Electronic Preprint for Journal of Information Processing Vol.23 No.4

Invited Paper

Public Data Dissemination via Broadcasting

Michael Wisely[1,a)] Sahra Sedigh Sarvestani[1,b)] Ali R. Hurson[1,c)]

Received: December 3, 2014, Accepted: April 7, 2015

Abstract: Many modern mobile applications appeal to users because they grant access to a wealth of information anytime, anywhere. However, a number of obstacles stand between users and the data they seek. Firstly, mobile devices have limited access to energy sources. Devices should be able to access information without sacrificing hours of battery life. Secondly, users expect timely access to data. While respecting the energy limitations of devices, data must be quickly accessible. Data broadcasting has been proposed as a quick and efficient solution for providing users with the data they desire. Broadcasting disseminates data from a server in a way that is analogous to AM/FM radio or television broadcasts. Devices tune in to wireless channels to fetch data items from a broadcast. Several technical challenges must be addressed to ensure efficient and timely data access for clients. These include organizing, indexing, and accessing the broadcast data items. Despite these issues, broadcasting is a scalable and efficient method for disseminating data to mobile clients. In this paper, we describe related techniques and compare and contrast them with respect to response time and energy efficiency.

Keywords: public data, data broadcasting, scalable data dissemination, mobile computing, green computing

1. Introduction

Every day, mobile technology becomes faster, smaller, and more ubiquitous. Smartphones, smartwatches, and other intelligent devices can accomplish a wide variety of tasks that make our jobs and lives more efficient and convenient. The appeal of these intelligent devices often lies in their ability to connect to the outside world. From weather forecasts to Twitter feeds, mobile devices provide users with access to information anytime, anywhere.

While mobile devices are shrinking in form factor, the amount of data available to users is growing. For example, Twitter users tweet over 400 million times per day [26]. At 140 bytes per tweet, this amounts to well over 100 terabytes of data since the site's creation in 2006. Despite this volume, users expect rapid access to recent tweets and, on occasion, older posts.

Resources available to mobile devices are constrained in several ways. These devices rely on batteries or harvest small amounts of energy from their surroundings [10]. Without access to a long-term power source, they must leverage energy-efficient hardware and software to conserve energy resources. Extending battery life by choosing a low-power processor may outweigh the computational benefits that would otherwise be gained using higher performance processors [6].

While data should be made available to clients in a fashion that respects their resource limitations, these are not the only criteria to consider. Users expect long battery life and timely responses to data requests. That is, the time elapsed between a client's request and a server's response should be short.

[1] Missouri University of Science and Technology, Rolla MO 65409, USA
[a)] mwwcp2@mst.edu
[b)] sedighs@mst.edu
[c)] hurson@mst.edu

Additionally, the location and movement of clients within a mobile computing environment should not significantly impact the quality of their data access. To some extent, this issue is unavoidable. Communication hardware incurs some base costs in terms of energy consumption, regardless of a user's location within the environment [19]. By reducing the amount of communication required of the clients, it is possible to reduce energy costs.

In general, the challenge is to enable mobile clients to access data while reducing their response time and energy consumption. The best solution to this challenge depends on the nature of the application and the mobile computing environment. The type of data being accessed, as well as the communication model, can help identify the best way to connect clients with their data.

1.1 Mobile Computing Environment

Mobile computing environments must have some connection to a data source, such as the Internet, from which to provide data to clients. In a client-server model, mobile support stations (MSS) function as the server, connecting mobile devices with the data they desire. Data requested by mobile devices is retrieved from a data source by the MSS and passed along to mobile devices. Though it is possible for clients to collaborate and communicate with one another [7], [8] to help disseminate data, we are not concerned with this aspect of mobile computing environments. Instead, this discussion is concerned with efficiently disseminating data from the MSS to its clients.

Naturally, wireless communication methods are attractive for mobile computing environments, as it is inconvenient or impractical for moving devices to be tethered to a network by a cable. However, the convenience of wireless communication comes at a cost. Several issues must be considered when constructing a mobile computing environment.

Firstly, wireless communication demands energy from energy-restricted mobile devices. When a device is not communicating, it can switch its wireless radio to a low-power mode to conserve energy [20]. However, an active wireless radio consumes a large amount of energy to send and receive data. An actively communicating cellular radio in a smartphone can consume five times the base energy consumed by the phone itself [22]. Additionally, as a device travels away from a MSS, its connection becomes weaker, and the mobile device must use more energy to communicate. Due to signal amplification, devices with weak signals can consume up to six times more energy than devices with access to stronger signals [22]. This energy consumption may drain a mobile device's energy resources more quickly.

Secondly, mobile devices typically have limited computational resources. Slower processors are often chosen to reduce energy consumption [5] and improve battery life. Complex algorithms increase the response time for users, as well as energy consumption of the processor and other hardware components.

1.2 Data Dissemination Methods

The mobile computing environment should enable mobile clients to retrieve desired data from a data source in a fast and energy-efficient manner. The nature of the application and the information exchanged in a mobile computing environment determines how best to disseminate data. In a client-server environment, an MSS disseminates data from a data source to mobile hosts. Thus, it is partly the responsibility of the MSS to provide client devices with fast and energy-efficient access to the data they desire. However, the method used to propagate data to clients should be selected with consideration of the type of data desired by the clients. Data requested by mobile clients falls into one of three categories [15]:

Private Data Data that is intended to be accessed by a single user. Personal schedules and address books are examples of private data. The number of devices that read from and write to the data source is very small.

Shared Data Data that is shared among a group of users. It is possible for many users to read and write shared data. Social media sites, like Twitter and Facebook, use shared data. Many users are permitted to retrieve and update data from the data source.

Public Data Data that is publicly available to a wide audience. Very few users update the data source, but many users subscribe to it. Traffic and weather information are public data. Authorized users at news outlets or other institutions update information, and subscribers retrieve their updates.

Data can be disseminated in several ways [3]:

On Demand In an *on-demand* environment, the server responds to requests made by clients. A strictly on-demand service is purely pull-based. That is, clients submit requests to the server asking for information, and the server directly responds to the requesting client.

Broadcasting In a *broadcasting* environment, the server repeatedly broadcasts data to all clients in its transmission range. It is a purely push-based environment. Clients do not submit requests for information. Instead, they only download desired data from the server transmissions.

Hybrid A *hybrid* environment combines features of on-demand and broadcasting environments. For example, clients may be allowed to submit requests for information that influences the content of future broadcasts.

