Data Dissemination

Data Dissemination and Synchronization : Communications Asymmetry, Classification of Data Delivery Mechanisms, Data Dissemination, Broadcast Models, Selective Tuning and Indexing Methods, Data Synchronization – Introduction, Software, and Protocols.

Ongoing advances in communications including the proliferation of internet, development of mobile and wireless networks, high bandwidth availability to homes have led to development of a wide range of new-information centered applications. Many of these applications involve data dissemination, i.e. delivery of data from a set of producers to a larger set of consumers.

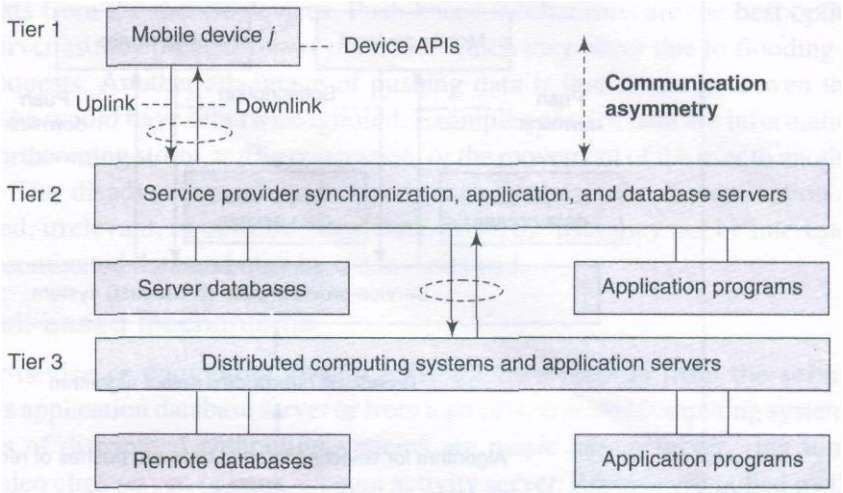
Data dissemination entails distributing and pushing data generated by a set of computing systems or broadcasting data from audio, video, and data services. The output data is sent to the mobile devices. A mobile device can select, tune and cache the required data items, which can be used for application programs.

Efficient utilization of wireless bandwidth and battery power are two of the most important problems facing software designed for mobile computing. Broadcast channels are attractive in tackling these two problems in wireless data dissemination. Data disseminated through broadcast channels can be simultaneously accessed by an arbitrary number of mobile users, thus increasing the efficiency of bandwidth usage.

Communications Asymmetry

One key aspect of dissemination-based applications is their inherent communications asymmetry. That is, the communication capacity or data volume in the downstream direction (from servers-to-clients) is much greater than that in the upstream direction (from clients-to-servers). Content delivery is an asymmetric process regardless of whether it is performed over a symmetric channel such as the internet or over an asymmetric one, such as cable television (CATV) network. Techniques and system architectures that can efficiently support asymmetric applications will therefore be a requirement for future use.

Mobile communication between a mobile device and a static computer system is intrinsically asymmetric. A device is allocated a limited bandwidth. This is because a large number of devices access the network. Bandwidth in the downstream from the server to the device is much larger than the one in the upstream from the device to the server. This is because mobile devices have limited power resources and also due to the fact that faster data transmission rates for long intervals of time need greater power dissipation from the devices. In GSM networks data transmission rates go up to a maximum of 14.4 kbps for both uplink and downlink. The communication is symmetric and this symmetry can be maintained because GSM is only used for voice communication.



Communication asymmetry in uplink and downlink and participation of device APIs and distributed computing systems when an application runs

The above figure shows communication asymmetry in uplink and downlink in a mobile network. The participation of device APIs and distributed computing systems in the running of an application is also shown.

Classification of Data-Delivery Mechanisms

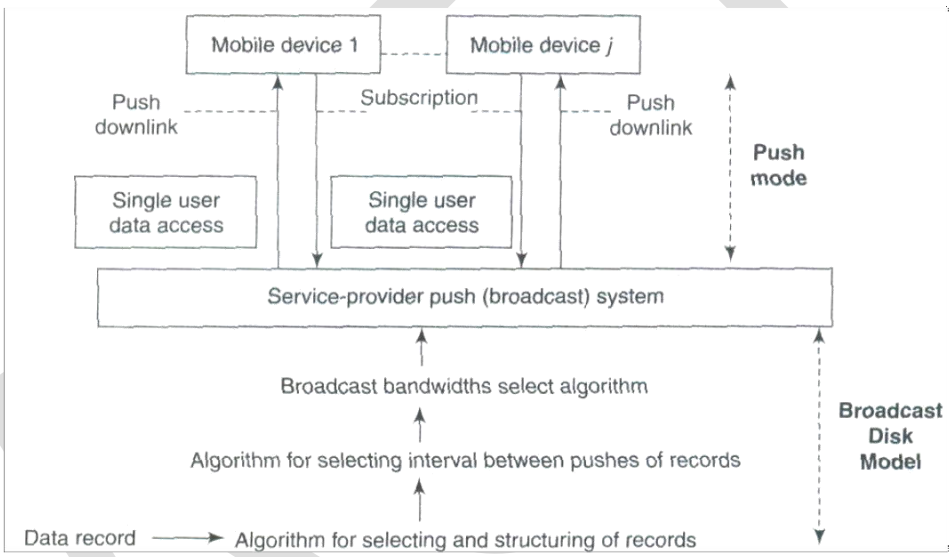
There are two fundamental information delivery methods for wireless data applications: Point-to-Point access and Broadcast. Compared with Point-to-Point access, broadcast is a more attractive method. A single broadcast of a data item can satisfy all the outstanding requests for that item simultaneously. As such, broadcast can scale up to an arbitrary number of users.

There are three kinds of **broadcast models**, namely *push-based* broadcast, *On-demand* (or *pull-based*) broadcast, and *hybrid* broadcast. In push based broadcast, the server disseminates information using a periodic/aperiodic broadcast program (generally without any intervention of clients). In on demand broadcast, the server disseminates information based on the outstanding requests submitted by clients; In hybrid broadcast, push based broadcast and on demand data deliveries are combined to complement each other. In addition, mobile computers consume less battery power on monitoring broadcast channels to receive data than accessing data through point-to-point communications.

Data-delivery mechanisms can be classified into three categories, namely, push-based mechanisms (publish-subscribe mode), pull-based mechanisms (on-demand mode), and hybrid mechanisms (hybrid mode).

Push-based Mechanisms

The server pushes data records from a set of distributed computing systems. Examples are advertisers or generators of traffic congestion, weather reports, stock quotes, and news reports. The following figure shows a push-based data-delivery mechanism in which a server or computing system pushes the data records from a set of distributed computing systems. The data records are pushed to mobile devices by broadcasting without any demand. The push mode is also known as ***publish-subscribe*** mode in which the data is pushed as per the subscription for a push service by a user. The subscribed query for a data record is taken as perpetual query till the user unsubscribe to that service. Data can also be pushed without user subscription.



Push-based data-delivery mechanism

Push-based mechanisms function in the following manner:

1. A structure of data records to be pushed is selected. An algorithm provides an adaptable multi-level mechanism that permits data items to be pushed uniformly or non-uniformly after structuring them according to their relative importance.
2. Data is pushed at selected time intervals using an adaptive algorithm. Pushing only once saves bandwidth. However, pushing at periodic intervals is important because it provides the devices that were disconnected at the time of previous push with a chance to cache the data when it is pushed again.
3. Bandwidths are adapted for downlink (for pushes) using an algorithm. Usually higher bandwidth is allocated to records having higher number of subscribers or to those with higher access probabilities.

