# Slingshot: Deploying Stateful Services in Wireless Hotspots

Ya-Yunn Su and Jason Flinn

University of Michigan

Presented by Adrian Kirchner

## Motivation

- Mobile devices have limited resources
- Users want to run resource-intensive applications
- Remote execution introduces latency
- Connections within the local network are often much faster than the network’s connection to the internet
  - Even on wireless networks
- Backhaul connections between the network and the internet are a bottleneck for remote execution of an application

## Proposed Solution

Create alternative servers on surrogate computers closer to the user to help with latency

Client device will have a proxy that manages connections to each of the servers and helps coordinate consistency between them

## Surrogate Computers

- Located at local hotspots on the same local network as the user
- Contains a replica of the remote server application
- Maintains a high-speed connection with the user to reduce latency
- Maintains a lower-speed connection with the main server to ensure consistency
- Runs application in VM to reduce the impact of hardware and software discrepancies

## Client Proxy

- Sends requests to all servers
- Uses the first response returned
- Checks consistency between responses
- Responsible for initializing and managing replicas
- Maintains a log of all requests
    - Can be replayed to ‘catch-up’ one server to another

## Ease of Maintenance

- Surrogates should be as easy to deploy and manage as a wireless hotspot
  - Expectation: Surrogate availability in bookstores, coffee shops, etc
- To make surrogates easy to manage:
  - The required software base is very limited (Host OS, VMware, Slingshot)
  - All applications use a heavy-weight virtual machine
  - No hard state is maintained, just a cache