On-demand services are appropriate for disseminating private or shared data [11]. Unicast client-server communication enables the server to enforce access control rules, and two-way communication enables clients to request updates to data on the server. Public data, on the other hand, can be disseminated via broadcasting. In a broadcasting environment, the server repeatedly broadcasts data for client devices to receive. They can tune in to interesting parts of a broadcast and otherwise leave their wireless radios in a power-saving state. Although the server must expend energy to constantly transmit information, clients can save energy, as they are no longer required to transmit requests for the data that they seek. Hybrid services are appropriate for public data and possibly shared data. Items that are sufficiently popular may be broadcast, while less popular items are sent to individual clients [21].

Though broadcasting offers several benefits to client devices, it comes with a number of technical challenges. The server must decide which data items to include in each broadcast and how to disclose the location (channel) of each data item to clients. The client devices must identify methods for efficient retrieval of broadcast data items. The solutions to these challenges may vary based on the nature of the data to disseminate and its application. However, a properly implemented broadcasting environment offers scalability and could extend client battery life while maintaining short response times.

2. Broadcast Content

Methods that determine the contents of a broadcast fall into one of three categories: pull-based, push-based, or hybrid. *Pull-based* methods construct broadcasts solely based on requests submitted by users. These methods usually require the server to accept user requests on one communication channel and broadcast responses on a second communication channel. Because of this request/response behavior, pull-based methods may be called *on-demand* broadcast [4]. *Push-based methods*, on the other hand, do not accept user requests. Instead, these methods use previous knowledge of user access patterns to determine the content of broadcasts. Users are passive and tune in to broadcasts to read requested items as they become available. *Hybrid* methods combine features of push-based and pull-based approaches. Some data items are broadcast to clients, while other items must be requested in an on-demand fashion. Based on user requests, hybrid methods adjust the contents of broadcasts to more efficiently meet the needs of users.

The use of these methods varies by application. Push-based methods are appropriate for environments where user access patterns do not change frequently. In these situations, prior, static knowledge is sufficient for choosing broadcast contents. Hybrid or pull-based methods are more appropriate for environments where user access patterns are less predictable.

Broadcast communication takes place over discrete time slots,

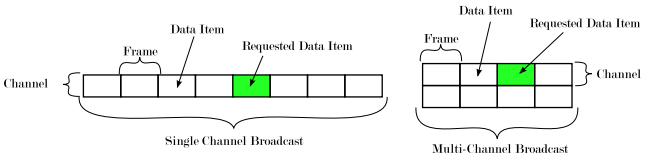


Fig. 1 Single- and multi-channel broadcasts.

each of which corresponds to one *frame* of the data. These frames may vary in size. If a single channel is used, a broadcast is simply a sequence of frames. When multiple, parallel channels are used, a broadcast can be viewed as a two-dimensional array. Each row of the parallel broadcast corresponds to a different channel, and each column corresponds to a different frame. **Figure 1** illustrates the difference between single- and multi-channel broadcasts. Note that the multi-channel broadcast of Fig. 1 contains the same number of frames as its single-channel counterpart, but the total length of the multi-channel transmission is shorter.

2.1 Pull-based Methods

Pull-based methods allow users to request data items from the server, while better utilizing bandwidth than an on-demand, unicast environment. These methods ensure that every broadcast data item will be useful to at least one user, so broadcast pages are not wasted on unrequested items. However, as the number of requests grows, it becomes increasingly difficult to answer all requests in a timely manner. While simpler pull-based methods will eventually respond to user requests, more advanced methods use information about the requests to decrease average response time.

The simplest method for determining broadcast responses is by servicing requests on a first-come first-serve (FCFS) basis. This treats each request with equal priority, but results in longer average wait times. Early work in this area considered other simple heuristics, such as most- requested-first (MRF) and longest-wait-first (LWF) [28]. When the number of user requests is low, these heuristics perform similarly. However, as the load increases, LWF maintains a lower average client response time than FCFS and MRF. This comes at a computational expense, as LWF incurs the greatest overhead in compute schedules.

Aksoy and Franklin evaluated these heuristics and showed that MRF has the shortest average response times for hot data items, while FCFS gives equal priority to each data item. Using these observations, they proposed the "$R \times W$" algorithm to reduce the response time of requests [4]. The algorithm balances access to hot and cold data items by maintaining two sorted lists, R and W, respectively. These lists reflect the number of requests for data items (R) and the time since the earliest request for data items (W). Starting at the most-requested item in the R list, the algorithm alternates between R and W, searching for the data item with the highest $R \times W$ value. The algorithm searches until half of each list has been covered, or until a precomputed limit has been reached. To improve the runtime of the algorithm, a parameterized threshold can be added, so that the search stops after finding a data item with an $R \times W$ value greater than the threshold. The threshold value is initially set to 1, and is updated after each broadcast according to Eq. (1), where $(R \times W)(t)$ is the $R \times W$ value from the previous broadcast. Use of a threshold may cause the algorithm to stop before an optimal $R \times W$ value is found. However, the authors argue that for highly skewed access patterns, the data item with the highest $R \times W$ value tends to be one of the first data items in the search. By choosing data items that are "good enough," but not optimal, broadcast contents can be determined quickly while still reducing response times.

$$threshold(t+1) = \frac{threshold(t) + (R \times W)(t)}{2} \tag{1}$$

$R \times W$ aims to reduce the response time of user requests, but it does not guarantee any duration by which the requests will be answered. Some applications may require a response to a request within a short window of time, after which a response would be rendered obsolete. Xu et al. [29] proposed SIN-α to schedule broadcasts in response to time-critical requests. SIN-α aims to respond to as many requests as possible by reducing the request drop rate. The algorithm chooses broadcast contents by considering when request deadlines expire and how many requests will be serviced. It maintains two sorted lists, denoted as the *S-list* and *N-list*, respectively. The *S-list* records the earliest deadline for data items, and the *N-list* records the number of requests submitted for data items. The lists are traversed to find the data item with the minimum $sin.\alpha$ value, defined in Eq. (2), where *num* represents the number of requests. Requests remain in the lists until they are serviced or expire. If a request for a cold item has a short deadline, it may not have a low enough $sin.\alpha$ value to be serviced, and the request will be dropped by both the server and the client.

$$sin.\alpha = \frac{closestDeadline}{num^{\alpha}} \tag{2}$$

2.2 Push-based Methods

Without explicit requests for data, push-based methods must rely on previous knowledge of user access patterns to decide the content of broadcasts. This eliminates the need for a communication channel for user requests, but it requires push-based methods to broadcast the full contents of a database. Removing the request bottleneck makes push-based methods highly scalable, although it limits possible client applications.

