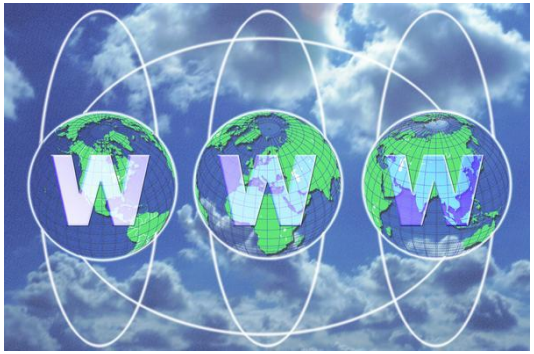
Mobile Caching

- Why mobile caching?
- Cache consistency mechanisms
- Cooperative caching in multi-hop wireless Internet access

Why mobile caching?

- In a mobile wireless environment, how to reduce latency in accessing data from servers, or how to overcome data unavailability due to disconnections?
- Caching and hoarding the data record at the client device provides an answer.
  - Cache the pushed hot data
  - Pre-fetching
  - Hoarding



Performance metrics of caching

- **Goal:**
  - Cache the most probable data item for best future use, given limited amount of cache storage and potential changes in data item values.
- **Performance**
  - Cache hit ratio
    - Percentage of accesses that results in cache hit, or percentage of total volume of data retrieved from cache to total volume of requested data.
  - Access response time
    - Average service time to complete the data item request, starting from the time when data is requested to the time the data becomes available
  - Message / energy cost
- **Correctness**
  - Consistency

Client caching in data broadcast

- **Data are cached at clients to improve access time; also lessen dependency on server's choice of broadcast priority**
  - Clients cache "hottest" data to improve hit ratio
- **In general, the cost of obtaining a page on a cache miss is considered constant. However, this is not the case in data broadcast:**
  - The cost of servicing a miss on a page depends on when the requested page will appear on the broadcast
  - Cost-based page replacement policies are recommended

Client caching in data broadcast

- ***Cost-based Page Replacement***
  - The cost of obtaining a page on a cache miss is taken into account when deciding which pare in the cache to replace.
  - *The PIX method*: replace the page having the lowest P/X ratio: ratio between probability of access (P) and frequency of broadcast (X) of the page.
    - Optimal under certain conditions, but not practical - it requires perfect knowledge of access probabilities and comparison of pix values for all pages.

Client caching in data broadcast

- ***The LIX method*: LRU with broadcast frequency**
  - Pages are placed on lists based on their broadcast frequency (*X*)
  - Lists are ordered based on *L*, the running avg. of inter-access times
  - Page with lowest *L/X* is replaced

Client cache invalidation

- **Value of data may have changed since caching. Cache consistency ensures that copies of a data record are identical at the server and at the device cache.**
- **Consistency mechanism: server broadcasts invalidation information to its client.**
- **When?**
  - *Synchronous:* send invalidation reports periodically
  - *Asynchronous:* send invalidation information for an item, as soon as its value changes

Client cache invalidation

- **To whom?**
  - *Stateful server*: sent to affected clients
    - server maintains information about its clients, e.g., the contents of their cache and when it was last validated
  - *Stateless server*: broadcast to everyone
- **What to send?**
  - *invalidation*: only which items were updated
  - *propagation*: the values of updated items are sent
  - aggregated information/ materialized views

Cache consistency maintenance

- **Polling mechanism (client-initiated case):**
  - The client checks from the server about the state of data record: valid, invalid (expired, modified), etc.
- **Pushing mechanism (server-initiated case):**
  - Just like in client cache invalidation, when a cached data item becomes invalid and thus unusable, the server sends invalidation reports, either synchronously or asynchronously.

Polling method

- **Each cached record copy is polled (checked) whenever required**
  - If data copy becomes invalid at the device or has been modified at the server, the device requests for the modified data for replacement.
- **Time-to-live mechanism**
  - Each cached record is assigned a TTL (time-to-live)
  - The TTL assignment is adaptive (adjustable) to previous update intervals of that record
  - After TTL expires, the client device polls for invalid or modified state
  - If invalid or modified state found then the device requests the server to replace
  - When TTL is set to 0, the TTL mechanism is equivalent to the polling mechanism

Pushing methods

- **Four cache invalidation mechanisms**
  - Stateless asynchronous
  - Stateless synchronous
  - Stateful asynchronous
  - Stateful synchronous