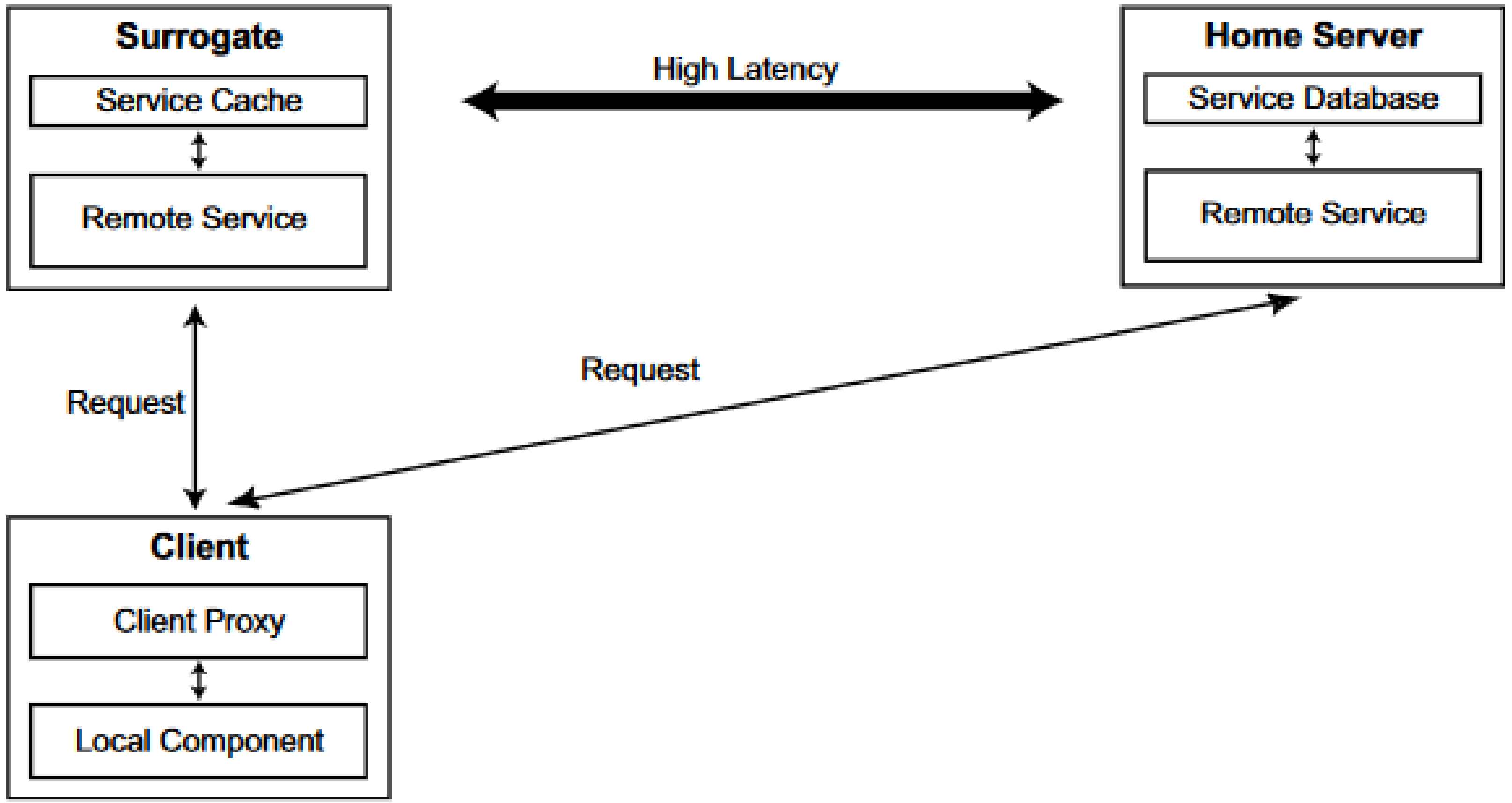


**Figure 1.** Slingshot architecture

## Service Database

- Stores all persistent state for each service
- The disk image of each virtual machine is divided into chunks and indexed by it's SHA-1 hash value
  - Assumes that any two blocks with the same hash value are identical
  - Allows for common data between two VMs to be shared instead of duplicated
  - Each chunk stores a reference count to ensure that shared data is not deleted

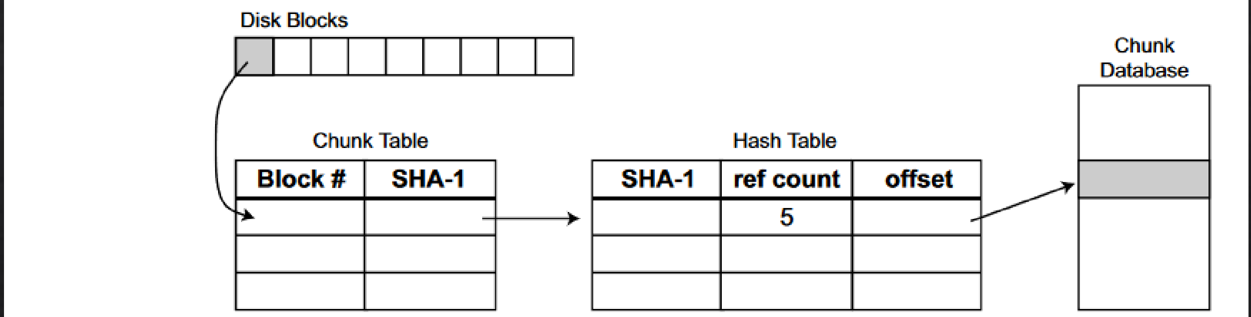


**Figure 2.** Reading a chunk from the service database

## Home Server Director

- Starts a VM and restores its volatile state from disk in response to an initial request from the client
- Retrieves the persistent state from the service database as needed
- Saves the volatile state and terminates the VM when notified by the client proxy that the user has closed the application
  - Volatile state is compressed before saving

## Surrogate Director

- Receives requests from client proxy to create a replica
- Requests the application state from the home server
  - Home server suspends and restarts the VM to create a checkpoint
- Starts a replica server using the application state
- Receives replayed requests from the client to catch up to the home server

## Service Cache

- Uses a similar hash-based storage architecture to the service database
- Duplicate data can exist across multiple services and users
- Uses copy-on-write approach for stateful applications
- All chunks on the replica are included in the reference counts in the system database, to prevent deletion of a chunk being used by the replica
- When a chunk is modified, it is pinned in the cache so that subsequent requests to that chunk are not affected by non-determinism from disk I/O on the home server
- The cache uses an LRU algorithm, excepting currently pinned chunks