Flat broadcasting is the simplest method for determining the content of a push-based broadcast. Given information about client requests, the server takes the union of all requested data items and repeatedly broadcasts the results [1]. Without any organization or repetition of data items, the average wait time for any data item is half of the broadcast length. All data items are broadcast with the same priority. Thus, every client can eventually retrieve their desired data. However, it is unlikely that all data items will be equally popular among clients. Some items will be requested more frequently than others. Broadcast disks [2] attempt to reduce response time by repeating popular items in broadcasts. By broadcasting additional copies of popular data items, clients have more opportunities to retrieve the information they desire. Data items are grouped into several, separate "disks", where the size and speed of a spinning disk reflects the popularity of the items on that disk. For example, a disk with hot items is smaller and spins faster than a disk with cold items. As the disks spin, data items are read from each disk and are interleaved to create

a broadcast. Data items from faster spinning disks appear more frequently in broadcasts, allowing users faster access to hot data.

To further reduce the average response time for broadcast disks, one can tune several system parameters including disk sizes and angular velocities. Liaskos et al. [18] mathematically modeled clients and their use of broadcast disks, facilitating analytical determination of optimal system parameters. Simulation results showed that analytically calculated parameters outperformed those found heuristically. However, the model assumes that the server has extensive historical knowledge of client requests in order to compute the probability density function that models client behavior. Incomplete knowledge or deviations from the model may result in poor performance in terms of response time.

2.3 Hybrid Methods

Hybrid methods combine features of both push-based and pull-based methods. Like push-based methods, clients can retrieve data items from the air without submitting a request to the server. However, if a requested item is not available via broadcast, clients can directly retrieve it from the server through a unicast communication channel. The requests for data items can then be analyzed to dynamically change the contents of broadcasts. By including the most popular data items in broadcasts, the goal is to satisfy a large number of client requests while reducing the number of requests that must be submitted to the server. Unlike push-based methods, it is not necessary to broadcast all items in the database. Clients seeking unpopular data items have an alternate communication channel for retrieving data, at the expense of additional energy expenditure.

Stathatos et al. [24] developed a hybrid environment that treats broadcast content like a global cache. Like typical cache memory, clients first check the "air cache" for a data item. If there is a cache miss, a request is submitted to the server so that the item can be retrieved directly. Based on these cache misses, the server determines which data items are broadcast. The authors use the states of water as an analogy for the states of data items. That is, data items can be frozen, liquid, or vapor.

As requests for a data item arrive, the temperature of that data item increases. If the temperature surpasses a certain threshold, it transitions from frozen to liquid. Similarly, at a higher temperature threshold, a data item transitions from liquid to vapor. Data items in the vapor state are popular enough to be included in broadcasts. Liquid and frozen items are still obtainable, but they must be requested directly. Eventually, vapor data items may become less popular, but the lack of requests for these broadcast items makes it difficult to determine their popularity. By gradually cooling down data items, those in the vapor state will become liquid and cease being broadcast. If they are popular enough to become vapor again, incoming requests will heat them back to that state. The temperature at which a data item becomes vapor is configurable, allowing for adjustment of the number of broadcast data items.

Similar to the cutoff between vapor and liquid/frozen data items, Guo et al. [9] suggests partitioning data items into those that are broadcast and those that are not. In this scheme, incoming requests are used to compute the access probability for each data item in the database. A threshold, K, is then used to partition the data items into two sets according to their access probabilities. The server broadcasts the K data items with the highest probability of being accessed. Based on the probabilities, the cutoff point K is computed, so that the optimal expected access time can be achieved.

3. Indexing Objects

Indices provide a fast way to determine the location, e.g., channel and frame, of one or more data items on a multi-channel broadcast. Given a key, an index can provide the address of the data described by that key. In file systems and databases, indices are used to speed up search. Indices can quickly reduce the size of a key's search space, which results in fast address retrieval.

Broadcasts can also utilize indices. Indices enable mobile devices to determine the temporal location of items requested by a user. That is, a device can determine when and on which channel a requested item will be broadcast by the server. Using this information, the device can decide when to switch its wireless radio to a power-saving mode and, if necessary, when to switch channels to fetch upcoming data items.

3.1 Signature-based Indexing

A *signature-based index* is an abstract representation of data in a broadcast [17]. Broadcasts are broken into records, which are used to create signatures. Each value stored in a record is hashed and combined to form the signature for that record. To determine whether a record contains a given value, a device hashes the value and compares it with the signature. If the hash is contained within the signature, then the value *may be* in the corresponding record. There is a possibility for false positives. However, if the hash is *not* contained within the signature, then the value is definitely not in the corresponding record.

The records used to create signatures vary in size depending on the placement of indices in the broadcast. Indices may be interleaved according to a simple, integrated, or multilevel scheme. The *simple* scheme creates a signature for each data frame. Each data frame follows its signature frame. The *integrated* scheme creates signatures for groups of frames. Like the simple scheme, signature frames precede their group of data frames. Finally, the *multilevel* scheme creates multiple levels of signatures - the upper level is an integrated signature, and the lower level is a simple signature. Simulations have shown that the multilevel scheme offers the shortest tune-in time but the longest access time, while the integrated scheme offered the shortest access time but the longest tune-in time [17]. Additionally, all three schemes offered improved tune-in time and access time over broadcasting without indices.

3.2 Tree-based Indexing

It is often convenient to organize data based on the relationship between its values. Organized data is easier to navigate and can accelerate searches. In many cases, a tree is a useful data structure for capturing relationships between data. *Tree-based indexing* does this for broadcasts; each leaf in the index tree points

to a data item in a broadcast.

Tree indices are useful for providing a global view of a broadcast based on the tree's index key. This feature makes tree indices suitable for reducing tune-in time. By indexing signature-based indices with a sparse tree, it is possible to maintain the benefits of signature-based indices, while achieving shorter tune-in times [16]. Clients search the sparse index to retrieve an approximate location of their desired data. Signatures can then be used to further determine which records to inspect.

As with signature-based indices, there are several ways to place tree-based indices within a broadcast. It makes sense to periodically include an index within a broadcast channel for the benefit of clients that join mid-broadcast. $(1, m)$ indexing and distributed indexing interleave index information within each broadcast [13]. $(1, m)$ indexing inserts a complete index every $\frac{1}{m}$ of a broadcast cycle, totaling m copies of the same index. However, this method replicates information and increases response time as a result. Distributed indexing trims out some of the duplicate index information before adding indices to the broadcast. Analysis suggests that $(1, m)$ indexing offers improved power consumption at the cost of slower response times, while distributed indexing offers faster response time at a higher energy cost.