4. A mechanism is also adopted to stop pushes when a device is handed over to another cell.

The application-distribution system of the service provider uses these algorithms and adopts bandwidths as per the number of subscribers for the published data records. On the device handoff, the subscription cancels or may be passed on to new service provider system.

Advantages of Push based mechanisms:

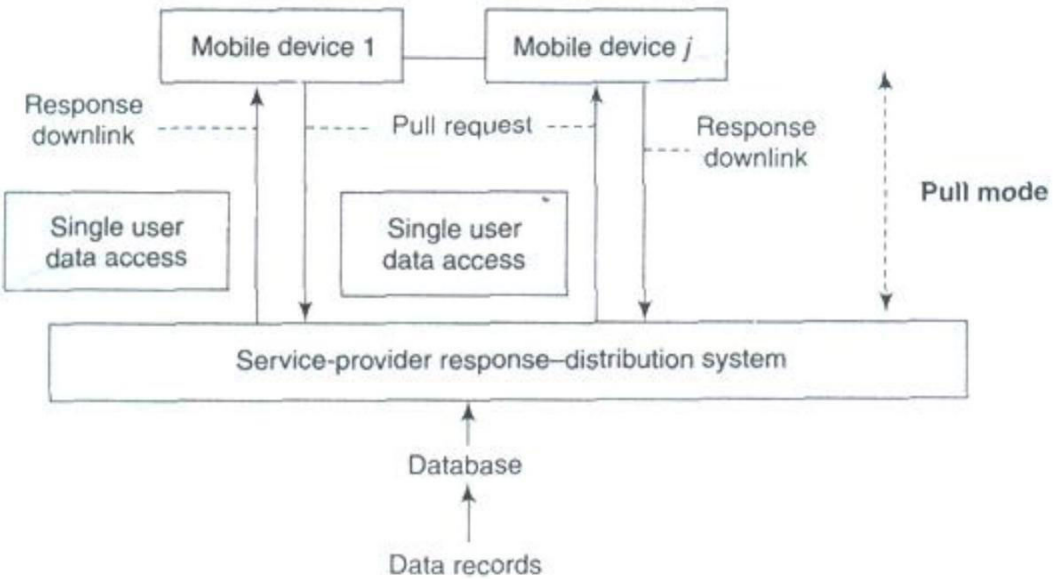
- Push-based mechanisms enable broadcast of data services to multiple devices.
- The server is not interrupted frequently by requests from mobile devices.
- These mechanisms also prevent server overload, which might be caused by flooding of device requests
- Also, the user even gets the data he would have otherwise ignored such as traffic congestion, forthcoming weather reports etc

Disadvantages:

- Push-based mechanisms disseminate of unsolicited, irrelevant, or out-of-context data, which may cause inconvenience to the user.

Pull based Mechanisms

The user-device or computing system pulls the data records from the service provider's application database server or from a set of distributed computing systems. Examples are music album server, ring tones server, video clips server, or bank account activity server. Records are pulled by the mobile devices on demand followed by the selective response from the server. Selective response means that server transmits data packets as response selectively, for example, after client-authentication, verification, or subscription account check. The pull mode is also known as the on-demand mode. The following figure shows a pull-based data-delivery mechanism in which a device pulls (demands) from a server or computing system, the data records generated by a set of distributed computing systems.



Pull based Delivery Mechanism

Pull-based mechanisms function in the following manner:

1. The bandwidth used for the uplink channel depends upon the number of pull requests.
2. A pull threshold is selected. This threshold limits the number of pull requests in a given period of time. This controls the number of server interruptions.
3. A mechanism is adopted to prevent the device from pulling from a cell, which has handed over the concerned device to another cell. On device handoff, the subscription is cancelled or passed on to the new service provider cell

In pull-based mechanisms the user-device receives data records sent by server on demand only.

Advantages of Pull based mechanisms:

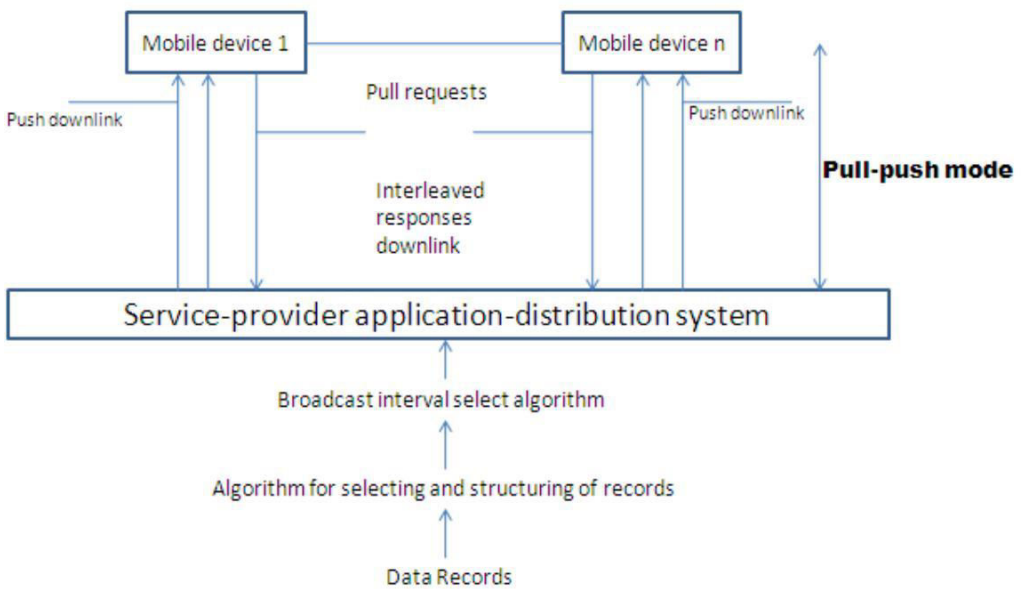
- With pull-based mechanisms, no unsolicited or irrelevant data arrives at the device and the relevant data is disseminated only when the user asks for it.
- Pull-based mechanisms are the best option when the server has very little contention and is able to respond to many device requests within expected time intervals.

Disadvantages:

- The server faces frequent interruptions and queues of requests at the server may cause congestion in cases of sudden rise in demand for certain data record.
- In on-demand mode, another disadvantage is the energy and bandwidth required for sending the requests for hot items and temporal records

Hybrid Mechanisms

A hybrid data-delivery mechanism integrates pushes and pulls. The hybrid mechanism is also known as interleaved-push-and-pull (IPP) mechanism. The devices use the back channel to send pull requests for records, which are not regularly pushed by the front channel. The front channel uses algorithms modeled as broadcast disks and sends the generated interleaved responses to the pull requests. The user device or computing system pulls as well receives the pushes of the data records from the service provider's application server or database server or from a set of distributed computing systems. Best example would be a system for advertising and selling music albums. The advertisements are pushed and the mobile devices pull for buying the album.



Hybrid interleaved push-pull-based data-delivery mechanism

The above figure shows a hybrid interleaved, push-pull-based data-delivery mechanism in which a device pulls (demands) from a server and the server interleaves the responses along with the pushes of the data records generated by a set of distributed computing systems. Hybrid mechanisms function in the following manner:

1. There are two channels, one for pushes by front channel and the other for pulls by back channel.
2. Bandwidth is shared and adapted between the two channels depending upon the number of active devices receiving data from the server and the number of devices requesting data pulls from the server.
3. An algorithm can adaptively chop the slowest level of the scheduled pushes successively The data records at lower level where the records are assigned lower priorities can have long push intervals in a broadcasting model.

Advantages of Hybrid mechanisms:

- The number of server interruptions and queued requests are significantly reduced.

