can resume its applications during the recovery process.

2.4 Initial Implementation

To test the power efficiency and reliability of FlashBack we have implemented a Java version that performs FlashBack as we have described. In our implementation, the network layer was abstracted to allow for simulations as well as actual deployment. The experimental results in Section 3 were generated using the simulated network layer.

3. EXPERIMENTS

This section describes a series of simulation experiments designed to explore the performance and reliability characteristics of FlashBack. In Section 1, we listed power-efficiency and best effort reliability in the face of heterogeneity and dynamicity as our design goals, and we show how well we have done in each of these areas.

3.1 Experimental Setup

In all of the experiments, we are running an implementation of FlashBack using a simulated network layer. In all cases, we are modeling a fixed set of devices, all of which are running FlashBack, and analyzing the behavior of the PAN. All data blocks are written with the same priority and therefore have the same replication factor.

The simulated network layer uses a simple, but realistic communication and cost model. In our network model, each device can send messages to other devices within its PAN in one network hop. In current wireless devices, the cost of sending and receiving on the network is much more expensive that the cost of accessing

memory or performing computation. Recent measurements show that the cost of sending a bit on an 802.11a network to be 1000 times more power than performing an add on the Compaq Personal Server [2]. Since FlashBack is neither compute, no memory intensive, we ignore all costs other than network sends and receives. It is import to note that we are only modeling the overhead introduced by FlashBack, and not by the applications running on behalf of the user. This allows us to compare the impact that allowing FlashBack to performing backups has on the device's time-to-live. The cost of receiving relative to sending varies from technology to technology, with some costing more to send than to receive and vice-versa for the others. The measurements in [2] also showed the energy used to transmit and receive a bit in a Compaq Personal Server running 802.11b to be approximately equal. In our simulated network, the cost per byte for both sending and receiving is set to 1 power unit. All messages sent and received within the PAN are included in the cost calculations.

3.1.1 Device Configuration

Device Type	Percent Devices	Average Storage Space (MB)	Average Power Budget
I	70%	64	300k
II	10%	128	800k
III	10%	256	1M
IV	10%	512	2M

Figure 6: Initial Device Configuration

Our experiments were conducted with a collection of devices with varying resources (See Figure 6). The bulk of the devices (Type I) have the smallest storage capacity and power budget. Types II-III represent the minority of the devices and have increasingly more storage and power budget. These devices are intended to span the range from cell phone and 2-way pager class devices up to PDAs. The configurations used for these initial experiments do not capture any real-world configuration we measured. Rather they are intended as a simple starting point we are using to understand how our design tradeoffs have manifested themselves in both power usage and reliability/recall.

At the start of each experiment, each device is initialized with the storage and power parameters stated above. 50% of the storage resources will be devoted to FlashBack. Each device is given a power budget that is not replenished during the duration of the experiments.

Each device is initialized with an initial amount of data and is periodically fed with a stochastic workload. Each data block written is 1KB in size. The workload is a mixture of 50% writes and 50% updates, and is Poisson-distributed with a mean-time-between-writes of 1 hour. Reads are ignored, as they have no effect on our measurements.

Replicas are created lazily, ranging from 0.5 to 2 hours between the creations of new replicas. Periodic neighbor rediscovery ranges from 3 minutes (when it has no neighbors) to 1 hour (when it tries to improve its full set of neighbors). Lease durations were set to 480 hours.

All of the experiments were run using three different utility functions: *random*, *power-only* and *balanced*. The random utility function values all neighbors equally, causing devices to choose neighbors randomly. The power-only utility function causes devices to value a neighbor higher if it has more available power. The balanced utility function (described in Section 2.1.1) causes devices to blend available power, storage and co-location when valuing neighbors.

3.1.2 Experimental Metrics

We use two metrics to measure the performance of FlashBack. *Median Time-To-Live (TTL)* is the median time when devices run out of their power budget and is our primary metric of power consumption. The higher the Median TTL is, the more efficiently and evenly Flashback is using the power resource available. *Recall* is the percentage of data that can be recovered at any instant in time from the PAN. This is our metric for the reliability achieved by FlashBack. Recall is measured by freezing the PAN hourly and examining the data stored in each device currently active in the PAN, and figuring out if a corresponding replica is present in another device within the PAN.

