Overcoming Cellular Connectivity Limitations with M2Blue Autonomic Distributed Data Caching

[1]Brian Dougherty, [1]Daniel Guymon, [2]Douglas C. Schmidt and [1]Jules White
[1]Virginia Tech, {brianpd,dguymon,julesw}@VT.edu
[2]Vanderbilt University, schmidt@dre.vanderbilt.edu

Abstract

The memory constrained nature of mobile devices, such as smartphones, limits the amount of data that can be stored locally. As a result, mobile devices often rely on cellular connections to retrieve application data. Environmental factors, however, can partially or completely restrict cellular connectivity. Autonomic distributed caching mechanisms can be used to allow mobile device networks to self-heal by storing data needed across multiple devices, but cannot be applied without a means to determine if devices are within a given range. Moreover, it is hard to identify the best way(s) of mapping application data to device memory to allow devices to self-heal in spite of limited cellular connectivity.

This article provide the following contributions to the study of autonomic self-healing mobile communication: (1) we present a Bluetooth-driven localization technique that determines the quantity and configuration of devices present within a given range, (2) we provide a system for mapping data across nearby devices allows mobile computing environments to self-heal in response to limited cellular connectivity, and (3) we show this autonomic system can be augmented to handle the dynamic nature of devices entering and leaving ad hoc mobile networks.

1 Introduction

Current trends and challenges Mobile computing devices, such as smartphones, use cellular connections to retrieve information needed by applications. Unfortunately, cellular connectivity can be lost or restricted by environmental factors, such as physical barriers, or by a lack of cellular coverage [11, 2]. A promising means for constructing reliable mobile computing environments therefore involves implementing autonomic devices [10] that can self-heal in response to environmental changes, such as slow or unreliable cellular connections.

Autonomic computing is computing paradigm in which systems have the ability to self-manage without intervention by an outside entity [6]. One potential benefit of autonomic systems is the introduction of self-healing properties. Systems with self-healing capabilities are able to remain functional by reconfiguring or repairing themselves in response to system faults, potentially increasing their fault-tolerance and reliability.

Memcached [5] is a promising mechanism that could be applied to create an autonomic mobile computing enviroinment in which devices can self-heal in response to limited cellular connectivity. Memcached is a distributed data caching mechanism that uses hashing to map data to multiple computing nodes. When data stored with Memcached is needed, a requesting node can use the hash function to retrieve data either locally, if a local copy exists, or from another node that it has been stored on. Figure 1 shows how mobile computing environments could use autonomic distributed caching mechanisms like Memcached to implement self-healing communications [1].

Introducing autonomic properties, such as self-healing, canallow mobile networks to continue to function in spite of limited cellular connectivity by storing application data on the device. Resource-constrained mobile devices, however, often cannot store the complete set of data needed by applications. Since multiple devices often use the same application, data could be distributed between devices so that collectively, most or all application data is stored across devices. This approach can reduce the need for data requests across the cellular network, thereby allowing devices to execute autonomically in the absence of cellular connectivity.

Open problem $\Rightarrow$ Applying autonomic distributed caching to add self-healing communication in mobile computing systems. There are several issues that currently prevent the use of Memcached to implement self-healing communication in mobile networks. First, Memcached must know the devices available in a distributed system [12], which is relatively simple in environments where stationary devices are added/removed before execution. A mechanism is therefore needed to autonomically determine the profile of devices that are present in a mobile network so Memcached can execute.

Moreover, the devices present in a given mobile network change over time. Devices may physically enter a network by coming in range of other devices or exit the network by going out of range, crashing, or running out of power [8].

