# An Evaluation of Cache Invalidation Strategies in Wireless Environments

Authors: Kian-Lee Tan, Jun Cai and Beng Chin Ooi

Presenters: Feng Wang, Anh Phan, Mark Yue Li

## Two obstacles for mobile computing

- The limited bandwidth of wireless communication channels
- Short battery lifespan

## Solution

- Cache but the challenge is to let the cache content consistent with the server
- Server periodically broadcast invalidation reports.

## Three research issues

- Content of invalidation report
- How the invalidation is performed
- Support the server provides

## Two different ways

- Develop a framework for designing cache invalidation schemes
- Proposed two schemes that facilitate selective tuning

## Motivations

- Framework approach. Identify the similarities and differences. Identify the issues.
- Comparative evaluation is lacking.
- The issue of minimizing energy consumption is largely ignored.

## Model

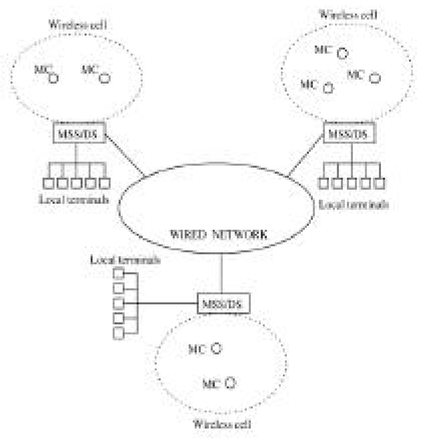


Fig. 1. The wireless computing environment.

## Assumption

- All queries are batched in a query list and are not processed until the MC has invalidated its cache with the most recent invalidation report
- Each server stores a copy of database and broadcasts the same invalidation reports.
- One serve and one cell.

## Two metrics

- Access time: the time elapsed from the moment that client submits a request to the point when all the resultant objects are downloaded by the client
- Energy efficiency
- Tuning time: the time that the client listen on the channel.
- Number of uplink bits transmitted

## Taxonomy of Cache invalidation

- Stateful-based server
- Stateless server (broadcast)
- Asynchronous (notify the client immediately once an update)
- Synchronous (broadcast periodically)
- Issues: Content, invalidation process and server log.

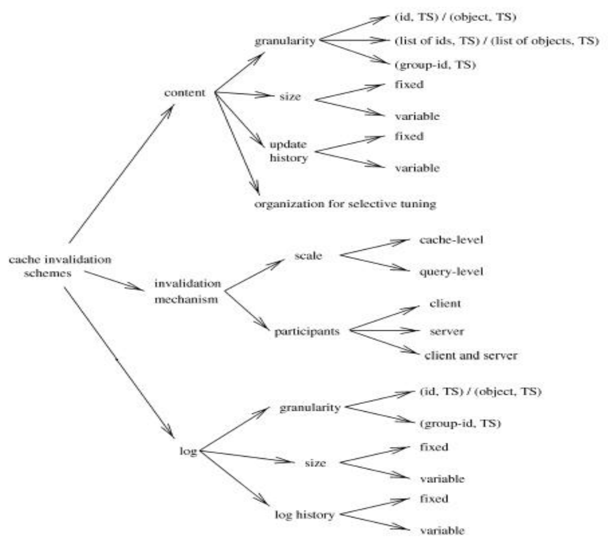


Fig. 2. Issues and solutions in designing cache invalidation schemes.

## Content of invalidation report (1)

- Granularity: the level of details of information each record of the report captures.
- Update invalidation. (Pro: reflect a longer history of updates. Con: request again to obtain the updated copy of cache object)
- Update propagation (Pro: update immediately. Con: big report
- Listed based report. (Pro: reduce the number of timestamps needed. Con: false invalidation)
- Group based report (false invalidation)

## Content of invalidation report (2)

- Report size: fixed or varied
- Update history: the history of the updates that are reflected in the report and can be fixed or varied.
- Affect each other.
- E.g. Fixed no.of object $\rightarrow$ variable update history. Fix update history $\rightarrow$ variable report size.
- [T-wL, T] update history, variable report size. (w is broadcast window)

## Content of invalidation report (3)

- Organization for selective tuning
- Interleaving the content of the report with the indexes

## Invalidation mechanism (1)

- Scale
- Cache level (a single timestamp for all cached object, not suited for selective tuning)
- Query level (each object with a timestamp – last time to be known valid)

## Invalidation mechanism (2)

- Participants
- Server only
- Client only
- Server client collaboration.