The placement of indices within parallel channel broadcasts has also been studied. Three different schemes varying in degrees of replication have been tested using a tree-based index inspired by the B^+ tree [27]. Non-replicated indexing splits the tree into several smaller, disjoint index trees. Each sub-tree is broadcast on its own channel, so the client may need to check several channels to retrieve the correct index. Partially-replicated indexing splits the tree into several smaller index trees, but a sub-tree may in turn refer to other sub-trees. Each sub-tree is broadcast on its own channel, and the links between sub-trees maintain the structure of the original tree. Fully replicated indexing duplicates the original index tree on all index channels. As long as the client is able to determine the correct index channel to use, the non-replicated index offers the lowest index access time of the three. When this is not possible, partially-replicated indexing offers good performance, while providing clients with information about the other index channels.

3.3 Hash-based Indexing

Several indexing techniques, including signature-based indexing, take advantage of hash functions. MHash is a hash function developed specifically for indexing and organizing broadcast contents [30]. It accepts two parameters: the item's key, k, and its sequential identifier, l. The sequential identifier allows an item to be repeated in a broadcast. The hash function is used by the server to determine the location of a specific data item within the broadcast. Hash collisions are handled by sorting the colliding items by their access probability. The item that is most likely to be accessed is placed in the broadcast frame indicated by the hash function. The other colliding items are placed in open frames in order of decreasing access probability.

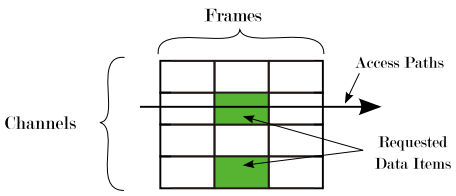
To access data from a broadcast, the client must know the hash function, the length of the broadcast cycle, and the maximum number of times a broadcast item can be repeated. Using this information, the client can calculate the possible locations of desired data items. Simulation results show that MHash outperforms distributed tree indexing in terms of energy consumption, tune-in time, and access time.

4. Access Conflicts

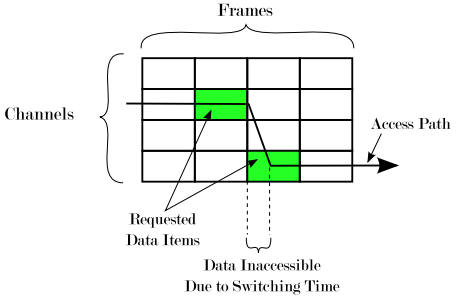
Wireless radios on many devices are only able to read only a single channel at a time. This fact leads to some difficulty when accessing data items from a parallel broadcast. Namely, it leads to access conflicts.

Two data items are said to be in *conflict* if a device cannot read both of them during the same pass through a broadcast [12]. As shown in **Fig. 2** (a), conflicts can arise when multiple requested data items are broadcast simultaneously on different channels. A device must choose one channel to read, fetching the data item on that channel and leaving the other conflicting items unread. These unread, requested items can still be acquired by reading repeated transmissions of the same broadcast.

Conflicts can also be caused by channel switches. Channel switches are not instantaneous; that is, a non-zero amount of time is required to switch between channels. At the end of a frame, it is not possible to switch channels quickly enough to be tuned in to a different channel before the next frame has begun. As shown in Fig. 2 (b), a portion of the next data item may have been transmitted while a device's radio is in the process of switching. The lost portion of the data item renders the remaining portion useless. Thus, it is impossible to read two consecutively transmitted data items that are in different channels. In other words, any two consecutively transmitted data items are in conflict, unless they are in the same channel. As is the case with simultaneously broadcast data items, adjacent data items on different channels can be read during subsequent transmissions of the same broadcast.



(a) Conflicting data items in the same frame



(b) Conflicting data items in adjacent frames

Fig. 2 Requested data items in conflict.

5. Data Retrieval

5.1 Single-channel Broadcast

Retrieving data from a single-channel broadcast is a straightforward process. Based on data item locations specified by an index, devices tune in to a broadcast when requested items are available on the air. When unrequested items are being broadcast, devices are free to switch their radios into an "idle" or "sleep" mode to conserve energy. This process of fetching data items from a single-channel broadcast is simple, but single channel broadcasts have several drawbacks. Namely, a large number of data items invariable results in a long broadcast. Longer broadcasts increase response time and may require more energy, as wireless radios may remain in an active state for longer periods of time.

5.2 Multi-channel Broadcast

Like single-channel broadcasts, devices retrieve data items from parallel broadcasts using an index to determine their locations within a broadcast and tuning in to retrieve them. However, parallel broadcasts may require a device to tune its radio to several different wireless channels to obtain data items. To ensure all requested items are retrieved, a device must be able to compute schedules that dictate when to switch between wireless channels and when to read requested items [12]. Additionally, these schedules must be able to resolve conflicts between requested data items.

Several algorithms exist that can compute channel switching schedules for retrieving data from parallel broadcasts. Based on the set of requested data items, these algorithms generate a set of access paths that indicate when a device should tune in to different channels and read data items. In the remainder of this paper, we present one simple and several more sophisticated algorithms for generation of access schedules.

5.2.1 Row Scan

One naïve algorithm that computes access paths through parallel broadcast channels is known as *row scan* [15]. Row scan produces simple schedules that iterate through available broadcast channels. It only considers which channels contain requested data items, creating an access path through each channel that contains a requested item (see **Fig. 3**). As the device iterates through each channel, according to the schedule, it is free to read requested items from the current channel when they are available.

Row scan is simple to implement, as its access paths simply iterate through the broadcast channels. It can be written as a `for` loop that switches channels after each broadcast pass. Because row scan is such a simple algorithm, it requires few CPU operations to generate a set of access paths. Fewer CPU operations implies that less energy is required to generate the schedule. From the CPU's perspective, row scan is an efficient method for generating access paths.

However, from the wireless radio's perspective, row scan is an inefficient approach. Its access paths naïvely pass through every frame of any channel that contains a requested item. As a result, a device wastes time tuned in to channels populated with many unrequested data items. If the number of requested items is sparse enough, it is possible that a device might even tune in to a channel that contains only one item of interest. The time wasted on unrequested data items translates into longer response times for the user and higher energy consumption.

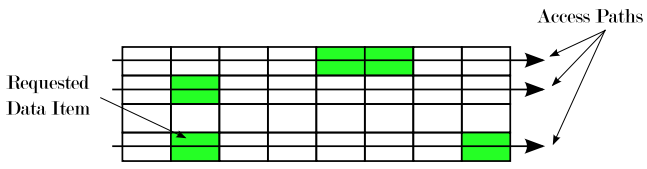


Fig. 3 An example of access paths produced by row scan.