Disadvantages:

- IPP does not eliminate the typical server problems of too many interruptions and queued requests.
- Another disadvantage is that adaptive chopping of the slowest level of scheduled pushes.

Selective Tuning and Indexing Techniques

The purpose of pushing and adapting to a broadcast model is to push records of greater interest with greater frequency in order to reduce access time or average access latency. A mobile device does not have sufficient energy to continuously cache the broadcast records and hoard them in its memory. A device has to dissipate more power if it gets each pushed item and caches it. Therefore, it should be activated for listening and caching only when it is going to receive the selected data records or buckets of interest. During remaining time intervals, that is, when the broadcast data buckets or records are not of its interest, it switches to idle or power down mode.

Selective tuning is a process by which client device selects only the required pushed buckets or records, tunes to them, and caches them. Tuning means getting ready for caching at those instants and intervals when a selected record of interest broadcasts. Broadcast data has a structure and overhead. Data broadcast from server, which is organized into buckets, is interleaved. The server prefixes a directory, hash parameter (from which the device finds the key), or index to the buckets. These prefixes form the basis of different methods of selective tuning. Access time ($\mathbf{t_{access}}$) is the time interval between pull request from device and reception of response from broadcasting or data pushing or responding server. Two important factors affect t_{access} – (i) number and size of the records to be broadcast and (ii) directory- or cache-miss factor (if there is a miss then the response from the server can be received only in subsequent broadcast cycle or subsequent repeat broadcast in the cycle).

Directory Method

One of the methods for selective tuning involves broadcasting a directory as overhead at the beginning of each broadcast cycle. If the interval between the start of the broadcast cycles is T, then directory is broadcast at each successive intervals of T. A directory can be provided which

specifies when a specific record or data item appears in data being broadcasted. For example, a directory (at header of the cycle) consists of directory start sign, 10, 20, 52, directory end sign. It means that after the directory end sign, the 10th, 20th and 52nd buckets contain the data items in response to the device request. The device selectively tunes to these buckets from the broadcast data.

A device has to wait for directory consisting of start sign, pointers for locating buckets or records, and end sign. Then it has to wait for the required bucket or record before it can get tuned to it and, start caching it. Tuning time t_{tune} is the time taken by the device for selection of records. This includes the time lapse before the device starts receiving data from the server. In other words, it is the sum of three periods—time spent in listening to the directory signs and pointers for the record in order to select a bucket or record required by the device, waiting for the buckets of interest while actively listening (getting the incoming record wirelessly), and caching the broadcast data record or bucket.

The device selectively tunes to the broadcast data to download the records of interest. When a directory is broadcast along with the data records, it minimizes t_{tune} and t_{access}. The device saves energy by remaining active just for the periods of caching the directory and the data buckets. For rest of the period (between directory end sign and start of the required bucket), it remains idle or performs application tasks. Without the use of directory for tuning, $t_{tune} = t_{access}$ and the device is not idle during any time interval.

Hash-Based Method

Hash is a result of operations on a pair of key and record. Advantage of broadcasting a hash is that it contains a fewer bits compared to key and record separately. The operations are done by a hashing function. From the server end the hash is broadcasted and from the device end a key is extracted by computations from the data in the record by operating the data with a function called hash function (algorithm). This key is called hash key.

Hash-based method entails that the hash for the hashing parameter (hash key) is broadcasted. Each device receives it and tunes to the record as per the extracted key. In this method, the records that are of interest to a device or those required by it are cached from the broadcast cycle by first extracting and identifying the hash key which provides the location of the record. This helps in tuning of the device. Hash-based method can be described as follows:

1. A separate directory is not broadcast as overhead with each broadcast cycle.
2. Each broadcast cycle has hash bits for the hash function H, a shift function S, and the data that it holds. The function S specifies the location of the

record or remaining part of the record relative to the location of hash and, thus, the time interval for wait before the record can be tuned and cached.

3. Assume that a broadcast cycle pushes the hashing parameters $\mathbf{H}(R_i)$ [H and S] and record R_i. The functions H and S help in tuning to the H(R_i) and hence to R_i as follows—H gives a key which in turn gives the location of H(R_i) in the broadcast data. In case H generates a key that does not provide the location of H(R_i) by itself, then the device computes the location from S after the location of H(R_i). That location has the sequential records R_i and the devices tunes to the records from these locations.
4. In case the device misses the record in first cycle, it tunes and caches that in next or some other cycle.

Index-Based Method

Indexing is another method for selective tuning. Indexes temporarily map the location of the buckets. At each location, besides the bits for the bucket in record of interest data, an offset value may also be specified there. While an index maps to the absolute location from the beginning of a broadcast cycle, an offset index is a number which maps to the relative location after the end of present bucket of interest. Offset means a value to be used by the device along with the present location and calculate the wait period for tuning to the next bucket. All buckets have an offset to the beginning of the next indexed bucket or item.

Indexing is a technique in which each data bucket, record, or record block of interest is assigned an index at the previous data bucket, record, or record block of interest to enable the device to tune and cache the bucket after the wait as per the offset value. The server transmits this index at the beginning of a broadcast cycle as well as with each bucket corresponding to data of interest to the device. A disadvantage of using index is that it extends the broadcast cycle and hence increases t_{access}.

The index I has several offsets and the bucket type and flag information. A typical index may consist of the following:

1. $I_{offset}(1)$ which defines the offset to first bucket of nearest index.
2. Additional information about $T_{b,}$ which is the time required for caching the bucket bits in full after the device tunes to and starts caching the bucket. This enables transmission of buckets of variable lengths.
3. I_{offset} (next) which is the index offset of next bucket record of interest.

4. $I_{offset}(end)$ which is the index offset for the end of broadcast cycle and the start of next cycle. This enables the device to look for next index I after the time interval as per $I_{offset}(end)$. This also permits a broadcast cycle to consist of variable number of buckets.
5. I_{type}, which provides the specification of the type of contents of next bucket to be tuned, that is, whether it has an index value or data.
6. A flag called dirty flag which contains the information whether the indexed buckets defined by $I_{offset}(1)$ and $I_{offset}(next)$ are dirty or not. An indexed bucket being dirty means that it has been rewritten at the server with new values. Therefore, the device should invalidate the previous caches of these buckets and update them by tuning to and caching them.

The advantage of having an index is that a device just reads it and selectively tunes to the data buckets or records of interest instead of reading all the data records and then discarding those which are not required by it. During the time intervals in which data which is not of interest is being broadcast, the device remains in idle or power down mode.

Transmission of an index I only once with every broadcast cycle increases access latency of a record as follows: This is so because if an index is lost during a push due to transmission loss, then the device must wait for the next push of the same index-record pair. The data tuning time now increases by an interval equal to the time required for one broadcast cycle. An index assignment strategy ***(I, m)*** is now described. *(I, m)* indexing means an index I is transmitted *m* times during each push of a record. An algorithm is used to adapt a value of **m** such that it minimizes access (caching) latency in a given wireless environment which may involve frequent or less frequent loss of index or data. Index format is adapted to (I, *m)* with a suitable value of m chosen as per the wireless environment. This decreases the probability of missing I and hence the caching of the record of interest

Indexing reduces the time taken for tuning by the client devices and thus conserves their power resources. Indexing increases access latency because the number of items pushed is more (equals m times index plus *n* records).

Distributed Index Based Method

Distributed index-based method is an improvement on the (I, *m)* method. In this method, there is no need to repeat the complete index again and again. Instead of replicating the whole index *m* times, each index segment in a bucket describes only the offset I' of data items which immediately follow. Each index I is partitioned into two parts—I' and I". I" consists of

unrepeated k levels (sub-indexes), which do not repeat and I' consists of top I repeated levels (sub-indexes).