## Client Proxy

- Responsible for discovering new surrogates to host replicas
- Maintains an event log for replicating state across multiple replicas
- When the user moves to a new network, the proxy can instantiate a new replica on a closer host
- If the application is stateless, the home server checkpointing and bringing the replica 'up-to-date' can be skipped

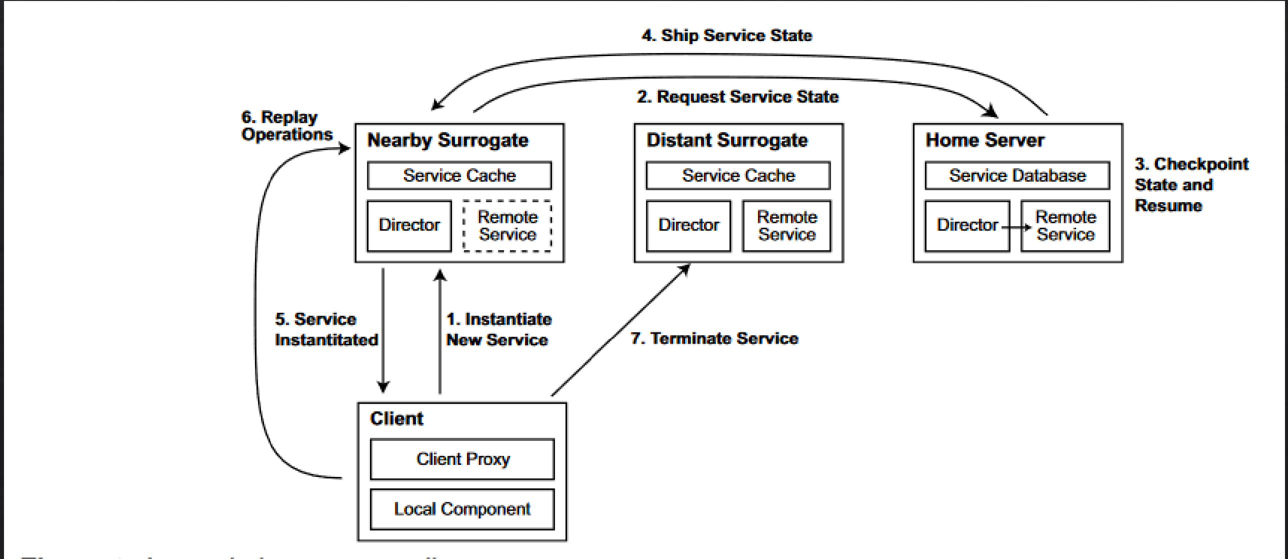


**Figure 3.** Instantiating a new replica

## Mobile Storage

- Instead of transferring state from the home server to each replica over the internet, which can be slow, a copy of the server application can be saved on the mobile device
- This allows the basic application state to be transferred over the higher-speed local network
- The event log can then be replayed to bring the replica up to date
- The user can create a new checkpoint of the server application whenever she’s near the home server
- There will be cases where the event log will be long enough that the advantage is lost
- When the service cache misses, the replica can check for a copy of the necessary chunk on the mobile device first before requesting it from the home server

# Evaluation

- How much do surrogates improve interactive response time?
- What is the perceived performance impact of instantiating a new replica?
- How much does the use of mobile storage reduce replica instantiation time?