## Update Log (1)

- Server side to reflect the updates on the DB
- Granularity
- Size (fixed and variable)
- Log history (fixed and variable)

Fixed size and variable log history (Bit sequence )

[T-WL, T] log history, variable sized logs. (W is update log window)

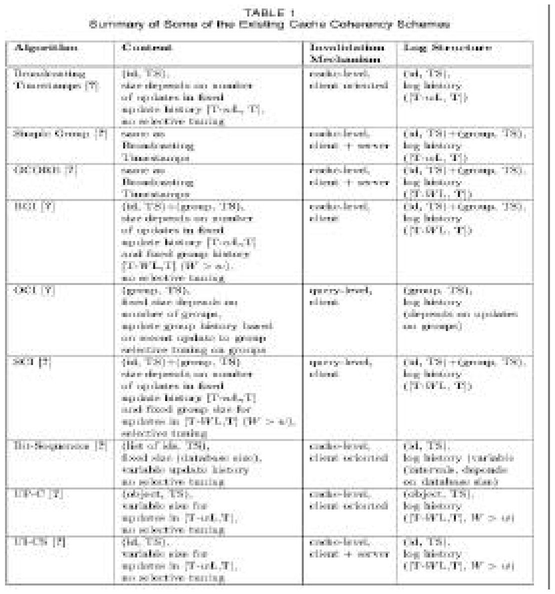


TABLE 1
Summary of Some of the Existing Cache Coherency Schemes

| Algorithm | Content | Invalidation Mechanism | Log Structure |
|---|---|---|---|
| Broadcasting Timestamps [?] | (id, TS), size depends on number of updates in fixed update history [T-wL, T], no selective tuning | cache-level, client oriented | (id, TS), log history ([T-wL, T]) |
| Simple Group [?] | same as Broadcasting Timestamps | cache-level, client + server | (id, TS)+(group, TS), log history ([T-wL, T]) |
| GCORE [?] | same as Broadcasting Timestamps | cache-level, client + server | (id, TS)+(group, TS), log history ([T-WL, T]) |
| BGI [?] | (id, TS)+(group, TS), size depends on number of updates in fixed update history [T-wL,T] and fixed group history [T-WL,T] (W > w), no selective tuning | cache-level, client | (id, TS)+(group, TS), log history ([T-WL, T]) |
| OGI [?] | (group, TS), fixed size depends on number of groups, update group history based on recent update to group selective tuning on groups | query-level, client | (group, TS), log history (depends on updates on groups) |
| SGI [?] | (id, TS)+(group, TS) size depends on number of updates in fixed update history [T-wL,T] and fixed group size for updates in [T-WL,T] (W > w), selective tuning | query-level, client | (id, TS)+(group, TS), log history ([T-WL, T]) |
| Bit-Sequence [?] | (list of ids, TS), fixed size (database size), variable update history no selective tuning | cache-level, client oriented | (id, TS), log history (variable (intervals, depends on database size) |
| UP-C [?] | (object, TS), variable size for updates in [T-wL,T], no selective tuning | cache-level, client oriented | (object, TS), log history ([T-WL,T], W > w) |
| UI-CS [?] | (id, TS), variable size for updates in [T-wL,T], no selective tuning | cache-level, client + server | (id, TS), log history ([T-WL,T], W > w) |

## SELECTED CACHE INVALIDATION SCHEMES

- Dual-Report Cache Invalidation (DRCI)
- Bit Sequences (BS)
- Selective Dual-Report Cache Invalidation (SDCI)
- Bit Sequences with Bit Count (BB)
- **Running example**: use a database of 16 objects and the timestamps at which each object has been updated

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 24 | 16 | 10 | 6 | 22 | 18 | 26 | 32 | 2 | 20 | 14 | 30 | 8 | 4 | 12 | 28 |