Assume that a device misses I(includes I' and I' once) transmitted at the beginning of the broadcast cycle. As I' is repeated *m* - I times after this, it tunes to the pushes by using I', The access latency is reduced as I' has lesser levels.

Flexible Indexing Method

Assume that a broadcast cycle has number of data segments with each of the segments having a variable set of records. For example, let *n* records, R_o to R_{n-1}, be present in four data segments, $R_{()}$ to R_{i-1}, R_i to R_{j-1} , R_j to R_{j-1} and R_k to R_{n-1}. Some possible index parameters are (i) I_{seg}, having just 2 bits for the offset, to specify the location of a segment in a broadcast cycle, (ii) I_{rec}, having just 6 bits for the offset, to specify the location of a record of interest within a segment of the broadcast cycle, (iii) I_b, having just 4 bits for the offset, to specify the location of a bucket of interest within a record present in one of the segments of the broadcast cycle. Flexible indexing method provides dual use of the parameters (e.g., use of I_{seg} or I_{rec} in an index segment to tune to the record or buckets of interest) or multi-parameter indexing (e.g., use of I_{seg}, I_{rec}, or I_b in an index segment to tune to the bucket of interest).

Assume that broadcast cycle has *m* sets of records (called segments). A set of binary bits defines the index parameter I_{seg},. A local index is then assigned to the specific record (or bucket). Only local index (I_{rec} or I_b) is used in (I_{loc}, m) based data tuning which corresponds to the case of flexible indexing method being discussed. The number of bits in a local index is much smaller than that required when each record is assigned an index. Therefore, the flexible indexing method proves to be beneficial.

Alternative Methods

Temporal Addressing Temporal addressing is a technique used for pushing in which instead of repeating I several times, a temporal value is repeated before a data record is transmitted. When temporal information contained in this value is used instead of address, there can be effective synchronization of tuning and caching of the record of interest in case of non-uniform time intervals between the successive bits. The device remains idle and starts tuning by synchronizing as per the temporal (time)-information for the pushed record. Temporal information gives the time at which cache is scheduled. Assume that temporal address is 25675 and each address corresponds to wait of 1 ms, the device waits and starts synchronizing the record after 25675 ms.

Broadcast Addressing: Broadcast addressing uses a broadcast address similar to IP or multicast address. Each device or group of devices can be assigned an address. The devices cache the records which have this address as the broadcasting address in a broadcast cycle. This address can be used along with the pushed record. A device uses broadcast address in place of the index I to select the data records or sets. Only the addressed device(s) caches the pushed record and other devices do not select and tune to the record. In place of repeating I several times, the broadcast address can be repeated before a data record is transmitted. The advantage of using this type of addressing is that the server addresses to specific device or specific group of devices.

Use of Headers: A server can broadcast a data in multiple versions or ways. An index or address only specifies where the data is located for the purpose of tuning. It does not specify the details of data at the buckets. An alternative is to place a header or a header with an extension with a data object before broadcasting. Header is used along with the pushed record. The device uses header part in place of the index / and in case device finds from the header that the record is of interest, it selects the object and caches it. The header can be useful, for example it can give information about the type, version, and content modification data or application for which it is targeted.

Notes for Indexing Techniques (Prepared by Kancherla Yasesvi, 08071A0522)

(1, m) Index

The (1, m) indexing scheme is an index allocation method where a complete index is broadcast m times during a broadcast. All buckets have an offset to the beginning of the next index segment. The first bucket of each index segment has a tuple containing two fields. The first field contains the key value of the object that was broadcast last and the second field is an offset pointing to the beginning of the next broadcast. This tuple guides clients who missed the required object in the current broadcast so that they can tune to the next broadcast.

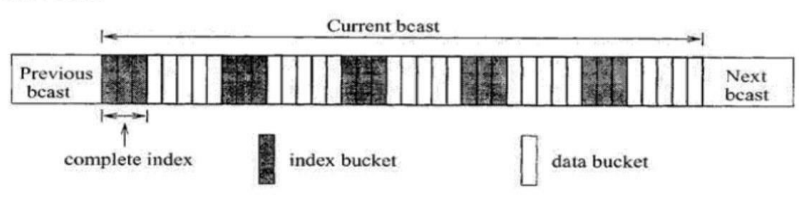


Figure 4.3. Bcast organization in the (1 ,*m*) indexing method.

The client's access protocol for retrieving objects with key value k is as follows:

1. Tune into the current bucket on the broadcast channel. Get the offset to the next index segment.
2. Go to the doze mode and tune in at the broadcast of the next index segment.
3. Examine the tuple in the first bucket of the index segment. If the target object has been missed, obtain the offset to the beginning of the next bcast and goto 2; otherwise goto 4.
4. Traverse the index and determine the offset to the target data bucket. This may be accomplished by successive probes, by following the pointers in the multi-level index. The client may doze off between two probes.
5. Tune in when the desired bucket is broadcast, and download it(and subsequent ones as long as their key is k).

Advantage:

1. This scheme has good tuning time.

Disadvantage:

1. The index is entirely replicated m times; this increases the length of the broadcast cycle and hence the average access time.

The optimal m value that gives minimal average access time is (data file size/index size)$^{1/2}$.

There is actually no need to replicate the complete index between successive data blocks. It is sufficient to make available only the portion of index related to the data buckets which follow it. This is the approach adopted in all the subsequent indexing schemes.

Tree-based Index/Distributed indexing scheme

In this scheme a data file is associated with a B^+-tree index structure. Since the broadcast medium is a sequential medium, the data file and index must be flattened so that the data are index are broadcast following a preorder traversal of the tree. The index comprises two portions: the first k levels of the index will be partially replicated in the broadcast, and the remaining levels will not be replicated. The index nodes at the $(k+1)^{th}$ level are called the non-replicated roots.

Essentially, each index subtree whose root is a non-replicated root will appear once in the whole bcast just in front of the set of data segments it indexes. On the other hand, the nodes at the replicated levels are replicated at the beginning of the first broadcast of each of its children nodes.

To facilitate selective tuning, each node contains meta-data that help in the traversal of the trees. All non-replicated buckets contain pointers that will direct the search to the next copy of its replicated ancestors. On the other hand, all replicated index buckets contain two tuples that can direct the search to continue in the appropriate segments. The first tuple is a pair(x, ptr_{begin}) that indicates that key values less than x have been missed and so search must continue from the beginning of the next bcast(which is ptr_{begin} buckets away). The second pair (y, ptr) indicates that key values greater than or equal to y can be found ptr offset away. Clearly, if the desired object has key value between x and y, the search can continue as in conventional search operation.

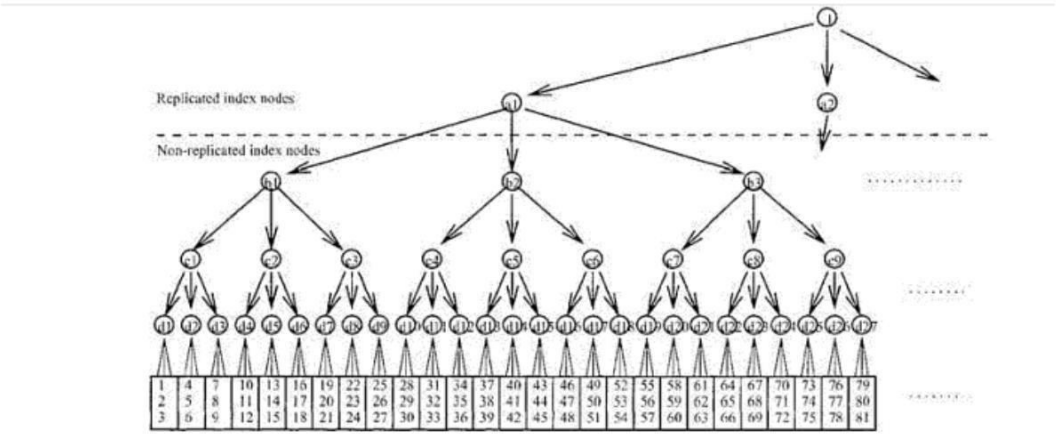


Figure 4.4. A partial data file and its index tree.