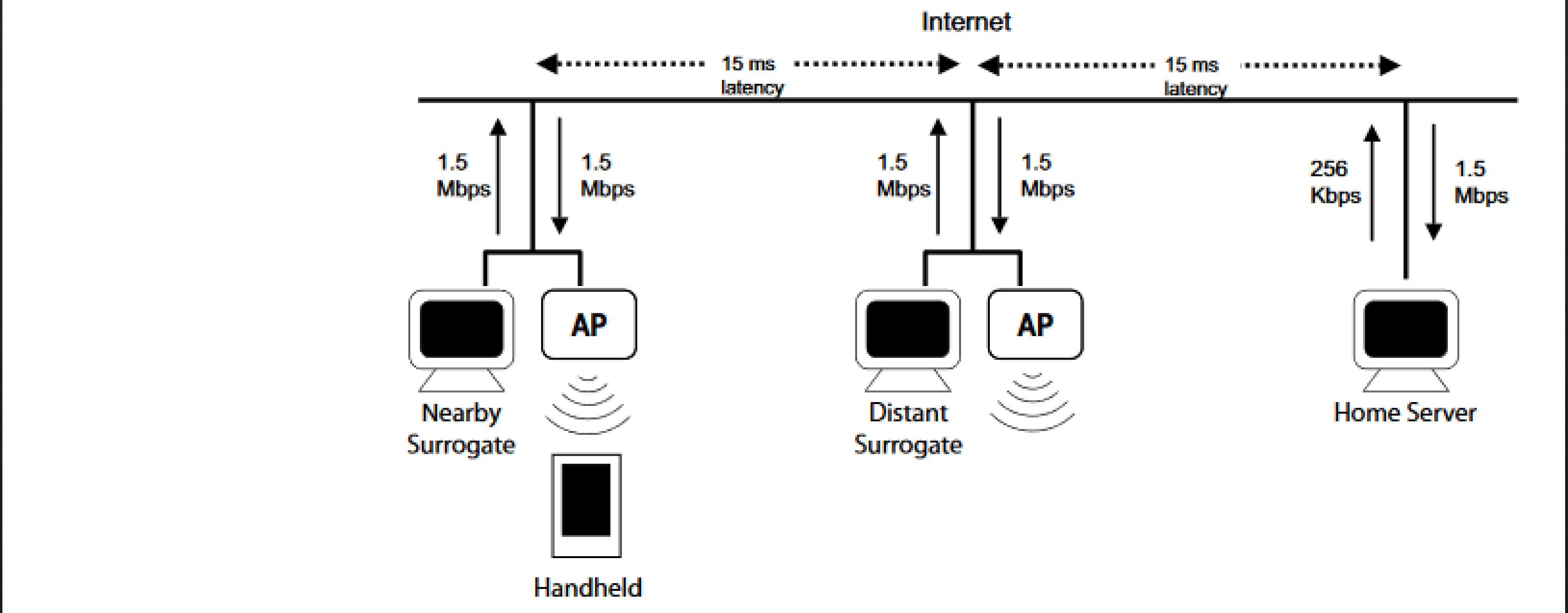


**Figure 4**. Network topology used in evaluation

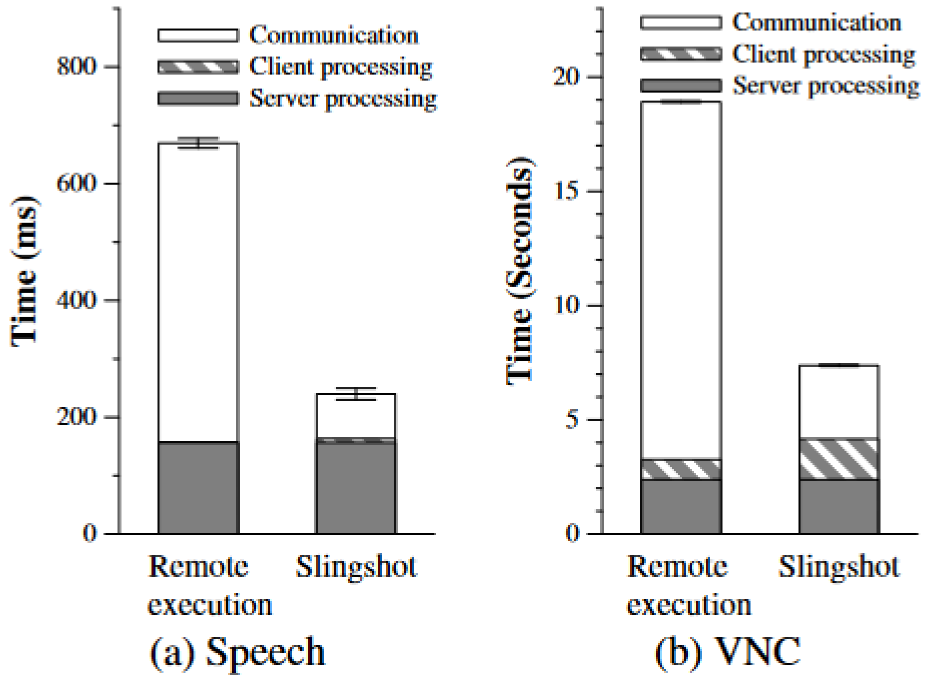
## Applications

### Speech Recognition

- Stateless
- Client sends a voice sample to the server and receives a text string back