3.2 Replication Factor and Power Usage

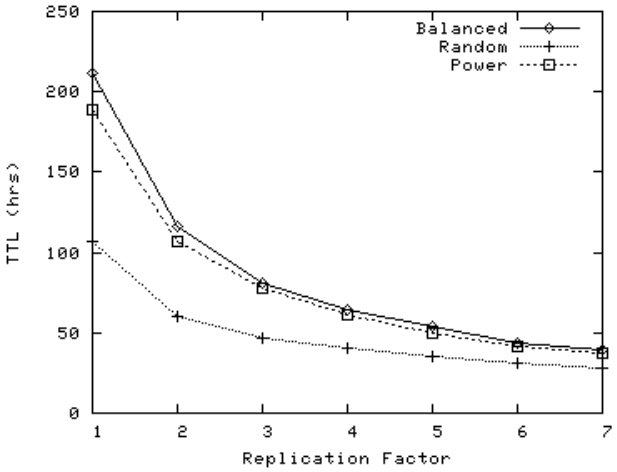

Figure 7: Median TTL (hrs) against Replication Factor

In our first experiment, we measure the effects of increasing the replication factor on the lifetime of a PAN. In this experiment, we introduce a collection of 30 devices chosen from the distribution in Figure 6. These devices all start running their workload of writes and updates at time 0 and continue doing so until all of the devices have exhausted their FlashBack power budget. In this experiment the PAN is static and none of the device departs until they run out of power. Figure 7 shows the Median TTL for all three utility functions varying the replication factor from 1 to 7. (Recall in section 3 that the replication factor refers to the number of replicas created per data block.)

As expected, this graph shows that increasing the replication factor shortens the lifetime of the PAN. This graph also shows that the TTL is much higher for the power-only and balanced utility function than the random utility function. This demonstrates that both are effectively utilizing the devices with more power in order to extend the lives of the less capable devices. The balanced utility function yields a median TTL around 10% larger than power-only. This is due to two factors. First, power-only causes the device with the most power to be valued above all other and produces a degree of churn as devices switch places in the ranking of their available power. Second, the balanced utility results in a device placing all of its replicas on a small number of peers, lowering the number of maintenance messages that need to be sent. Power-only does not have this affordance and will spread replicas out over a large number of peers as the power rankings change.

Figure 8 shows another view of the same experiment by focusing on a configuration with replication factor is 4, and measuring the device drainage. During the experiment, we logged the time when each device exhausted their power budget. In this graph, the Y-Axis shows the number of devices remaining in the PAN. For both power-only and balanced, effective use of the more capable devices extends the lifetime of the weaker devices. Hence, all the devices leave the PAN at approximately the same time. On the other hand, the random utility function drains each device equally causing the impoverished devices to exit the PAN first, followed much later by the more capable devices.

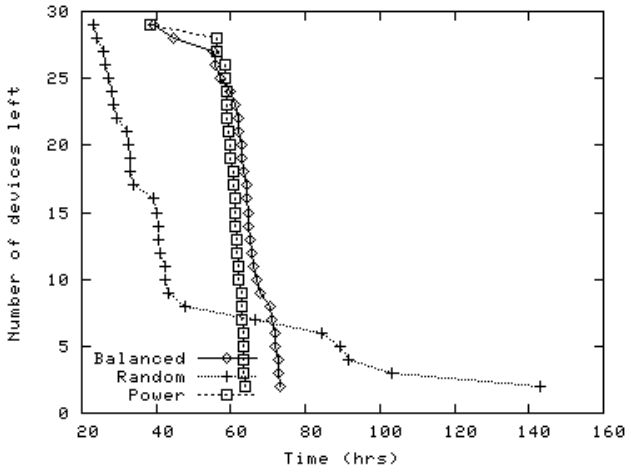

Figure 8: Device Drainage Over Time Replication Factor = 4.

3.3 Disconnection and Recall

Our second experiment illustrates the importance of placing replicas on devices that are frequently co-located. In our second experiment we configure the PAN with the same 30 devices running the same workload as the previous experiment. In this case 15 of the devices are mobile and periodically disconnect and reconnect from the other 15 non-mobile devices. The mobile devices move independently of each other and the disconnections and reconnections are Poisson distributed with a mean time of 4 hours before disconnection and 8 hours before reconnection. We then measure the recall of the 15 devices that remain within the PAN.