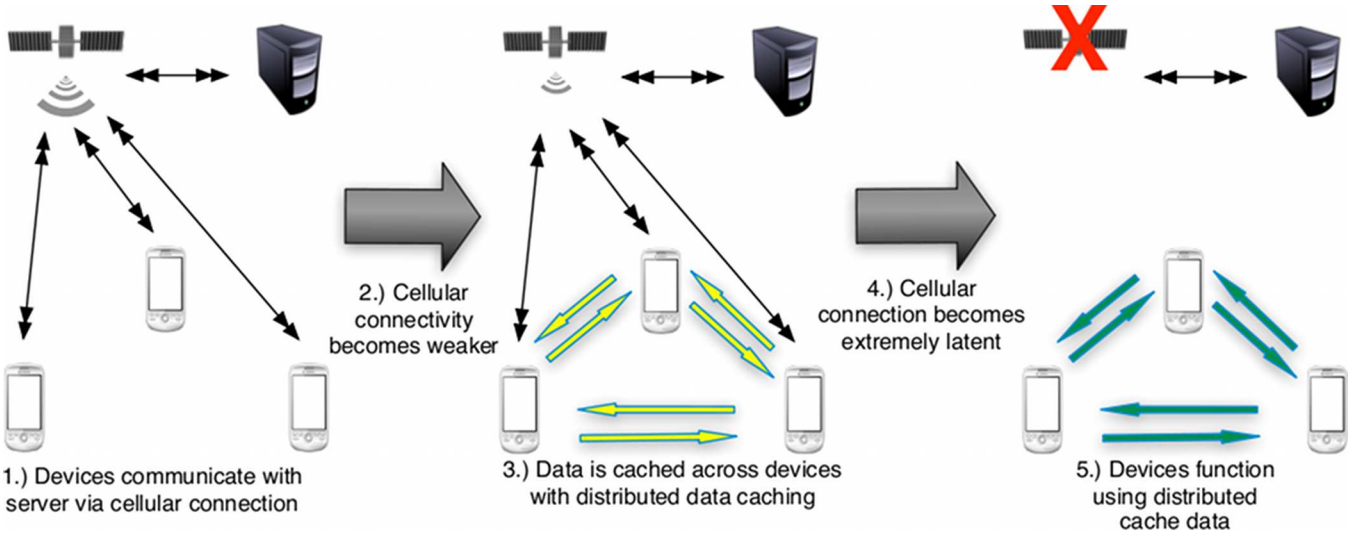

Figure 1. Overcoming Weak Cellular Connectivity with Mobile Autonomic Distributed Data Caching

Any autonomic distributed caching mechanism that allows a mobile network to function despite limited connectivity must self-heal in response to new devices entering or exiting the network.

Solution approach → Autonomic mobile distributed data caching with M2Blue This article presents *Mobile Memcached with Bluetooth* (M2Blue), which is an autonomic distributed data caching mechanism aimed at mobile computing environments with limited or no cellular connectivity. M2Blue uses Bluetooth to automatically detect devices present within a predefined range in the event of reduced cellular connectivity. An augmented version of Memcached is then applied to map application data across devices. Devices use Bluetooth to read cached data instead of transmitting requests to a cellularly connected server, thereby minimizing the need for cellular connections.

This article provides the following contributions to autonomic self-healing in mobile distributed systems:

• We provide an innovative approach for automatically determining the profile of distributed mobile computing environments,

• We present an algorithmic technique for applying autonomic distributed data caching to mobile computing environments so systems can self-heal in response to limited cellular connectivity, and

• We provide mechanisms for autonomously supporting the dynamic entry or exit of devices from mobile networks despite limited cellular connectivity.

2 Challenges of Applying Autonomic Distributed Caching Mechanisms to Allow Mobile Networks to Self-heal

This section summarizes the challenges of implementing self-healing communication in autonomic distributed caching mechanisms to allow mobile computing environments to self-heal in periods of limited cellular connectivity.

Data transmission between devices cannot require the use of a cellular connection. Even if the profile of a mobile device network is known, devices must be able to exchange data with each other to take advantage of autonomic distributed data caching. Transmitting data over a cellular connection, however, may not be possible due to increased latency as a result of diminished signal strength. An alternative communication protocol, therefore, must be used to transmit data wirelessly. Ideally, this protocol would not require additional hardware.

Autonomically determining devices present within a physical range must be done at runtime. Unlike stationary nodes, the number of mobile devices in a network within a range can be hard to predict. Determining which devices are present is exacerbated when locations where cellular connectivity becomes weak or limited are not known *a priori*. Memcached, however, requires knowledge of devices available for storing data to determine an appropriate hash function. Autonomic mechanisms must be developed that can routinely determine the profile of mobile device networks autonomically so Memcached can be applied.

Devices that later enter/exit the mobile network should be able to participate in autonomic distributed data caching. Many factors can cause the number of de-

vices in a mobile network to change, *e.g.*, devices may move out of range with each other, crash, break, or run out of power and turn off, removing any previously cached data. Devices entering the network, however, should be able to access previously cached data on the devices that remain. Any autonomic distributed caching mechanism must therefore be augmented to handle the dynamic nature of mobile networks.

3 Using Mobile Distributed Caching with M2Blue to Self-heal in Response to Limited Cellular Connectivity

Mobile Memcached with Bluetooth (M2Blue) is an autonomic distributed data caching mechanism we created to (1) allow mobile devices continued access to data by using autonomic distributed caching to self-heal in the event of limited cellular connectivity and (2) provide a mechanism for caching data that can self-heal to function despite the entry/-exit of devices from the network. This section describes the sequence for preparing, securing, and commencing autonmic distributed caching with M2Blue.