The client's access protocol for retrieving objects with key value k is as follows:

1. Tune to the current bucket of the bcast. Get the offset to the next index bucket, and doze off.
2. Tune to the beginning of the designated bucket and examine the meta-data.
   - If the desired object has been missed, doze off till the beginning of the next bcast. Goto 2.
   - If the desired object is not within the data segment covered by the index bucket, doze off to the next higher level index bucket. Goto 3.
   - If the desired object is within the data segment covered by the index bucket, goto 3.

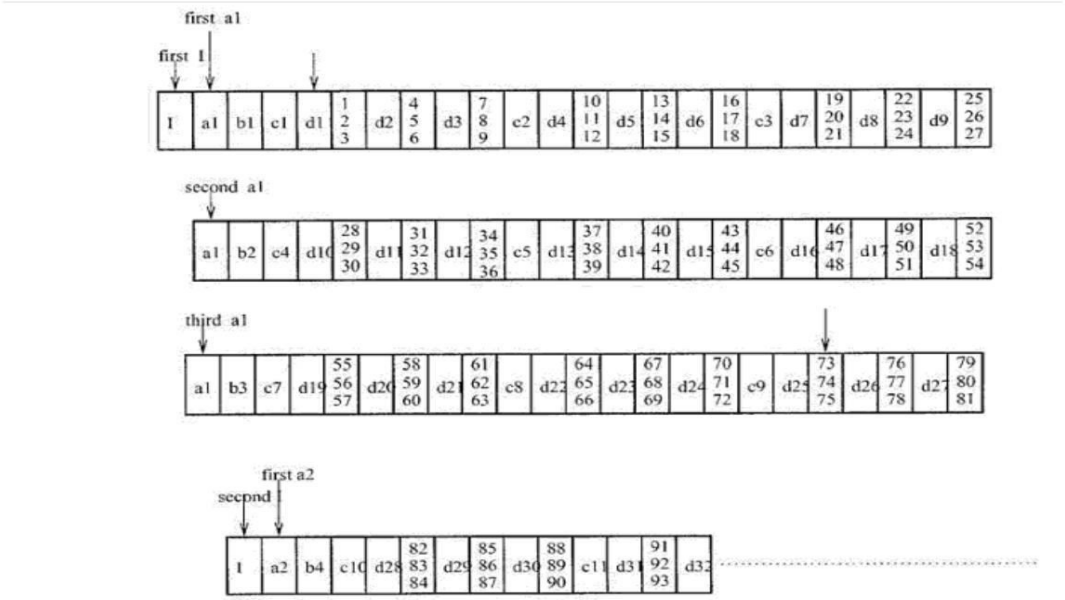


Figure 4.5. The data broadcast for the distributed indexing scheme with partial path replication.

3. Probe the designated index bucket and follow a sequence of pointers to determine when the data bucket containing the target object will be broadcast. The client may doze off in between two probes.
4. Tune in again when the bucket containing objects with key k is broadcast, and download the bucket (and all subsequent buckets as long as they contain objects with key k).

Advantage:

1. Compared to (1, m) index scheme this scheme has lower access time and its tuning time is also comparable to that of (1, m) index scheme.

Flexible Indexing Scheme

This scheme splits a sorted list of objects into equal-sized segments, and provides indexes to navigate through the segments. At the beginning of each segment, there is a control index which comprises of two components: a global index and a local index. The global index is used to determine the segment which object may be found, while the local index provides the offset to the portion within the segment where the object may be found.

Suppose the file is organized into p segments. Then the global index at a segment, says, has $[\log_2 i]$ (key, ptr) pairs, where i the number of segments in front of and including segment s, key is an object key, and ptr is an offset. For the first entry, key is the key value of the first data item in segment s and ptr is the offset to the beginning of the next version. Bold examining this pair, the client will know if it has missed the data and if so wait till the next bcast. For the j^{th} entry (j>1), key is the key value of the first data item in the $([\log_2 i/2^{j-1}] +1)^{th}$ segment following segment s and ptr is the offset to the first data bucket of that segment.

The local index consists of m(key, ptr) pairs that essentially partition each segment further into m+1 sections. For the first entry, key is the key value of the first data item of section m+1 and ptr is the offset to that section. For the jth pair, key is the key value of the first data item of section (m+1-j) and ptr is the offset to the first bucket of that section.

Hence, it is clear that the number of segments and the number of sections per segment can affect the performance of the scheme. Increasing the number of segments or sections will increase the length of the broadcast cycle and reduce the tuning time, and vice versa. Thus, the scheme is flexible in the sense it can be tuned to fit an application's needs.

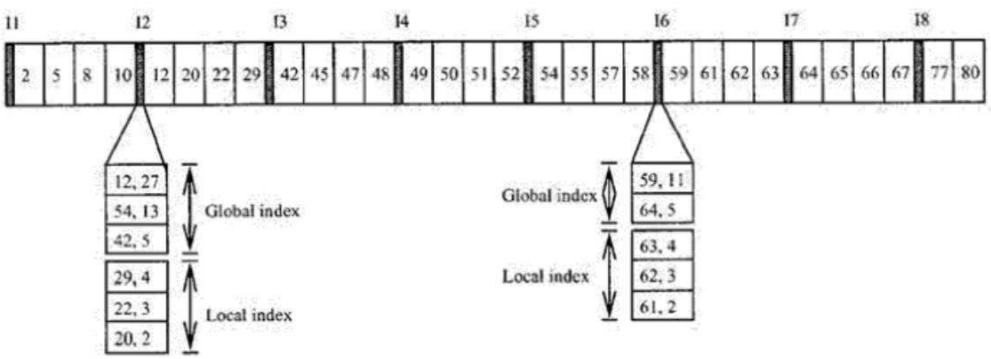


Figure 4.8. Flexible index scheme.

The client's access protocol for retrieving objects with key value k is as follows:

1. Tune into the channel for a bucket, obtain the offset to the next index segment. Doze off until the next index segment is broadcast.
2. Examine the global index entries. If the target object belongs to another segment, get the offset; doze off for appropriate amount of time and goto 2.
3. Examine the local index entries. Obtain the offset to the section where the target data is stored. Switch to doze mode for appropriate amount of time.
4. Examine objects in the data bucket for the desired object, and download the object.