Utility Function	% Recall
Balanced	98.6 %
Power-only	64.6 %

Figure 9: Comparing Recall for Different Utility Functions When Devices Can Disconnect

Figure 9 shows the resulting recall for the balanced and power-only utility functions when the replication factor is 1. This table shows that the blended utility function does a much better job of ensuring that the devices that always remain in the PAN replicate data among themselves. This is not a surprising result given that the blended function includes a factor that represents the pair-wise co-locality of devices. Nevertheless, this does show that to achieve high recall, such a factor is likely needed.

3.4 Replication Factor and Recall

In our final experiment, we look at how the replication factor affects the recall and age of recovered data in a dynamic PAN. To some extent this may not actually be a terribly important issue. If a device fails and recovery needs to be performed, a user is probably willing to take the time to gather their devices in one place to perform a complete recovery. It may be the case, however, that with a reasonable replication factor, recall and recovery age can be sufficiently good to make this unnecessary. To measure the effect that increasing the replication factor has on recall we configure the PAN so that all of the 30 devices join and leave the PAN at random intervals while executing their workloads. Each device has an average PAN uptime and downtime, which represent the amount time in which they stay active in the PAN and are not in the PAN, respectively. Devices that have left the PAN continue to be subjected to their workload. When disconnected, devices are alone and are not in proximity to any other devices. To change the volatility of the PAN, the portion of time spent away from the PAN was varied between 20% and 50%. This was achieved by using an average downtime of 2 hours and average uptimes of 2, 4 and 8 hours. We measured the recall for every device periodically until the first time a device has exhausted its power budget and left the PAN permanently.

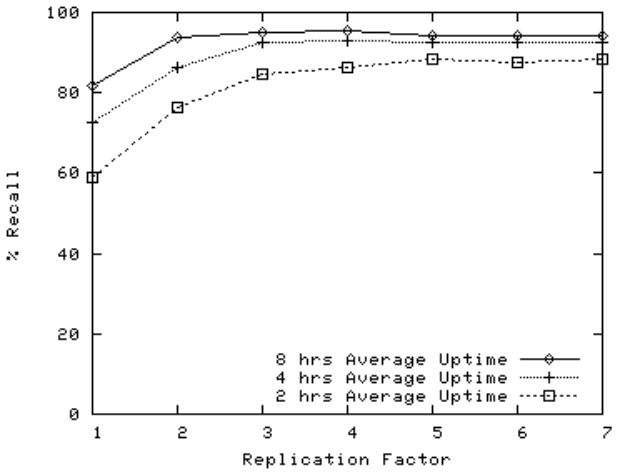

Figure 10: The Effect of Replication Factor on Recall

Figure 10 shows the impact that changing the replication factor has on recall. This graph shows that replication factor actually has less of an effect on recall that volatility does. The curve for uptime=2 hours shows considerably worse recall that the more stable configurations. All of the curves show that beyond a replication factor of 2, there is little additional recall to be gained. This suggests that using a modest replication factor of 2 or 3 and carefully identifying co-located devices is the best strategy for achieving high recall. It is worth noting that the highest recall reached by the most stable configuration is still well under 100%. This is largely due to the fact the FlashBack is replicating lazily and even when the PAN is stable some loss can occur due to data that has yet to be replicated.

4. RELATED WORK

Most backup mechanisms for mobile devices are what we call *occasional synchronization systems*. Users move around their environment updating state in the device, and then using local communication mechanisms like USB, infrared or Bluetooth, run a synchronization protocol such as SynchML [19] to synchronize the device's data with a server. In some cases, the synchronization is automatic; in most cases, it is manual. While there protocols are easy to understand and offer basic data sharing, they have the significant drawback that more often than not, the user is wandering around with data that is not backed up. Solutions involving automatic synchronization usually involve a specific

desktop machine (at home or work) that is configured for synchronization. When users are away from this special machine for long periods of time their data is vulnerable.

A competing backup model is what we call *continuous background synchronization* in which it is assumed that one of the user's devices will always be one hop from the fixed infrastructure. All devices are then configured to use this connection to run a background synchronization process. A common gateway device for this sort of scenario is a GPRS cell phone. In these solutions, devices use local networking technologies like Bluetooth to buffer new data on the cell phone, which then sends the data to a designated backup server using the cell phone's data communication capabilities. This solution offers the benefit that the data is backed up as it is generated without any extra local infrastructure and without intervention on the part of the user. The downside is that this synchronization typically incurs a financial cost proportional to the amount of data sent. In addition, despite the promises of ubiquitous connectivity, there are lapses in coverage, exposing the user to potential data loss. Finally, connections of this type are bandwidth limited and may not reasonably be expected to synchronize the large amounts of data produced and consumed by all the devices with a PAN.