5.2.2 Maximum Cut

Every additional broadcast pass increases response time and energy consumption for client devices. The *maximum cut* [25] of a broadcast refers to the minimum number of broadcast passes required to read all requested data items. A cut is defined as the number of requested data items in conflict for a given broadcast frame (column). That is, the cut for a frame j is the number of channels containing requested items in frames j and $j + 1$. This accounts for conflicts caused by simultaneous broadcast, as well as those caused by data items in frames that are adjacent in time, but located on different channels. In the example of **Fig. 4**, the largest cut value among all columns is two.

To elaborate, the cut of a frame counts the number of access paths required to fetch every requested data item in both the current frame and the next frame. Each channel containing a requested item in frame j must have its own path. Since switches are not instantaneous, any channel with a requested item in frame $j + 1$ that is not covered by the paths for frame j must also have its own path. Thus, due to conflicts, every channel that contains a requested item in frame j or $j + 1$ must have a dedicated access path.

The frame with the maximum cut value determines the minimum number of access paths required. In other words, the maximum cut provides a numerical minimum number of broadcast passes required to retrieve all data items. If fewer access paths are used, then some requested data items will remain unread. Sophisticated algorithms leverage a broadcast's maximum cut when computing access paths to improve response time and energy efficiency.

5.2.3 Parallel Object Scan

Leveraging a broadcast's maximum cut, the *parallel object scan* (*POS*) [25] technique determines the minimum number of access paths required fetch requested data items from a broadcast. POS iterates though the frames of a broadcast, simultaneously constructing access paths. Though the paths are constructed simultaneously, they are not used simultaneously. The mobile device will use each access path one at a time to fetch data items.

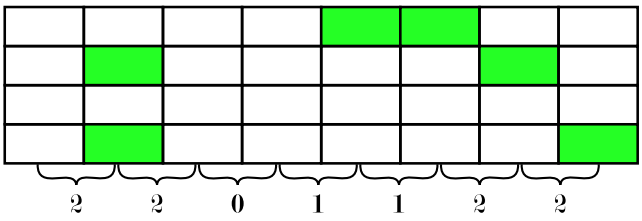


Fig. 4 A broadcast with maximum cut of two.

POS follows a set of rules that attempt to reduce the number of required channel switches. Before iterating through a broadcast's frames, POS initializes its access paths. Each access path is set up such that the data items in the earliest frames are read first. Then, the algorithm begins iterating through the frames of the broadcast, following several rules. For each access path, at frame j:, one of the following actions will be taken:

Continue If the data item on a path's current channel in frame $j+1$ is a requested item, that path should continue reading the same channel. This way, the requested item in frame $j+1$ will be read without performing any switches.

Switch If there is a requested data item on channel c in frame $j+2$, and no path is currently tuned into channel c, some path will need to switch to channel c. The path whose current channel will not encounter a requested item for the longest time should be switched. The idea is that this path will not be reading any items anyway, so it is free to switch to another channel.

Default If a path has not yet been covered by the **Continue** or **Switch** rules, it should continue reading from its current channel.

These rules are applied in the order listed above. Meaning, if an access path meets the criteria for the **Continue** rule, it will only behave according to that rule. If an access path does not meet the criteria for **Continue**, then it will be evaluated according to the criteria for **Switch**. An access path will only perform the **Default** behavior if it has not already taken the **Continue** or **Switch** action.

The **Continue** and **Default** rules attempt to reduce the number of channel switches required. **Continue** does this by retrieving upcoming requested data items using paths already tuned into corresponding channels, and **Default** adds no additional switches when paths are not immediately needed. **Switch** performs channel switches when necessary to cover requested data items in channels into which no access paths are currently tuned. **Figure 5** depicts an example where POS requires fewer paths than row scan to access all requested data items.

5.2.4 Serial Empty Scan

While POS approaches access path construction by considering upcoming requested items, *serial empty scan* (*SES*) [25] instead considers groups of *unrequested* data items. SES attempts to generate efficient access paths by logically rearranging broadcasts, so that groups of unrequested data items, known as "empty blocks," can be eliminated from consideration. Like POS, SES leverages a broadcast's maximum cut to minimize the number of access paths required.

SES begins by creating a logical representation of the broadcast. This includes locations of all requested and unrequested items. Channels containing zero requested items are immediately eliminated from the logical broadcast. That is, the rows representing those channels are removed from the logical representation of the broadcast. Next, the algorithm sets a pointer to the first frame (column) and identifies the channel that contains the longest empty block that starts at the pointer. From here, the pointer is moved to the end of the longest empty block, and the channel with the longest empty block overlapping the pointer's new position is located. This overlapping block is tagged, so that logical access paths can eventually be transformed into access paths for the original broadcast. Data items are swapped between the two overlapping areas, collecting requested data items in one of the channels and forming a larger empty block in the other. The pointer is moved further down the broadcast, and the next overlapping empty block is located. The process of locating and swapping empty blocks repeats until the pointer reaches the end of the broadcast. At this point, one logical channel will contain zero requested data items. This empty row can be eliminated from the broadcast. This process repeats until the number of remaining channels is equal to the maximum cut of the original broadcast. Each logical channel corresponds to a single logical access path. Based on the tags added to overlapping blocks, the set of logical access paths is converted into a set of access paths for the original, physical broadcast. In the example shown in **Fig. 6**, SES produces the same number of paths and channel switches as POS does in Fig. 5. SES attempts to reduce the number of channel switches used in its generated set of access paths by searching for the longest overlapping empty blocks.

5.2.5 Least Switch

If a client device has an advanced wireless radio that is capable of reading from more than one channel at a time, the data access problem changes substantially. A multiple input/multiple output (MIMO) wireless radio can tune in to several channels at once [14]. A variable number of processes is used to read from this type of radio. If the number of channels containing requested items, K, is equal to the number of processes reading from the air, M, then scheduling is not necessary. A device can simply tune into those M channels and read every data item in one pass. Otherwise, a device with a MIMO radio will still need to schedule its channel accesses.

The *least-switch* algorithm [23] works in two phases. Phase one identifies which of the M channels to read by choosing those

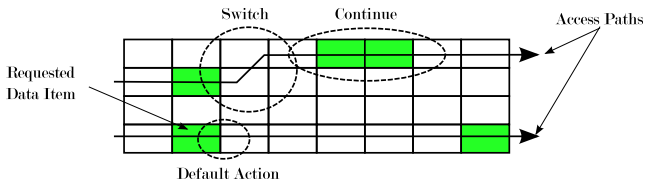


Fig. 5 An example of access paths generated by POS.