Introduction to SYNCHRONIZATION

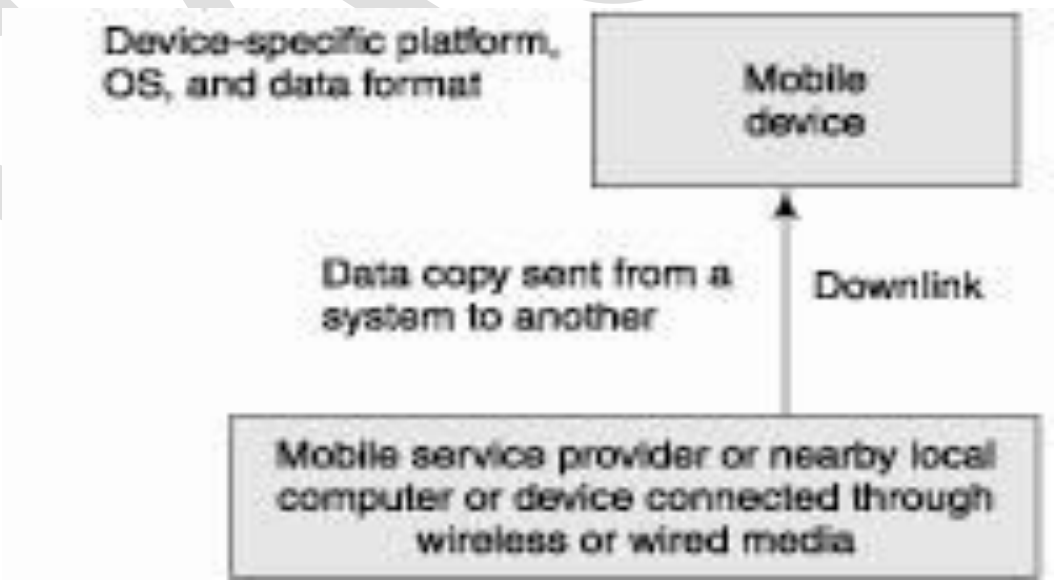
Syllabus: Introduction to Synchronization, What is Synchronization in Mobile Computing Systems, Usage models for synchronization n mobile application, Domain-dependent specific Rules for data synchronization, Personal information manager, Synchronization & conflict resolution strategies, About Synchronizer Mobile Agent, Mobile Agent Design, Application server.

Data Replication and Synchronization in Mobile Computing Systems

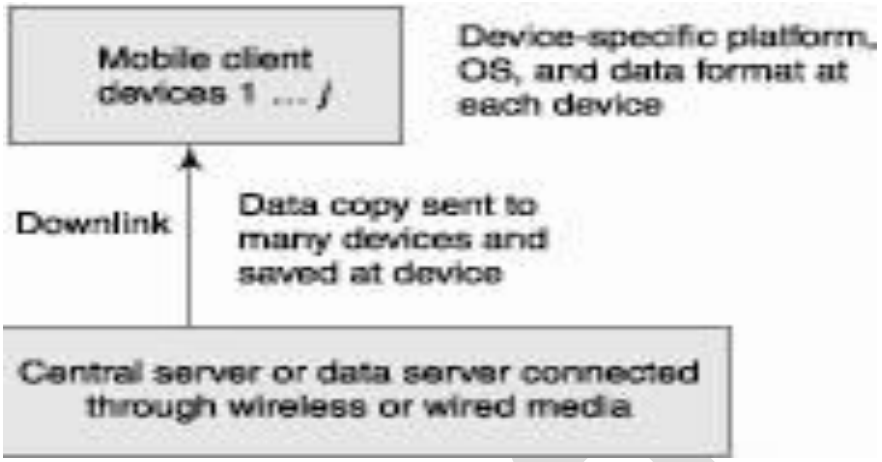
Data dissemination and replication

- Data replication at either remote or local location (s) may entail copying of data at one place after copying from another (i.e., recopying), copying from one to many others or from many to many others
- For example, videos of faculty lectures or music files get replicated at a mobile phone

Data replication from data source and device



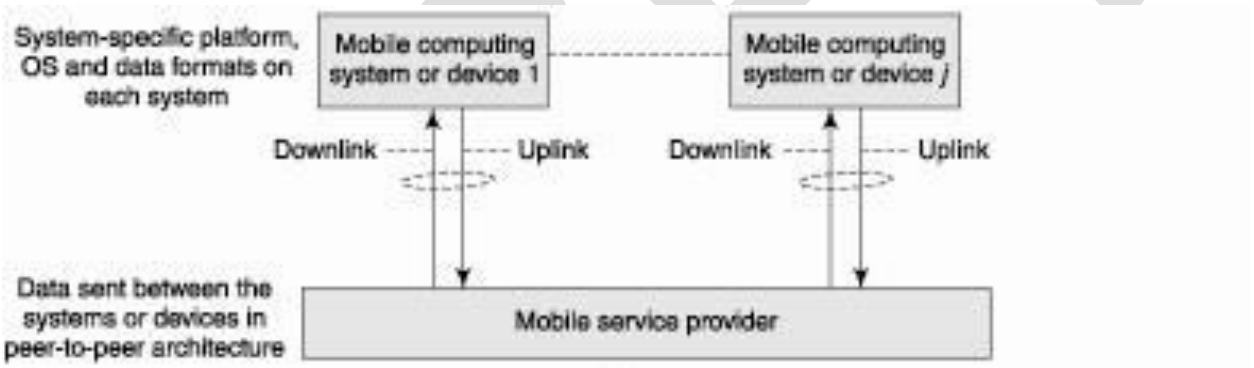
Data replication from data source server to many clients (devices)



One to many synchronization

- Each system or device caches the data pushed from the server or sends a pull request to the central server and gets a response

Data replication among systems and devices in peer to peer architecture



Many-to-many synchronization

- Employs peer-to-peer architecture where each system is capable of sending pull requests and of pushing responses

Full copy from a source

- Means that the full set of data records replicates according to certain domain-specific data format rules at the replicating devices or systems
- A server having a set of 8 images with resolution 640 × 640 pixels
- In the domain of a mobile device, it can replicate and hoard with 160 × 160 pixels
- When all 8 images copied, though with the different resolution, then it is known as full copy replication

Full copy from a source Example

- A server having a set of 8 images with resolution 640 × 640 pixels
- In the domain of a mobile device, it can replicate and hoard with 160 × 160 pixels

- Full copy replication─ when all 8 images copied, though with the different resolution

Partial copying of data from the source

- A subset of the data set copied according to certain domain-specific rules at the devices or systems
- Assume that a server has a hourly data set of 24 temperature records with ± 0.1°C
- Partial copy replication In mobile device domain, assume that it replicates and hoards three hourly records with ± 1°C

Data Synchronization

- Data replication precedes data synchronization
- The synchronization refers to maintaining data consistency among the disseminated or distributed data
- Data consistency─ if there is data modification at the server then that should reflect in the data with the device within a defined period

Data Synchronization in Mobile computing systems

- Defined as the process of maintaining the availability of data generated from the source and maintaining consistency between the copies pushed from the data source and local cached or hoarded data at different computing systems without discrepancies or conflicts among the distributed data.

Consistent copy of data

- A copy which may not be identical to the present data record at the data-generating source, but must satisfy all the required functions and domain-dependent specific rules

Domain Specific rules for consistency

- In terms of resolution, precision, data format, and time interval permitted for replication
- A consistent copy should not be in conflict with the data at the data-generating source

Data synchronization for accessing data from server

- Helps mobile users in accessing data and using it for computing on mobile devices
- When a device not connected to a source or server, the user may employ data that is not in conflict with the present state of data at the source

Data synchronization for caching data in and from personal area computer

- Helps mobile users in hoarding the device data at the personal area computer
- Also helps mobile users in hoarding the personal area computer data

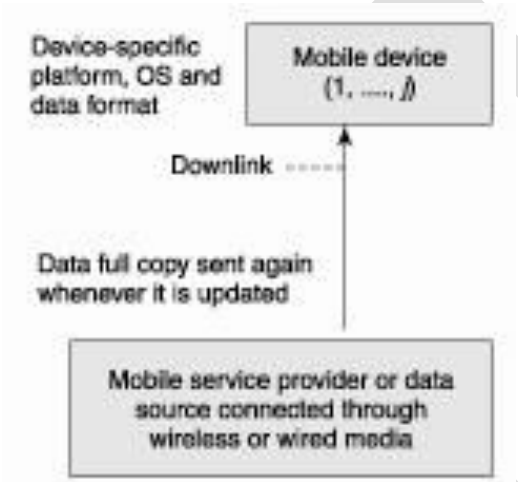
Data synchronization enhancing device mobility