Outside the mobile computing space, a number of P2P backup systems have been developed. Both Pastiche [4] and HiveCache [9] perform backup services for workstations by utilizing free disk space on other machines. Unlike FlashBack, these systems have not been designed with power efficiency in mind. The design of these systems has also been heavily influenced by the fact that a large percent of the data (like executables and libraries) are identical from machine to machine. It is unclear if these same assumptions apply to the varied devices found in a personal area network.

There are a variety of general-purpose data management systems that support disconnected operation and are therefore potentially deployable within PANs. Bayou [20] is a system that supports replication in a weakly consistent fashion, thus enabling the sharing of data between mobile users operating in disconnected modes. Bayou utilizes *update-anywhere* replication and leverages application-specific rules to transparently help deal with reconciliations. While Bayou offers a more flexible and powerful data sharing model than FlashBack, Bayou requires substantial application modification and was not been designed with power efficiency in mind. Coda [16] is an update-anywhere file system that allows users to read and write cached files while disconnected. Unlike systems like Bayou that make the applications manage conflicts, Coda has the users themselves decide how to resolve conflicts between divergent versions of a file. Similar systems include Deno [5] and Mobisnap [12]. Like Bayou, none of these systems has been designed with power efficiency as a goal.

FlashBack also shares common issues and design element with other P2P data management system. Microsoft Research's Farsite [1] aims to provide a very high degree of reliability by replicating data across PCs sharing a local area network. OceanStore [10] aims to build a transparent and reliable peer-to-peer storage system. CFS [6] and OceanStore have been using distributed hash tables [13] [14] [17] [23] to build stable data repositories from dynamic collection of peers distributed across the internet. These systems have primarily been designed for PC-class devices performing traditional file-oriented workloads and as a result, are very different in design than FlashBack.

The FlashBack mechanism for providing reliability is based on replication. Replication systems are well explored in distributed file systems [3] [15] and commercial database systems [11] [18]. These systems tend to focus on the issues of trading off consistency and reliability for performance. The database systems utilize log sniffing, which is not applicable in our environment since it does not deal with a write-ahead logging (WAL) transactional environment. On the other hand, FlashBack makes the best effort to replicate local data given the constraints imposed by the power and storage resources on these impoverished devices. Moreover, traditional replication systems tend to assume homogeneity among the participating machines, which will not hold true in our usage scenarios.

5. FUTURE WORK

As part of our future work, we intend to explore the following areas in detail:

- **Further Experimentation –** While our experiments have been useful in demonstrating the performance characteristics of Flashback, they do not model exactly devices used in PANs. To conduct more realistic experiments, we intend to capture real-world device configurations, user behaviors and workload over time and use these traces to drive a measurement of Flashback. We have conducted a simple experiment in section 3 that shows that recall can stay reasonably high if each device participating in the PAN is able to push replicas to devices that it sees more often in the PAN. This is especially important as we foresee usage scenarios where there will be multiple PANs merging and splitting over time. As part of our future work, we intend to conduct experiments with multiple PANs periodically merging and splitting, and examine the effectiveness of the FlashBack utility. We also intend to investigate the effects of having different durations of lease periods on reliability and power consumption.
- **Adaptive Utility Function** -- Our first experiment showed that even for a stable PAN, picking the right utility function is crucial. We have picked a simple utility function that is easy to implement and works well in our experiments. Despite our success, we feel that the results can be improved further by picking the utility function (with appropriate weights) adaptively at runtime. This is an interesting area of future research.
- **Adaptive Replication** -- Different devices have different levels of reliability. For example, a server within the PAN is definitely more reliable than a small mobile device. A more accurate measure of replication factor would be to allocate a greater factor to replicas on devices that have greater resources, and a lesser factor to flaky devices. The replication factor is currently hard-coded based on the priority of the data. As future work, we should tune the replication factor at run time based on the resources of devices within a PAN.

6. CONCLUSIONS

FlashBack is an algorithm for performing transparent backups between peer devices in a personal area network. FlashBack has been designed to be power efficient and to require no extra work on the part of the user. FlashBack replicates lazily and maintains small neighbor sets to keep power usage as low as possible. Device co-locality and background neighbor improvement are used to achieve high recall.