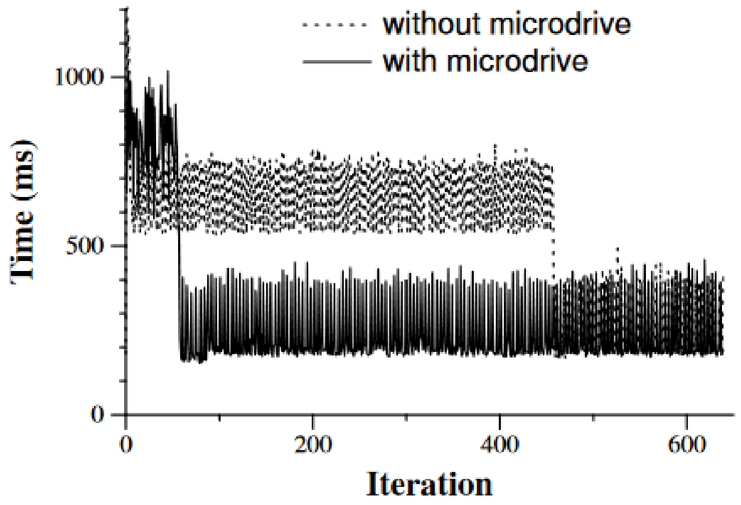
### Remote Desktop

- Stateful
- Client sends inputs to server, and receives display updates
- Display updates can be non-deterministic, so an application-specific wrapper was necessary to make sure checkpoints and event log could be made and replicated



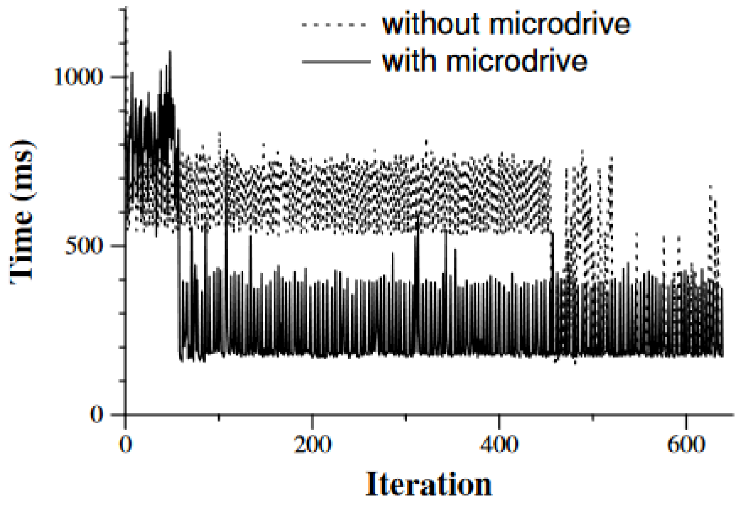
This graph compares the average time to execute the speech and VNC workloads when using remote execution and when using Slingshot. Each bar shows mean response time—the error bars are 90% confidence intervals.

**Figure 5**. Benefit of using Slingshot



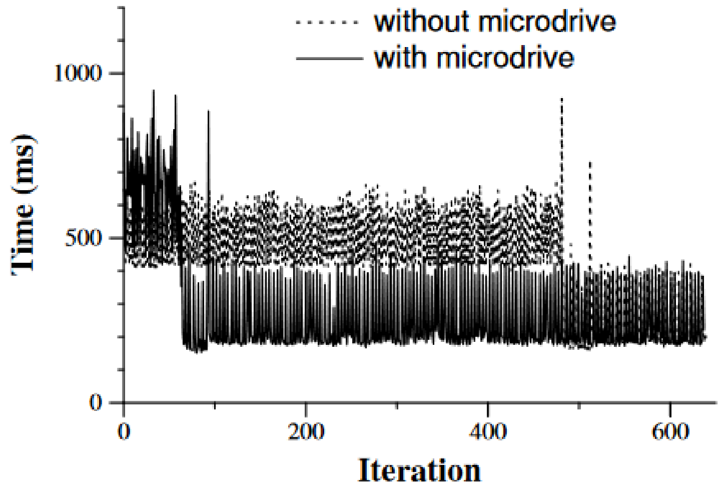
This graph shows how response time changes during the instantiation of a speech replica on the nearby surrogate. All chunks are in the service cache prior to each experiment.