- When initiated at frequent intervals enhances device mobility
- Ensures that device applications use the latest updated data from the source, even when the device is disconnected

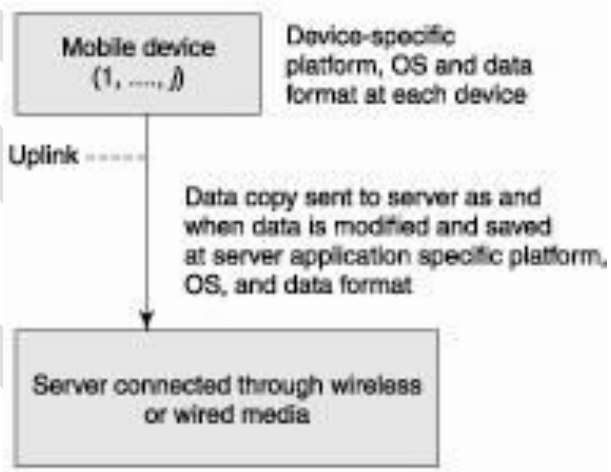
Data synchronization with enterprise server

- Helps storage at enterprise server a large chunks of information for the many devices connected to it and update partial copies of data at frequent intervals

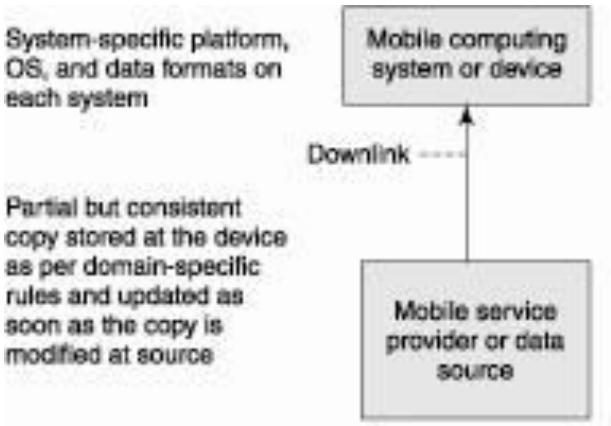
Full copy Synchronization at the device when the server sends data



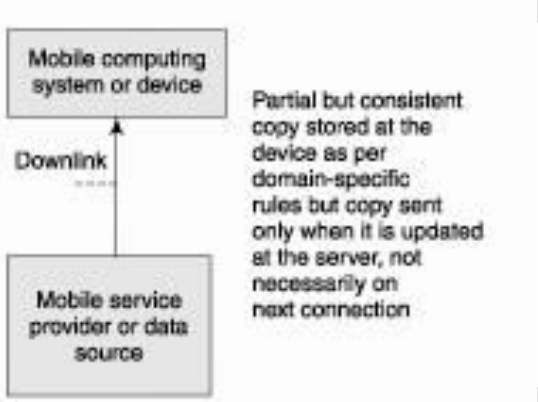
Full copy Synchronization at the server when the device sends data



Partial copy Synchronization of consistent copy without the delays



Partial copy Synchronization of consistent copy but after delays



Data Synchronization Types, Formats and Usage Models in Mobile Computing Systems

Data synchronization Needs

- Required between the mobile device and service provider
- Between the device and personal area computer
- With nearby wireless access point (in WiFi connection)
- Another nearby device

Two-way synchronization of partial or full copies of data

- **Between mobile-device and personal-area computer**
- **For example, whenever the list of contacts and personal information manager data is modified at any of them, it is made consistent after synchronization**

2. Server-alerted synchronization

- **`The server alerts the client the data modification or additions**
- **The client synchronizes the modified or new data by pull request**
- **For example, alerting new e-mail and the device pulls that**

3. One-way server-initiated synchronization

- **Server initiates synchronization of any new modification since communication of last modification**
- **Sends modified data copies to the client**
- **When a new email arrives at a server, it initiates the synchronization as and when the device connects to the server and pushes the mail**

4. Client initiated refresh synchronization

- The client initiates synchronization with the server for refreshing its existing data copies
- For refreshing the configuration parameters saved at the server for it
- For example, a computer or mobile device initiates refreshing of the hoarded contacts and personal information data either at periodic intervals or as and when it connects
- if the device configuration changes or a new device connects to a server, then the configuration parameters sent earlier refresh at the server

5. Client-initiated synchronization

- With the server for sending its modifications, for example, device configuration for the services
- For example, a client mobile device initiates synchronization of the mails or new ring tones or music files available at the server either at periodic intervals or as and when it connects to it

6. Refresh from client for backup and update synchronization

- The client initiates synchronization
- Sends backup to the server for updating its data
- For example, a computer or mobile device initiates refreshing of the hoarded contacts and personal information data either at periodic intervals or as and when it connects to the server

7. Slow (full data copy and thorough) synchronization

- Client and server data compared for each data field and are synchronized as per conflict resolution rules
- Full copy synchronization usually takes place in idle state of the device
- Not immediately on connecting to the server, that's why called slow

Formats of Synchronized Data Copies

- Can be different from each other at client and server
- When the data at a source synchronizes with the data at other end, it does so as per the format specified at that end

Formats of Database records

- The records indexed enabling search by querying using the indexes, for example, the relational database records
- The database record retrieved by sending a query specifying the entries in these indexes
- Format DB2 at server and DB2e Every place at the mobile device

Flat file Synchronization

- Data can be interpreted only if the file is read from beginning to end and that data cannot be picked from anywhere within the file
- For example, an XML or html file at the server synchronizes with the file at the device which is in text format or is a binary file depending upon the information format
- Information format in mobile computing XML document format
- For transmission it is WBXML (WAP Binary XML) content format
- Address book data at a mobile device with the data transmitted in WBXML format

Device-specific storage Format

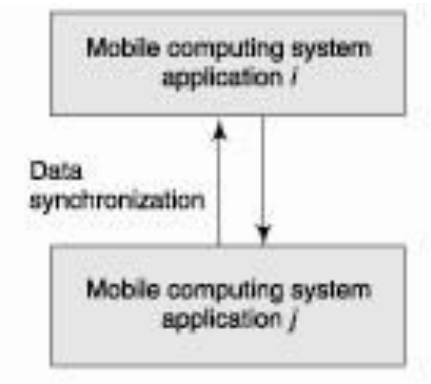
- AAC (Apple Audio Communication) files used for audio communication with an Apple iPhone
- A file in AAC format synchronizes with music files in some other format at a computer or remote website serving the music files
- At a mobile device the *Contacts* information in vCard format
- Calendar, tasks-to-do list, and journal information are in vCalendar, vToDo, and vJournal formats, respectively

Usage Models for Synchronization in Mobile Applications

- Four usage models employed for synchronization in mobile computing systems

1. **Synchronization between two APIs within a mobile computing system**

Synchronization between two APIs



Synchronization between two APIs

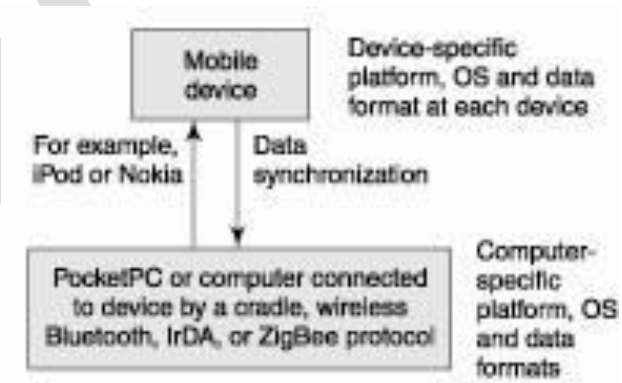
The data generated by an application synchronized and used in another application