## Dual-Report Cache Invalidation (DRCI)

- **Content**. Consists of a list of (object,TS), (group-id,TS).
- **Invalidation Mechanism.** At the cache level, performed by client only
- **Log.** Server maintains logs for objects. Size of logs varies. Log history is fixed at [T-WL,T], where T – current time stamp, $\omega$ and W – update log windows, L – fixed broadcast interval
- The server broadcasts every L time units a pair if invalidation report, an *object invalidation report* (OIR), and a *group invalidation report* (GIR)
- The server keeps tract of all objects updated between [T- $\omega$L,T] and [T-WL,T], where $W>\omega>0$
- The content of OIR is the list ($o_{id}$, $t_{id}$) – the most recent updates during [T-$\omega$L,T]
- The content of GIR is the list ($G_{id}$, $T_{id}$), where $G_{id}$ -- group identifier, $T_{id}$ -- the most recent timestamp at which the group is valid

## DRCI

- $T_{id}$ is determined as follows:

  **Step 1**: when determine the timestamp of a group, ignore all objects already included in OIR

  **Step 2**: among the remaining objects, find the one with the largest timestamp $t < T- \omega L$, $T_{id} = \max(T\text{-}WL, t)$
- Advantages and disadvantages

  - For clients with a small disconnection time, a direct cache is performed using OIR.

  - For clients disconnected before T- ωL OIR is used to invalidate the cache first, then GIR is used to invalidate the remaining objects so that the entire cache doesn't have to be discarded

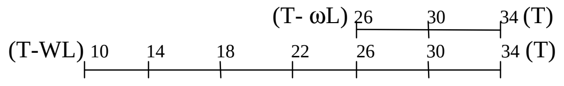
  - Performance is influenced by the grouping scheme, depending on the object categories: hot update - hot demand (HH), hot update - cold demand (HC), cold update – hot demand (CH), cold update - cold demand (CC)

  - No selective tuning, therefore, not energy efficient

## DRCI

- Example: Using the running example. Let T = 34, L =4, ω = 2, W = 6
- Suppose objects are initially split into 4 groups

  G1{o1,o2,o3,o4}, G2{o5,o6,o7,o8}, G3{o9,o10,o11,o12}, G4{o13,o14,o15,o16)



- At time T the update reports are:

  OIR = {34,(o7,26), (o8,32), (o12,30), (016,28)}

  GIR = {(G1,24), (G2, 22), (G3,20), (G4,12)}
- Suppose the mobile client (MC) disconnects at time 23 and reconnects at T=34, timestamp of last valid report is Tc = 22
- Suppose the query is Q = {o1, o2, o6, o7, o9, o12, o14}

## DRCI

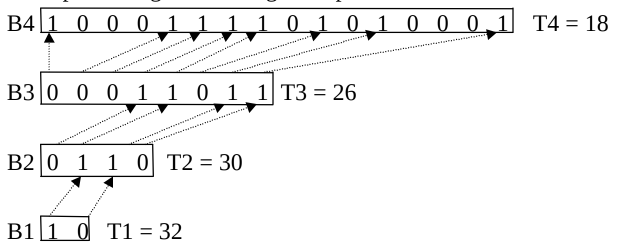
- From OIR, MC invalidates o7 and o12. These objects are removed from the cache
- The remaining objects are given by the resultant groups
  G1{o1,o2}, G2{o6}, G3{o9}, G4{o14}
- From the group timestamp it is determined that all objects in G1 are invalid, all objects in G2, G3, G4 are valid

## Bit-Sequences Scheme (BS)

- **Content.** Consists of a list (id, TS).
- **Invalidation Mechanism.** Performed by the client at cache level
- **Log.** The server keeps track of individual object update information using (id,TS) for up to half the database size
- Let $N = 2^n$. The invalidation report reflects updates for n different times $T_n$, $T_{n-1}$,…,$T_1$, where $T_i < T_{i-1}$, $1 < i \leq n$
- The report comprises n binary bit-sequences, each associated with a timestamp. A “1” bit means the objects has been updated, a “0” bit means the object has not been updated since the time specified by the timestamp of the sequence
- The n bit-sequences are organized as a hierarchical structure with the highest level having N bits and the lowest level having only 2 bits. For sequence $B_{n-i}$, $0 \leq i \leq n-1$, there are $N/2^i$ bits and $N/2^{i+1}$ objects have been updated after the timestamp $T_{n-I.}$ The kth bit in sequence $B_{n-i}$ corresponds to the kth “1” bit in the preceding sequence $B_{n-i+1}$