**Figure 6**. Speech replication with warm cache



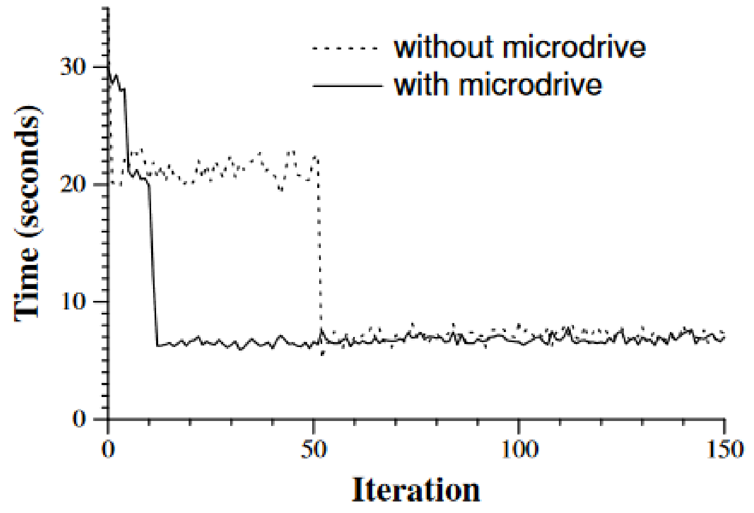
This graph shows how response time changes during the instantiation of a speech replica on the nearby surrogate. No chunks are in the service cache prior to each experiment.

**Figure 7**. Speech replication with cold cache



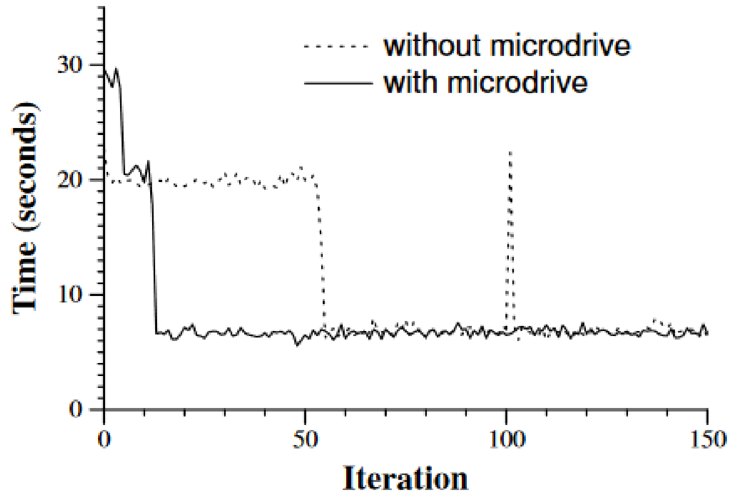
This graph shows how response time changes during the instantiation of a speech replica on the nearby surrogate while another replica executes on the distant surrogate. All chunks are in the the service caches prior to each experiment.

**Figure 8**. Speech: Moving to a new hotspot



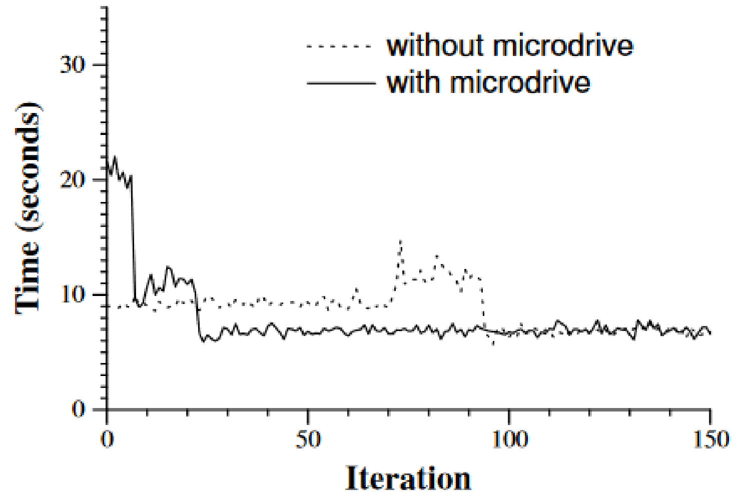
This graph shows how response time changes during the instantiation of a VNC replica on the nearby surrogate. All chunks are in the service cache prior to each experiment.