3.1 Autonomically Preparation of Devices to Self-heal

Prior to being fielded, each device is preloaded with a background service that periodically checks cellular connectivity. As devices move away from cellular towers and signal fades, cellular connectivity becomes incrementally weaker and drastically increases in latency. Once a predefined threshold of reduced connectivity is reached, a device, referred to as the *initiator*, will begin the self-healing process by autonomically invoking the *begin* command.

The initiator uses Bluetooth to detect the presence of nearby devices and then transmit a packet containing its device ID and a list of detected device IDs to the server over the cellular connection. Upon receiving this information, the server will reply to the initiatior with a packet containing the public key and a hash function. The initiator will then use Bluetooth to broadcast the public key and hash function to all discovered devices, creating a mobile network profile that can be used in conjunction with autonomic distributed caching to allow self-healing in the face of limited connectivity.

3.2 Secure Self-healing with Public/Private Key pairs

Upon recieving packets from all devices, the server begins the server-side M2Blue protocol. First, when a request for data is received, the server uses a predefined policy (*i.e.*, request frequency) to determine if the data associated with the key should be written to the distributed cache by the device. If so, the server produces a string, referred to as the *data tag*, that concatenates the key sent by the device, the data associated with the key from the server, and an expiration timestamp for the data. This string is then encrypted with the server's private key and transmitted to the device with the key and requested data. If the data should not be cached then only the key and data are transmitted to the device.

3.3 Autonomic Device Modes: Read/Write/Store

For the M2Blue protocol, a device can perform 3 different caching operations—write, read, and store—as described below.

3.3.1 Write Operation

The write operation defines how data is written into the autonomic distributed cache when the key, data, and data tag are received by a device from the server. First, the key is applied to the hash function received from the initiator to determine the destination device ID and memory location for storage. Each device also receives an *alias table* listing all device IDs from the intiatior during preparation. The alias table is used to determine if another device is now handling the storage of the data for the device ID determined by hashing the data key.

Once a request to store data is received by the destination device, the data and the data tag is stored in the appropriate location. If the destination device is not detected, then the request is forwarded to the initiator. If the initiator cannot detect the device, then the device is removed from the the alias table of the initiator, which is then broadcasted to all other devices.

For example, consider a mobile network consisting of four devices. After the preparation and data caching begins, Device 2 leaves the network. Upon attempting to store data to Device 2, the requesting device will not be able to detect the device and will forward the request to initiator device. The initiator will attempt to detect the device, fail to detect it, and then update its alias table so that all data being stored to Device 2 will now be stored to another specific device, such as Device 3, and then broadcast the table to all other devices. Upon receiving the new alias table, the requesting device will now look up Device 2 in the alias table and determine that Device 3 is the new location for data intended for Device 2.

3.3.2 Read Operation

The read operation allows a device requesting data to access autonomically cached data, thereby potentially avoiding additional requests to the server across the cellular network.

When the requesting device requires a data, the key representing that data, such as a variable name, is applied to the hash function received from the initiator to determine the destination device ID and address where the requested data would be stored in the distributed cache.

The destination device ID is checked in the alias table to determine if another device is now handling the storage for that device ID. A request is then sent over Bluetooth to the appropriate destination device for the requested data which then transmits the data and data tag stored at the requested location back to the device. The data tag is then decrypted using the public key to determine that the data corresponds to the correct key and has not expired.

If the device is not detected the request is forwarded to the initiator. The initiator either forwards the request to the destination device and replies back with the data and data tag or determines the device is no longer present, updates the alias table accordingly, and broadcasts the table to all other devices. If the data is not retrieved from the distributed cache, the device must hold until a cellular connection can be made.

3.3.3 Store Mode

While all devices should be able to read from and write to the cache, all devices are not always used to store cached data. This state is represented as Boolean variable named *store mode*. Each device involved in the preparation phase of M2Blue has store mode set to true by default, allowing it to store data. New devices that join the network after data has been cached may cause the network to grow larger than specified in the hash function generated during the preparation phase. These devices, however, could be used later to store data in the event of other devices leaving the network.

For example, the hash function generated by the server accounts for 5 different devices. If a new device joins and increases the number of devices to 6, the new device by default will not be in store mode, but will be able to read and write to the cache after making a request for the hash function, public key, and alias table from the initiator. If one of the other devices leaves the network, the new device can change store mode to true , update the alias to take the exited device's place, and then broadcast the new alias table to all devices. Anytime the alias table is updated any data cached to the original device is no longer available.

4 Related Work

This section compares our distributed caching technique for mobile devices with Memcached and Bluetooth localization.