## BS

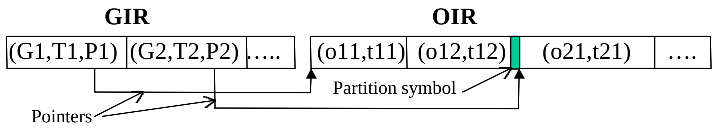
- Example: Using the running example


B4 1 0 0 0 1 1 1 1 0 1 0 1 0 0 0 1 T4 = 18
B3 0 0 0 1 1 0 1 1 T3 = 26
B2 0 1 1 0 T2 = 30
B1 1 0 T1 = 32

- Suppose the query is Q = {o5, o8}
- Suppose the last invalidation report is received at time Tc = 31
- According to the timestamps, B2 is used to invalidate the cache content. The sequences B2-B4 are checked bottom-up
- No selective tuning, therefore, not energy efficient

## Selective Dual-Report Cache Invalidation (SDCI)

- SDCI is different from DRSI in two ways
  - -The report is organized to facilitate selective tuning
  - -The invalidation is query-level based



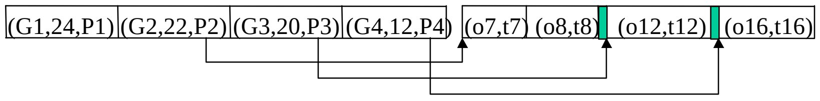
- GIR is broadcast before OIR and in the form (group-id,TS,ptr), where ptr is a pointer reflects the starting position of the objects within this group in OIR
- Entries in OIR are ordered and broadcast based on the group, i.e. updates in the same group will appear together
- A partition symbol separates continuous groups

## SDCI

- At the client the scheme operates as follows:
- For each group queried, it first selectively tunes to the GIR and keeps the pointers of interested groups in memory.
- The timestamp of the last valid invalidation report Tc is compared against the timestamp of a group $T_{id}$, if Tc < $T_{id,it}$ all objects in that group are invalidated
- For the remaining objects in the query, it selectively tunes to the respective position in the OIR by switching to the doze mode until the position p of the pointer of the group containing that object is coming
- The pair ($o_{id}$,$t_{id}$) is downloaded and the timestamp Tc is compared against $t_{id}$, if Tc < $t_{id}$, the object is invalidated, otherwise the object can be used to answer the query

## SDCI

- Example: Using the running example. Let T = 34, Tc = 22
- Suppose the query Q = {o1,o2,o9,o12}



- It first tunes to (G1,T1,P1), from the timestamp all object (o1,o2) are invalidated
- It then tunes to P3, downloads the pair (o12,t12), from the timestamp o12 is invalidated, o9 can be used to answer the query

## Bit-Sequences with Bit Count (BB)

- BB is different from BS in that only the relevant bits need to be examined. This is achieved by associating each bit sequence with a bit count array
- Let N be the number of objects in the database and query Q = ($o_1$,$o_2$,…,$o_q$), ($t_1$,t2,…,tq) are the last valid timestamps respectively
- The bit sequences in BB has the same structure as in BS, however, the examination of the sequences is done in a top-down fashion from $B_n$ to $B_{n-i}$
- For some valid objects, it may be possible to determine their validity and terminate the search before sequence $B_{n-i}$
- Only the relevant bits in each sequence are examined by using a mechanism that can count the number of "1" bits in a sequence without examining the entire sequence as follow:

  -Each bit sequence is associated with a bit count array

## BB

-For bit sequence $B_{n-i,}$ $0 \le i \le n-1$, the sequence is partitioned into packets of $2^j$ bits.

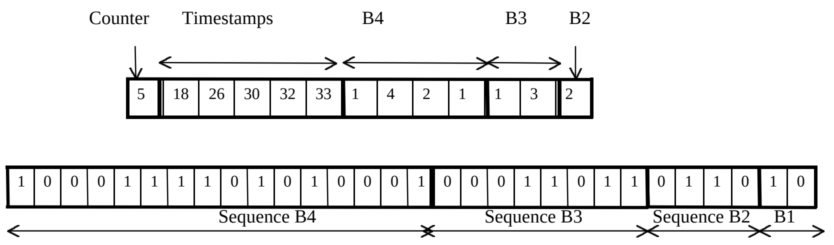
-For sequence $B_{n-i}$, the number of array entries is $N/2^i/2^j$, The sequences with fewer than j bits do not need to be associated with a bit count array as all bits need to be examined