**Figure 9**. VNC replication with warm cache



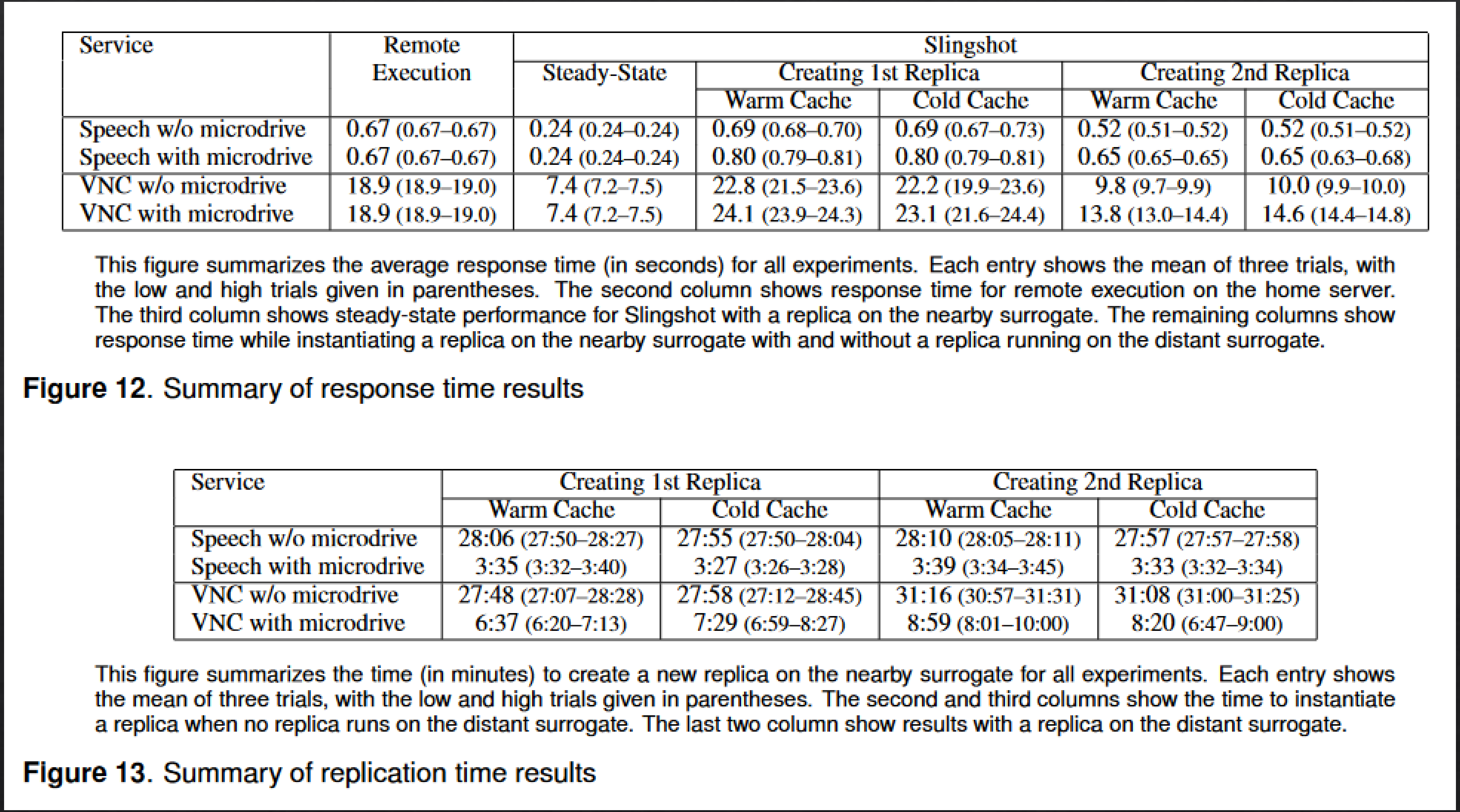
This graph shows how response time changes during the instantiation of a VNC replica on the nearby surrogate. No chunks are in the surrogate cache prior to each experiment.