The FlashBack system proposed in this paper is a proof-of-concept that such as backup system is not only viable, but can also potentially provide real value to end-users. We believe that the

successful deployment of FlashBack will enable new and interesting applications to be possible for PANs, while diminishing user's concern for the safety of application data. Moreover, Flashback enables users to easily "borrow" devices, have them join their PANs, and quickly have their corresponding data within the PAN available to these borrowed devices. Thus, Flashback may serve as an important element in making what Mark Weiser described as scrap devices [22] possible.

7. REFERENCES

[1] ATUL ADYA, WILLIAM J. BOLOSKY, MIGUEL CASTRO, GERALD CERMAK, RONNIE CHAIKEN, JOHN R. DOUCEUR, JON HOWELL, JACOB R. LORCH, MARVIN THEIMER AND ROGER P. WATTENHOFER. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)

[2] KENNETH C. BARR AND KRSTE ASANOVIC**,** Energy Aware Lossless Data Compression, The First International Conference on Mobile Systems, Applications, and Services, San Francisco, CA, May 2003.

[3] W. J. BOLOSKY, J. R. DOUCEUR, D. ELY AND M. THEIMER. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs/ Proceedings of the international conference on Measurement and modeling of computer systems, 2000, pp. 34-43

[4] LANDON P. COX, CHRISTOPHER D. MURRAY, AND BRIAN D. NOBLE. Pastiche : Making Backup Cheap and Easy. 5th Symposium on Operating Systems Design and Implementation. Boston, MA, December 2002.

[5] U. CETINTEMEL, P. J. KELEHER, AND M.FRANKLIN. Support for Speculative Update Propagation and Mobility in Deno**.** In 22nd International Conference on Distributed Computing Systems, *2001*.

[6] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, Wide-area cooperative storage with CFS, ACM SOSP 2001, Banff, October 2001

[7] K. FISHKIN, L. PARTRIDGE AND S. CHATTERJEE. User Interface Components for Lightweight WPANs. In IEEE Pervasive Magazine special issue on Wearable Computers.

[8] J. GRAY, P. HELLAND, P. O'NEIL AND D. SHASHA. The Dangers of Replication and a Solution. SIGMOD Conf. 1996

[9] HiveCache: Distributed Enterprise Online Backups. http://www.mojonation.net/

[10] J. KUBIATOWICZ, D. BINDEL, Y. CHEN, S. CZERWINSKI, P. EATON, D. GEELS, R. GUMMADI, S.RHEA, H. WEATHERSPOON, W. WEIMER, C. WELLS, AND BEN ZHAO. OceanStore: An Architecture for Global-Scale Persistent Storage, In Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000), November 2000.

[11] Microsoft SQL Server, Replication for Microsoft SQL Server 7.0 whitepaper.

[12] N. PREGUIÇA, C. BAQUERO, J. L. MARTIN. MobiSnap: Managing Database Snapshots in a Mobile Environment. In the Actas do 1º Encontro Português de Computação Móvel, November 1999.

[13] A. ROWSTRON AND P. DRUSCHEL, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, pages 329-350, November, 2001

[14] S. RATNASAMY, P. FRANCIS, M. HANDLEY, R. KARP AND S. SHENKER. A Scalable Content-Addressable Network. In Proceedings of ACM SIGCOMM 2001.

[15] G. SWART, A. BIRRELL, A. HISGEN AND T. MANN. The Echo Distributed File System. SRC Research Report 111.

[16] SATYANARAYANAN, M., KISTLER, J.J., SIEGEL, E.H. Coda: A Resilient Distributed File System, IEEE Workshop on Workstation Operating Systems, Nov. 1987, Cambridge, MA.

[17] I. STOICA, R. MORRIS, D. KARGER, F. KAASHOEK, AND H. BALAKRISHNAN, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM 2001, San Deigo, CA, August 2001, pp. 149-160.

[18] Replication Server: An Architecture for Distributing and Sharing Corporate Information. A Sybase White Paper

[19] SyncML, http://www.syncml.org/

[20] D. B. TERRY, M. M. THEIMER, K. PETERSEN, A. J. DEMERS, M. J. SPREITZER AND C. HAUSER. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In Proceedings 15th Symposium on Operating Systems Principles (SOSP), 1995

[21] Universal Plug and Play. http://www.upnp.org/.

[22] M. WEISER. The computer for the 21st century. Sci. Amer. September 1991, 933--940.

[23] B. Y. ZHAO, J. KUBIATOWICZ, AND A. JOSEPH. Tapestry: An Infastructure for Fault-tolerant Wide-area Location and Routing. Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, Computer Science Department, 2001.