- An API running at the device synchronizes data with another application on the same or another device or computer

Example of synchronization between APIs

- Data records at personal information manager (PIM) API synchronized with the email API
- When email from a new source retrieves at the email API in the device, the name and email address data fields at the application saved as new data record at PIM API
- When an email is to be sent to the same person, the email API uses the same data record from the PIM API

2. **Synchronization between the device and nearby device**

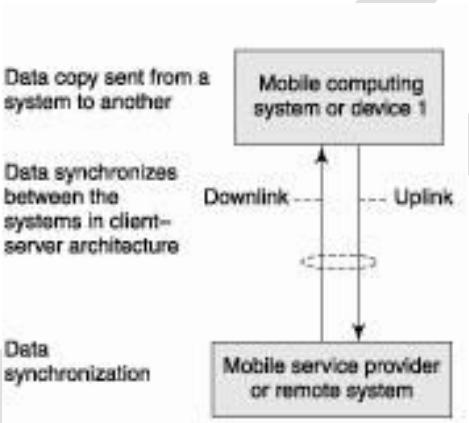


- Device and computer synchronize their data
- Also called personal area synchronization (PAS)
- Using PAS software, for example, HotSync or ActiveSync

Examples of PAS

- Synchronization with nearby PC through a serial port using a cradle and wired connection to PC through the cradle
- Synchronization with the nearby computer through a wireless personal area network (WPAN) using ZigBee or Bluetooth

3. Synchronization between remote systems and device



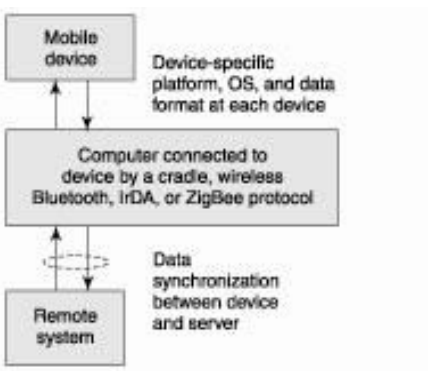
Synchronization between remote systems and device

- The device data records synchronize with the mobile service provider server records
- The remote server or systems synchronize their data with the mobile device
- The device connects to remote systems on Internet through the wired, wireless mobile service provider, or WiFi network

Example of Synchronization between remote systems and device

- Wireless email synchronization using Intellisync between the device and remote server using SyncML language

4. Synchronization through a local pass-through system



5.

Using local pass through computer or system

- Device data records synchronize with the records of remote system, for example, an enterprise server, through a local computer system

Example of Using local pass through computer or system

- The device first synchronizes through ActiveSync or HotSync or Intellisync or Bluetooth to local computer connected by personal area synchronizer
- Then the computer synchronizes to Internet through WLAN, WiFi, or wired network

Domain dependent Specific Rules and Conflict resolution Strategies

1. **Data synchronization in domain-specific platforms and data formats**

Data synchronization between data-generating domain and destined domain, both having different platform and data formats

Examples of synchronization in domain-specific platforms and data formats

- **A copy of database record at the device structured text or XML format and the device OS platform Symbian**
- **The record synchronized with the database record at the server where it is in DB2 or Oracle database format and the OS is Windows**

2. **Domain-specific data-property-dependent synchronization**

- **Data synchronization between one domain with one property of data and another domain having different property**

Examples of Domain-specific data-property-dependent synchronization

- **A data record at a device having an ID specified by a byte synchronizes with the record, which has an ID specified by 16-bit word at the server**
- **A device using 8-bit ASCII characters for an ID while the server using 16-bit Unicode characters**

3. **Synchronization up to the last successful act of synchronization**

- **A domain-specific rule that data record considered to be synchronized if it was updated at the last connection**

Example of Synchronization up to the last successful act

- A phonebook records of missed calls, dialled numbers, and received calls
- Data record at the device synchronized with the record in the phonebook
- If it updated at the last connection, then it eventually updates again on the next connection

4. Memory-infrastructure-dependent based synchronization at the domains

- A domain-specific rule that data records synchronized up to the allotted memory

Example of Memory-infrastructure-dependent based synchronization

- A remote server maintaining full address book with allotted memory of 8 MB and a device allocated 128 kB for the address book
- Only a part of e-mail database, only 100 new email addresses synchronizes and saves in the device PIM (personal information manager)

5. Synchronization with temporal properties of data

Domain-specific rule that data records synchronized with data generated at source within specific time interval and at time specified at the domain

Example of Synchronization with temporal properties of data

- The flight time table data set of device synchronized every week and weather report once every day
- At the device weather report updated and synchronized up to the last day
- Eventually updates on a day if available at the server

Synchronization with temporal properties of data

- May be periods of inconsistency when temporal properties of data being used for synchronization
- However, mobile applications remain unaffected if there are no temporal conflicts and unaccountable discrepancies

Conflict in synchronization

- Arises when a data copy changed at one end but not simultaneously modified at other ends
- Therefore, the same data item at two ends, P and Q, in conflict during computation in the time interval between $t1$ and $t2$, where $t1$ and $t2$— the instants when P and Q get the modified data copy

Synchronization and Conflict Resolution Strategies

- A conflict resolution strategy adopted in such cases to resolve conflicts

- The strategy specifies the rules that need to be applied for conflict resolution

1. Priority-based resolution rule

Data-server can be specified as dominant higher priority entity for conflict resolution of synchronized data records

Example of Priority-based resolution rule

- Mobile-service-provider server S having a list of missed, dialled, and received calls for the device D
- D has a synchronised list of missed, dialled, and received calls
- When the list at D in conflict with the list at S, priority-based resolution rule specifies that the server priority is higher

2. Time-based resolution rule

- Data node P specified as dominant entity when P always receives copies first from the server S

Time-based resolution rule ─ Example

- S having the emails disseminated to the device D at an instant $t1$
- D connects to a personal area computer (PC) to which the device always synchronizes the mails at a later instant $t2$
- Time-based resolution rule─ D dominant because it receives the mails earlier than the PC

3. Information-based resolution rule

a. Data node can be specified as dominant entity when information specific to it is synchronized with other nodes

Example of Information-based resolution rule

- Server S having the device configuration record disseminated from the device D
- Information-based resolution rule specifies that since the information is for the device D hence D is dominant node
- For device-specific information, the device data accepted rather than the server data

4. **Time-stamp-based resolution rule Device-specific storage Format**

- Time-stamp-based resolution rule necessitates that a time-stamp must be used while sending a data copy The copy found to be latest resolves the conflict

- **Example of Time stamping rule for conflict resolution**

- Server S having the flight information which it always disseminates at regular intervals with a time stamp over it to the device D and as well as to a PC
- Time-stamp-based resolution rule specifies that the node with flight information with latest time stamp dominant

5. User-interaction-based resolution rule

- **An API at a device allows a user to interact with the device**
- **Interaction resolves the conflict arising out of the duplicate or multiple entries**
- **The duplicate data entries permitted at the node when a receiver API later on resolves the conflict after interaction with user**

Example of User-interaction-based resolution rule

- Two phone number entries found for same name and address, the device prompts user to resolve the conflict
- User resolves the conflict by opting for one of it