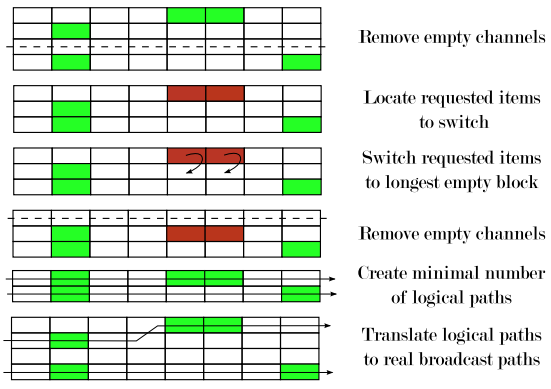


Fig. 6 An example of access paths generated by SES.

channels whose requested items can be read in the shortest time. Once a process finishes reading all the requested items from a channel, it switches to the channel that will take the longest to read all requested data items. The channel that takes the longest to read (starting from the beginning of the broadcast) determines the shortest amount of time in which a broadcast can be read. By switching from the channel with the shortest time to the channel with the longest, the hope is to reduce the impact of the channel that requires a longer time to fetch data items. The least-switch algorithm also attempts to conserve energy by reducing the number of channel switches required of the client. Depending on the number of reading processes and the number and locations of requested items, multiple broadcast passes may still be required to fetch all requested data items.

5.2.6 Best First

Least-switch works by considering the location of the last requested item in a channel. The location of this item indicates how long it would take to read all requested items from a channel starting from the beginning of the broadcast. The downside to this approach is that there may be large gaps between requested items in a given channel. During this downtime, a reading process may have the opportunity to switch to another channel, gather information, and switch back before the next requested data item is available.

The *best-first* approach attempts to address this issue [23]. The algorithm begins by assigning the M access paths to the channels that have the earliest requested items. Then, for each column, the paths that will incur the least cost to fetch the data items will switch channels to retrieve them. Costs are computed based on an access path's current channel and the locations of upcoming requested items. Finally, paths will switch channels according to the need for additional broadcast passes.

6. Simulation

Several studies have analyzed the performance of different data retrieval algorithms to verify their behavior. Specifically, the performance of POS, SES, and a tree-based method [12] were evaluated in a simulated broadcasting environment [25]. The *tree-based* method utilizes a set of heuristics designed to reduce the number of broadcast passes needed to read requested items from a broadcast. It builds a tree of possible paths and prunes away options by following three prioritized conditions:

1. Eliminate the maximum number of conflicts
2. Retrieve the maximum number of requested items per broadcast pass
3. Reduce the number of required channel switches

A database of 4290 NASDAQ securities was used as the data source in the simulated environment. Broadcasts were constructed on parallel channels using N channels and M frames. Each simulation run consisted of user requests for K random data items from the broadcast. Performance metrics were calculated based on the average of 1,000 simulation runs for each configuration. Additionally, the simulated mobile device uses a wireless radio that is capable of reading from a single channel. Channel switching methods for MIMO antennae (discussed above in Sections 5.2.5 and 5.2.6) were not simulated.

The number of broadcast passes (access paths) used by each algorithm is depicted in **Fig. 7**. Both POS and SES consistently require the same number of access paths. These algorithms calculate a broadcast's maximum cut to ensure that the minimum number of broadcast passes can be used. Though the tree-based algorithm attempts to reduce the number of broadcast passes, it does not always achieve the minimum.

Eventually, the curves for POS and SES saturate at the number of channels in the broadcast (N). At this point, the maximum cut is equal to the number of channels, so these algorithms default to a row scan behavior. The tree-based algorithm, however, continues to use a greater number of access paths. The access paths generated by this approach re-read at least one channel to retrieve data items. The additional re-reads add to the overall response time of the access paths, which is depicted in **Fig. 8**.

This effect of defaulting to row scan behavior is apparent in the resulting number of channel switches, as seen in **Fig. 9**. As the number of requested items increases, the number of channel switches used by POS and SES begins to settle. At this point, POS and SES construct an access path through every channel, so exactly $N - 1$ switches are used. Again, the tree-based algorithm attempts to reduce the number of channel switches, but it requires

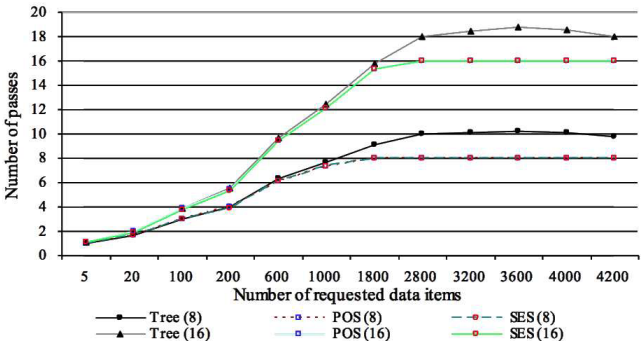


Fig. 7 Number of broadcast passes for 8 and 16 channels.

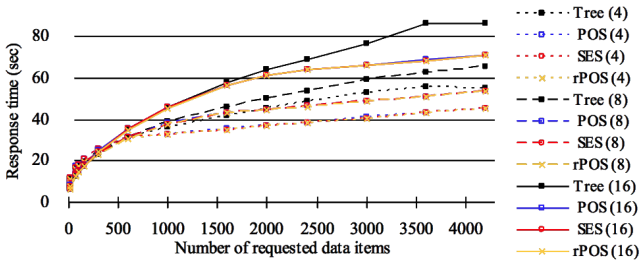


Fig. 8 Request response times for 8 and 16 channels.

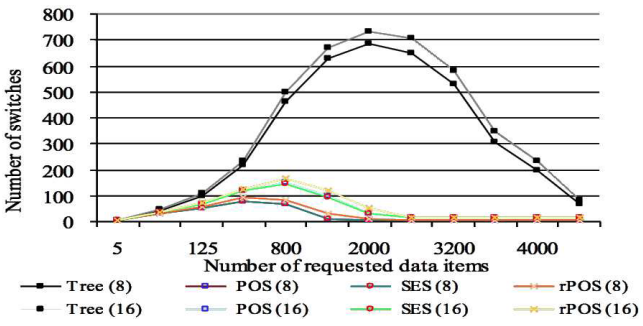


Fig. 9 Number of channel switches for 8 and 16 channels.