**Figure 10**. VNC replication with cold cache



This graph shows how response time changes during the instantiation of a VNC replica on the nearby surrogate while another replica executes on the distant surrogate. All chunks are in the the service caches prior to each experiment.

**Figure 11**. VNC: Moving to a new hotspot



| Service | Remote Execution | Slingshot | | | | | |
|---|---|---|---|---|---|---|---|
| | | Steady-State | Creating 1st Replica | | Creating 2nd Replica | | |
| | | | Warm Cache | Cold Cache | Warm Cache | Cold Cache | |
| Speech w/o microdrive | 0.67 (0.67–0.67) | 0.24 (0.24–0.24) | 0.69 (0.68–0.70) | 0.69 (0.67–0.73) | 0.52 (0.51–0.52) | 0.52 (0.51–0.52) | |
| Speech with microdrive | 0.67 (0.67–0.67) | 0.24 (0.24–0.24) | 0.80 (0.79–0.81) | 0.80 (0.79–0.81) | 0.65 (0.65–0.65) | 0.65 (0.63–0.68) | |
| VNC w/o microdrive | 18.9 (18.9–19.0) | 7.4 (7.2–7.5) | 22.8 (21.5–23.6) | 22.2 (19.9–23.6) | 9.8 (9.7–9.9) | 10.0 (9.9–10.0) | |
| VNC with microdrive | 18.9 (18.9–19.0) | 7.4 (7.2–7.5) | 24.1 (23.9–24.3) | 23.1 (21.6–24.4) | 13.8 (13.0–14.4) | 14.6 (14.4–14.8) | |

This figure summarizes the average response time (in seconds) for all experiments. Each entry shows the mean of three trials, with the low and high trials given in parentheses. The second column shows response time for remote execution on the home server. The third column shows steady-state performance for Slingshot with a replica on the nearby surrogate. The remaining columns show response time while instantiating a replica on the nearby surrogate with and without a replica running on the distant surrogate.

**Figure 12**. Summary of response time results

| Service | Creating 1st Replica | | Creating 2nd Replica | |
|---|---|---|---|---|
| | Warm Cache | Cold Cache | Warm Cache | Cold Cache |
| Speech w/o microdrive | 28:06 (27:50–28:27) | 27:55 (27:50–28:04) | 28:10 (28:05–28:11) | 27:57 (27:57–27:58) |
| Speech with microdrive | 3:35 (3:32–3:40) | 3:27 (3:26–3:28) | 3:39 (3:34–3:45) | 3:33 (3:32–3:34) |
| VNC w/o microdrive | 27:48 (27:07–28:28) | 27:58 (27:12–28:45) | 31:16 (30:57–31:31) | 31:08 (31:00–31:25) |
| VNC with microdrive | 6:37 (6:20–7:13) | 7:29 (6:59–8:27) | 8:59 (8:01–10:00) | 8:20 (6:47–9:00) |

This figure summarizes the time (in minutes) to create a new replica on the nearby surrogate for all experiments. Each entry shows the mean of three trials, with the low and high trials given in parentheses. The second and third columns show the time to instantiate a replica when no replica runs on the distant surrogate. The last two column show results with a replica on the distant surrogate.

**Figure 13**. Summary of replication time results

## Conclusions

- Slingshot provides a significant performance improvement over a simple home server
- For maximum benefit, surrogates should be located at wireless hotspots
- Slingshot uses off-the-shelf software to help with installation and maintenance of surrogates
- Replication lets surrogates transfer responsibility and fail gracefully, even with stateful services
- Surrogates do not need to be available 24/7, since the home server can act as a fallback
- Security is a weak point and needs to be addressed before this is deployed
- Automatic load balancing and replica creation/destruction is not implemented