-The kth entry in the bit count array of sequence $B_{n-i}$ represents the number of "1" bits in the kth package in the sequence

- Selective tuning is done as follow: Let packet i contains the bit in which we are interested. From the bit array count, we can determine the number of "1" bit that has been set for packet 1 to i-1. The client can then tune into the ith packet and scan the ith packet until the relevant bit. It determines whether the relevant bit is "1" or "0"
- By compute the number of "1" bits this way it then determines the position of the interested bit in the next sequence. This process is repeated until sequence $B_{n-.}$ is reached

## BB

- The process can be terminated if a “0” bit is encounter at any of the sequences from $B_n$ to $B_{n-i}$ If the relevant bit at $B_{n-i}$ is “1”, then the object is invalid, otherwise, it is valid
- The invalidation report is organized as follow: the counter is broadcast first, the timestamps are broadcast next, followed by the bit count arrays for sequences $B_n$, $B_{n-1}$, …, finally,the bit sequences $B_n$, $B_{n-1}$, $B_1$
- Example: Using the running example



## BB

- Assume query Q = (o5, o8) with cached timestamp are, respectively, 31 and 27.
- From the timestamp the MC needs to check B2 for o5 and B3 for o8
- From the first bit count array entry of B4, the MC knows that there is only one “1” bit among the 1st four objects. It tunes to the beginning of the second packet of B4 and examine the first bit till the 4th bits. It determines that o5 is the 2st bit in B3 and 08 is the 5th in B3
- For o5, the MC examines the 2nd bit in B3 which is set to “0” indicating that o5 is valid and stop the search
- For o8, the MC examines the bit count array for B3. The 1st entry contains “1”. By examining the 2nd bits of the second packet of B3 it determines that the bit corresponding to o8 is set to “1”. This means o8 is the second entry in B2. By examining the second bit of B2 it determines that o8 is invalid

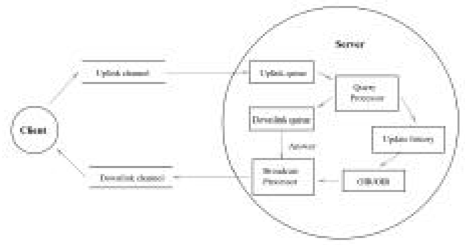
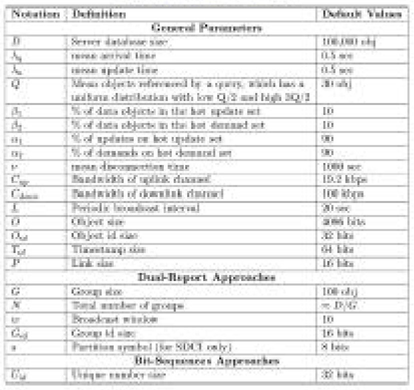


Fig. 11. Mobile communication model.

TABLE 2
System and Workload Parameters



| Notation | Definition | Default Values |
|---|---|---|
| | **General Parameters** | |
| $D$ | Server database size | 100,000 obj |
| $\lambda_q$ | mean arrival time | 0.5 sec |
| $\lambda_u$ | mean update time | 0.5 sec |
| $Q$ | Mean objects referenced by a query, which has a uniform distribution with low Q/2 and high 3Q/2 | 20 obj |
| $\beta_1$ | % of data objects in the hot update set | 10 |
| $\beta_2$ | % of data objects in the hot demand set | 10 |
| $\alpha_1$ | % of updates on hot update set | 90 |
| $\alpha_2$ | % of demands on hot demand set | 90 |
| $\nu$ | mean disconnection time | 1000 sec |
| $C_{up}$ | Bandwidth of uplink channel | 19.2 kbps |
| $C_{down}$ | Bandwidth of downlink channel | 100 kbps |
| $L$ | Periodic broadcast interval | 20 sec |
| $O$ | Object size | 4096 bits |
| $O_{id}$ | Object id size | 32 bits |
| $T_{id}$ | Timestamp size | 64 bits |
| $P$ | Link size | 16 bits |
| | **Dual-Report Approaches** | |
| $G$ | Group size | 100 obj |
| $N$ | Total number of groups | $= D/G$ |
| $w$ | Broadcast window | 10 |
| $G_{id}$ | Group id size | 16 bits |
| $s$ | Partition symbol (for SDCI only) | 8 bits |
| | **Bit-Sequences Approaches** | |
| $U_{id}$ | Unique number size | 32 bits |