Distributed caching techniques. Memcached is a distributed memory object caching mechanism designed to accelerate dynamic web applications by using a shared hash table to distribute data between multiple processes so that changes made by one process can be simultaneously seen by another [5]. Lerner tested the fusion of Memcached and Ruby on Rails by showing Memcached could reduce unnecessary server traffic [9]. Harris demonstrates Memcached execution time consistency in [7] by showing that the execution time of one-thousand consecutive cache accesses are relatively equal. Many websites, such as LiveJournal, Twitter, and Wikipedia, use multiple instances of Memcached on multiple servers to handle hundreds of website visits per second [5].

Our M2Blue mechanism differs from Memcached in several ways. In particular, Memcached is designed for cache handling on servers for websites, whereas M2Blue is used for mobile devices. Likewise, M2Blue provides self-healing properties for autonomically responding to periods of time where there is a decrease in cellular connectivity, which is a useful feature in autonomic computing where distributed systems must adapt to unpredicted changes.

Relative localization with Bluetooth. Relative localization with Bluetooth is the process of determining which mobile devices are present within a certain physical range. Cheung et al. [3] present a Bluetooth localization technique that uses beacons and smartphones to determine the location of the smartphone. These beacons dynamically change signal strength based on the physical layout of an environment. Fisher et al. used off-the-shelf Bluetooth beacons and a mobile client device to achieve +/- 1-meter accuracy by using phase difference calculations of the beacons and running the received beacon signals through a low-pass filter [4].

Our M2Blue mechanism uses Bluetooth to determine devices that are present, allowing us to execute the distributed caching algorithm without requiring additional hardware. Further, M2Blue is designed to autonomically apply distributed caching in response to limited cellular connectivity. This autonomic feature of self-healing in response to service availability allows M2Blue to remain robust despite limited connectivity.

5 Concluding Remarks

Providing mobile networks with the autonomic ability to self-heal in the event of limited cellular connectivity is hard. The dynamic nature of mobile networks, with devices entering and exiting unpredictably, makes it hard to apply self-healing communication with autonomic distributed caching mechanisms. Our M2Blue autonomic distributed caching mechanism helps mobile networks overcome these challenges and self-heal despite limited cellular connectivity.

Based on our work with M2Blue thus far, we have learned the following lessons pertaining to self-healing, autonomic mobile computing environments:

- **Detecting the presence of nearby devices does not require additional hardware.** Knowledge of the number and type of devices available is essential for autonomic distributed data caching. The M2Blue technique uses Bluetooth to effectively determine the presence of surround devices. Since Bluetooth comes preloaded on the majority of smartphones, such as the Google Android and Apple iPhone, no additional hardware is required to begin caching with M2Blue.

- **Responding to fluctuating network size is hard**. M2Blue uses Bluetooth to detect the presence of devices and alias tables to alter the destination of read-/write operations as devices leave the network. The impact of deciding which device to handle the exited devices cache responsibilities should be investigated.

- **Mobile networks use geo-location specific data**. Mobile devices are excellent platforms for using sensors, such as GPS and accelerometers, to capture physical data, making them ideal for location-specific applications. New cache-based removal policies should be investigated and applied to M2Blue that take into account the geographic origin of cached data.

References

[1] G. Anandharaj and R. Anitha. A distributed cache management architecture for mobile computing environments. In *Advance Computing Conference, 2009. IACC 2009. IEEE International*, pages 642–648. IEEE, 2009.

[2] Y. Chen and H. Kobayash. Signal strength based indoor geolocation. In *IEEE International Conference on Communications*, volume 1, page 436, 2002.

[3] K. Cheung, S. Intille, and K. Larson. An inexpensive bluetooth-based indoor positioning hack. *Proc. UbiComp06 Extended Abstracts*, 2006.

[4] G. Fischer, B. Dietrich, and F. Winkler. Bluetooth indoor localization system. In *Proceeding Workshop on Positioning, Navigation and Communication*, pages 147–56, 2004.

[5] B. Fitzpatrick. Distributed caching with memcached. *Linux journal*, 2004(124):5, 2004.

[6] A. Ganek and T. Corbi. The dawning of the autonomic computing era. *IBM Systems Journal*, 42(1):5–18, 2003.

[7] A. Harris. Distributed caching via memcached. *Pro ASP. NET 4 CMS*, pages 165–196, 2010.

[8] D. Johnsort. Routing in ad hoc networks of mobile hosts. In *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pages 158–163. IEEE, 1994.

[9] R. LERNER. Memcached integration in rails. *Linux Journal*, 2009.

[10] F. Saffre, J. Halloy, M. Shackleton, and J. Deneubourg. Self-organized service orchestration through collective differentiation. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(6):1237–1246, 2006.

[11] K. Wong. Geo-location in urban areas using signal strength repeatability. *Communications Letters, IEEE*, 5(10):411–413, 2001.

[12] P. Xiang, R. Hou, and Z. Zhou. Cache and consistency in nosql. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 6, pages 117–120. IEEE, 2010.