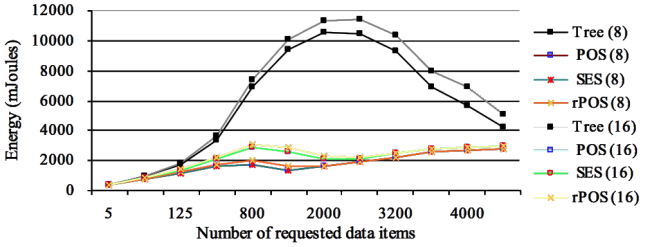


Fig. 10 Energy consumption for 8 and 16 channels.

a drastically greater number of switches as compared to POS or SES.

Using the average energy consumed by different wireless modes (active, doze) and channel switches, the energy consumption of the algorithms was computed based on Eq. (3). The *TuneInTime* represents the amount of time the device spends actively tuned in to a broadcast. Thus, the difference between the total response time (*ResponseTime*) and the *TuneInTime* yields the amount of time the device spent in doze mode. Multiplying by their respective average power consumption gives the amount of energy consumed in active and doze mode. Additionally, according to Ref. [12], each channel switch expends 10% of the active mode energy.

Figure 10 shows the energy consumed as the number of requested items increases. Because POS and SES utilize the same number of broadcast passes, the difference between the energy consumption of the two can be attributed to differences in channel switching. Additionally, based on the shape of the curves, one conclude that channel switching is a major source of energy consumption for the tree-based heuristic.

$$\begin{aligned} \text{Energy} = {} & (ResponseTime - TuneInTime) \times SleepMode \\ & + TuneInTime \times ActiveMode \\ & + Numberof Switches \times 10\% \times ActiveMode \end{aligned} \tag{3}$$

According to these simulations, SES and POS perform very similarly with respect to the number of broadcast passes and channel switches required to fetch requested data items. Both are guaranteed to use the minimum number of passes while reducing the number of switches required. The tree-based method, on the other hand, does not have these features. At times, paths generated with the tree-based method exceed the number of broadcast passes required by row scan (at most N). Although the heuristics used by the tree-based method are similar to those used by POS and SES, there are significant energy and response time advantages to using POS or SES instead.

Although the end result of POS and SES are very similar, the algorithms are quite different. SES likely incurs greater computational overhead than POS, as it must maintain and manipulate a logical copy of the broadcast in order to determine access paths. Additional computation required by SES may lead to longer execution time on the mobile device's CPU, which increases response time and CPU energy consumption. Further study is required to determine the impacts of this overhead on energy consumption and overall response time.

7. Conclusion

Disseminating public data to mobile clients can be a challenging task. Not only do users want timely responses to their requests, but they expect their devices to maintain long battery lives. Broadcasting has been proposed as an efficient method to make available to users the public data that they desire. With broadcasting, clients do not need to expend energy to send requests for data because data is always available on the air.

However, several issues should be considered when constructing a broadcasting environment. The structure and contents of the broadcast must be decided, so that clients can easily obtain data from the broadcast channel or channels. Additionally, indices of data items should be included in broadcasts so that clients know when to tune in to fetch data. These indices should provide a useful amount of information without significantly impacting response time. Once clients know the locations of requested items, they need a way to quickly retrieve them from the air. Sophisticated access algorithms should be able to schedule access to data items in a way that leverages a device's available technology and reduces response time and energy consumption.

Many applications that rely on disseminating data from a server to clients can benefit from broadcasting. For example, mobile robots that cooperate to accomplish a common goal might benefit from broadcasting. Tasks, environmental information, or other data could be transmitted to an area over broadcast channels. When a robot needs to refresh information, it can fetch data off the air instead of expending energy to submit requests.

Additionally, applications that require scaling to a large number of users can benefit from broadcasting. Natural disasters or other events that lead to sudden spikes of user activity may cause on-demand systems to slow down or fail. Broadcasting avoids this issue, as all communication is downstream from the server to the clients. As long as clients are within the range of the broadcast, they can retrieve data from the air.

References

[1] Acharya, S., Franklin, M. and Zdonik, S.: Dissemination-based data delivery using broadcast disks, *IEEE Personal Communications*, Vol.2, No.6, pp.50–60 (online), DOI: 10.1109/98.475988 (1995).
[2] Acharya, S., Alonso, R., Franklin, M. and Zdonik, S.: Broadcast Disks: Data Management for Asymmetric Communication Environments, *SIGMOD Rec.*, Vol.24, No.2, pp.199–210 (1995).
[3] Acharya, S., Franklin, M. and Zdonik, S.: Balancing Push and Pull for Data Broadcast, *SIGMOD Rec.*, Vol.26, No.2, pp.183–194 (1997).
[4] Aksoy, D. and Franklin, M.: R × W: A Scheduling Approach for Large-scale On-demand Data Broadcast, *IEEE/ACM Trans. Netw.*, Vol.7, No.6, pp.846–860 (online), DOI: 10.1109/90.811450 (1999).
[5] Albers, S., Müller, F. and Schmelzer, S.: Speed Scaling on Parallel Processors, *Algorithmica*, Vol.68, No.2, pp.404–425 (2014).
[6] Blem, E., Menon, J. and Sankaralingam, K.: Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures, *2013 IEEE 19th International Symposium on High Performance Computer Architecture* (*HPCA2013*), pp.1–12 (2013).
[7] Chow, C.-Y., Leong, H. and Chan, A.: Cache signatures for peer-to-peer cooperative caching in mobile environments, *18th International Conference on Advanced Information Networking and Applications* (*AINA 2004*), Vol.1, pp.96–101 (2004).
[8] Dai, C., Chow, C.-Y., Leong, H.V. and Chan, A.: An analytical study of cooperative data dissemination in push-based mobile environments, *2013 8th International ICST Conference on Communications and Networking in China* (*CHINACOM*), pp.569–574 (2013).
[9] Guo, Y., Das, S.K. and Pinotti, C.M.: A New Hybrid Broadcast Scheduling Algorithm for Asymmetric Communication Systems:

Push and Pull Data Based on Optimal Cut-off Point, *Proc. 4th ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (*MSWIM '01*), pp.123–130, ACM (online), DOI: 10.1145/381591.381630 (2001).
[10] Harb, A.: Energy harvesting: State-of-the-art, *Renewable Energy*, Vol.36, No.10, pp.2641–2654 (2011).
[11] Hurson, A.R., Jiao, Y. and Shirazi, B.: Broadcasting a means to disseminate public data in a wireless environment—Issues and solutions, *Advances in Computers*, Vol.67, pp.1–84 (2006).
[12] Hurson, A.R., Muñoz-Avila, A.M., Orchowski, N., Shirazi, B. and Jiao, Y.: Power-aware data retrieval protocols for indexed broadcast parallel channels, *Pervasive and Mobile Computing*, Vol.2, No.1, pp.85–107 (2006).
[13] Imielinski, T., Viswanathan, S. and Badrinath, B.R.: Data on air: Organization and access, *IEEE Trans. Knowledge and Data Engineering*, Vol.9, No.3, pp.353–372 (1997).
[14] Jensen, M. and Wallace, J.: A review of antennas and propagation for MIMO wireless communications, *IEEE Trans. Antennas and Propagation*, Vol.52, No.11, pp.2810–2824 (2004).
[15] Juran, J., Hurson, A.R., Vijaykrishnan, N. and Kim, S.: Data organization and retrieval on parallel air channels: Performance and energy issues, *Wirel. Netw.*, Vol.10, No.2, pp.183–195 (2004).
[16] Lee, D.L., Hu, Q. and Lee, W.-C.: Power conserving and access efficient indexes for wireless computing, *Information Organization and Databases*, pp.249–263, Springer (2000).
[17] Lee, W.-C. and Lee, D.L.: Using signature techniques for information filtering in wireless and mobile environments, *Distributed and Parallel Databases*, Vol.4, No.3, pp.205–227 (1996).
[18] Liaskos, C., Petridou, S., Papadimitriou, G.I., Nicopolitidis, P. and Pomportsis, A.S.: On the Analytical Performance Optimization of Wireless Data Broadcasting, *IEEE Trans. Vehicular Technology*, Vol.59, No.2, pp.884–895 (online), DOI: 10.1109/TVT.2009.2035357 (2010).
[19] Min, R. and Chandrakasan, A.: MobiCom Poster: Top Five Myths About the Energy Consumption of Wireless Communication, *SIGMOBILE Mob. Comput. Commun. Rev.*, Vol.7, No.1, pp.65–67 (2003).
[20] Polastre, J., Szewczyk, R. and Culler, D.: Telos: Enabling ultra-low power wireless research, *4th International Symposium on Information Processing in Sensor Networks* (*IPSN 2005*), pp.364–369 (2005).
[21] Polatoglou, M., Nicopolitidis, P. and Papadimitriou, G.I.: On low-complexity adaptive wireless push-based data broadcasting, *International Journal of Communication Systems*, Vol.27, No.1, pp.194–200 (2014).
[22] Schulman, A., Navda, V., Ramjee, R., Spring, N., Deshpande, P., Grunewald, C., Jain, K. and Padmanabhan, V.N.: Bartendr: A Practical Approach to Energy-aware Cellular Data Scheduling, *Proc. 16th Annual International Conference on Mobile Computing and Networking* (*MobiCom '10*), pp.85–96, ACM (2010).
[23] Shi, Y., Gao, X., Zhong, J. and Wu, W.: Efficient Parallel Data Retrieval Protocols with MIMO Antennae for Data Broadcast in 4G Wireless Communications, *Database and Expert Systems Applications*, Bringas, P., Hameurlain, A. and Quirchmayr, G. (Eds.), Lecture Notes in Computer Science, Vol.6262, pp.80–95, Springer Berlin Heidelberg (2010).
[24] Stathatos, K., Roussopoulos, N. and Baras, J.S.: Adaptive data broadcast in hybrid networks, Technical report, DTIC Document (1997).
[25] Sun, B., Hurson, A. and Hannan, J.: Energy-efficient scheduling algorithms of object retrieval on indexed parallel broadcast channels, *International Conference on Parallel Processing* (*ICPP 2004*), Vol.1, pp.440–447 (2004).
[26] Tsukayama, H.: Twitter turns 7: Users send over 400 million tweets per day, *The Washington Post* (2013).
[27] Waluyo, A., Srinivasan, B. and Taniar, D.: A taxonomy of broadcast indexing schemes for multi channel data dissemination in mobile databases, *18th International Conference on Advanced Information Networking and Applications* (*AINA 2004*), Vol.1, pp.213–218 (online), DOI: 10.1109/AINA.2004.1283913 (2004).
[28] Wong, J.: Broadcast delivery, *Proc. IEEE*, Vol.76, No.12, pp.1566–1577 (online), DOI: 10.1109/5.16350 (1988).
[29] Xu, J., Tang, X. and Lee, W.-C.: Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation, *IEEE Trans. Parallel and Distributed Systems*, Vol.17, No.1, pp.3–14 (online), DOI: 10.1109/TPDS.2006.14 (2006).
[30] Yao, Y., Tang, X., Lim, E.-P. and Sun, A.: An energy-efficient and access latency optimized indexing scheme for wireless data broadcast, *IEEE Trans. Knowledge and Data Engineering*, Vol.18, No.8, pp.1111–1124 (online), DOI: 10.1109/TKDE.2006.118 (2006).

Michael Wisely received B.S. degrees in Computer Science and Computer Engineering from the Missouri University of Science and Technology in 2012. He is currently a Ph.D. candidate in Computer Science at Missouri S&T, where he is a GAANN Fellow and a Chancellor's Fellow. His research interests include traffic modeling and distributed computing. Michael is a member of IEEE and ACM.

Sahra Sedigh Sarvestani received B.S.E.E. degree from Sharif University of Technology in 1995, and M.S.E.E. and Ph.D. degrees from Purdue University, in 1998 and 2003, respectively. She subsequently joined the Missouri University of Science and Technology, where she is currently an Associate Professor of Electrical and Computer Engineering. Her research centers on development and modeling of dependable networks and systems, with focus on critical infrastructure. She is a Fellow of the National Academy of Engineering's Frontiers of Engineering Education Program and held a Purdue Research Foundation Fellowship from 1996 to 2000. She is a member of HKN and ACM and a senior member of IEEE.

Ali R. Hurson received B.S. degree in Physics from the University of Tehran in 1970, M.S. degree in Computer Science from the University of Iowa in 1978, and Ph.D. from the University of Central Florida in 1980. He was a Professor of Computer Science at the Pennsylvania State University until 2008, when he joined the Missouri University of Science and Technology. He has published over 300 technical papers in areas including multidatabases, global information sharing and processing, computer architecture and cache memory, and mobile and pervasive computing. He serves as an ACM distinguished speaker, area editor of the CSI Journal of Computer Science and Engineering, and Co-Editor-in-Chief of Advances in Computers. He is a senior member of IEEE.