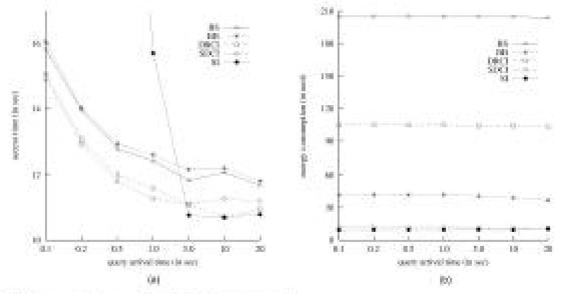


Fig. 12. Effect of query arrival time. (a) Access time. (b) Energy consumption.

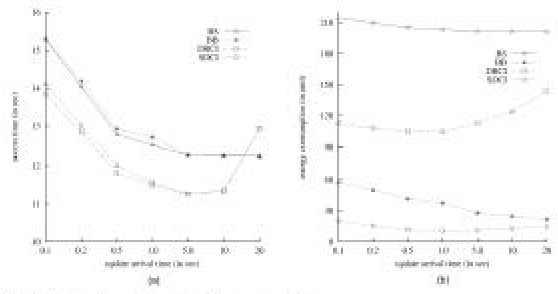


Fig. 13. Effect of update arrival time. (a) Access time. (b) Energy consumption.

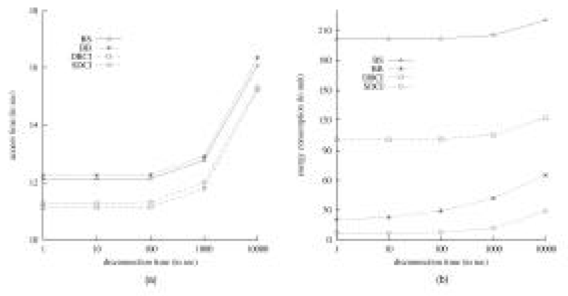


Fig. 14. Effect of disconnection time. (a) Access time. (b) Energy consumption.

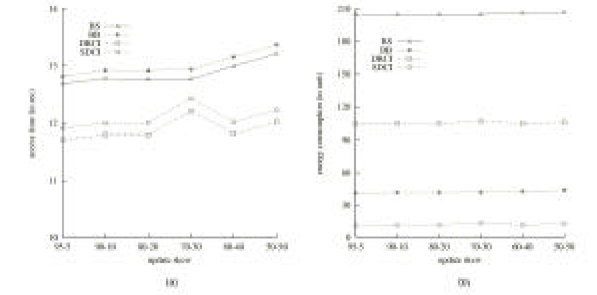


Fig. 15. Effect of update skew. (a) Access time. (b) Energy consumption.

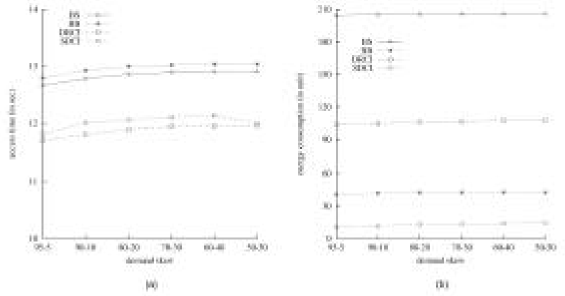


Fig. 16. Effect of demand skew. (a) Access time. (b) Energy consumption.

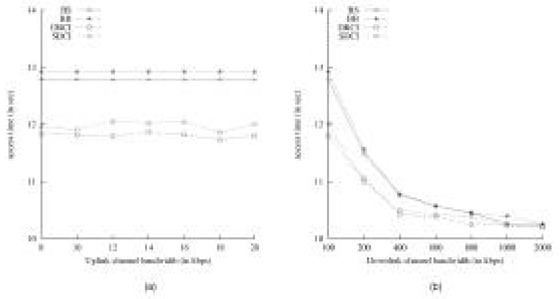


Fig. 17. Effect of channel bandwidth. (a) Uplink channel. (b) Downlink channel.