Stateless asynchronous

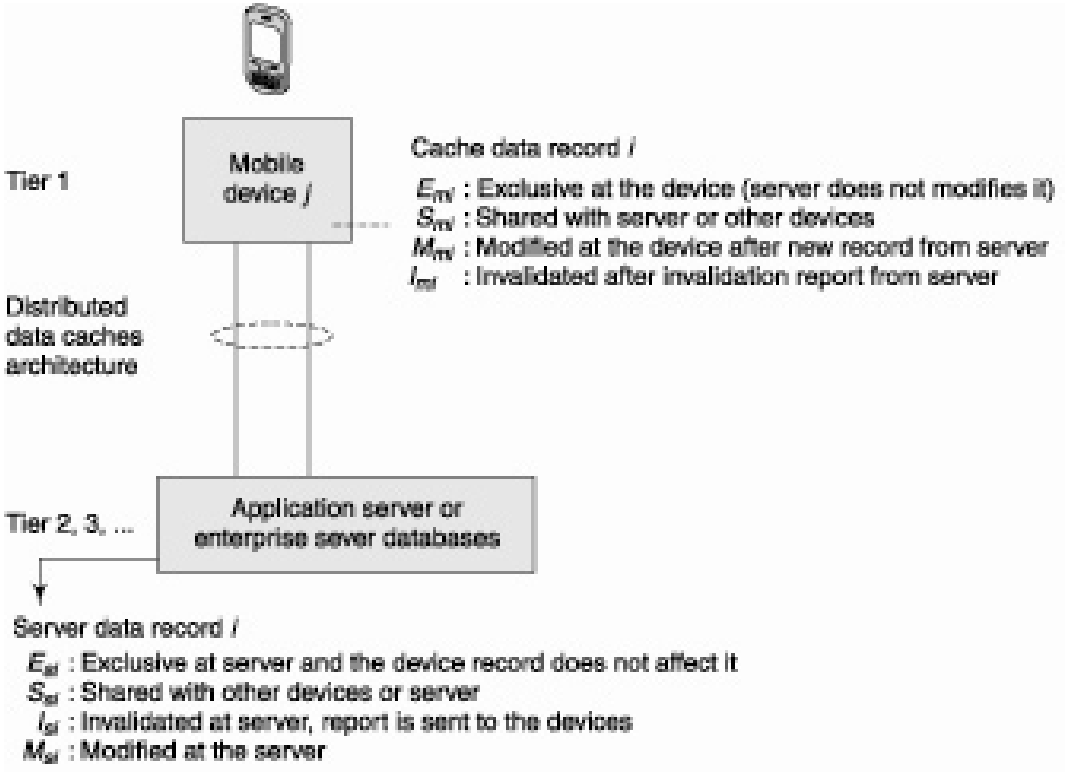
- Device cache states are not maintained at the server
- The server advertises invalidation report
- On receiving an invalidation report, the client requests for or caches the new data record
- Advantages
  - No frequent, unnecessary transfers of data reports
  - Mechanism more bandwidth efficient
- Disadvantages
  - Every client device gets an invalidation report, whether that client requires that copy or not
  - Client devices presume that as long as there is no invalidation report, the copy is valid for use (how about link failure?)

Stateless synchronous

- Server does not maintain device cache states, and periodically advertises invalidation report
- Client requests for or caches the data on receiving the invalidation report
  - If no report received till the end of the period, client requests for the report
- Advantages
  - Client receives periodic information regarding invalidity of the data caches, leading to greater reliability of cached data. This also helps server and devices maintain cache consistency through periodical exchanges
- Disadvantages
  - Unnecessary transfers of data invalidation reports – every client gets invalidation report periodically, irrespective of whether it has a copy of the invalidated data or not
  - During the period between two invalidation reports, client may have inconsistent data.

Stateful approach

- **Four possible states (*M, E, S, or I)* of a data record *i* at any instance at the server database and device *j cache***
  - Entail that each data record in a cache has a tag to specify its state at any given instant and the tag is updated (modified) as soon as the state of the record changes



Stateful asynchronous

- **Server keeps track of device cache states and transmits invalidation report to concerned devices when update occurs**
- **Client requests for new data on receiving the invalidation report and sends its new state after caching new record from the server**
- **Advantages**
  - Only affected clients receive the invalidation reports and other devices are not flooded with irrelevant reports
- **Disadvantages**
  - Client devices presume that, as long as there is no invalidation report, the copy is valid for use in computations.
  - Therefore, when there is a link failure, then the devices use invalidated data

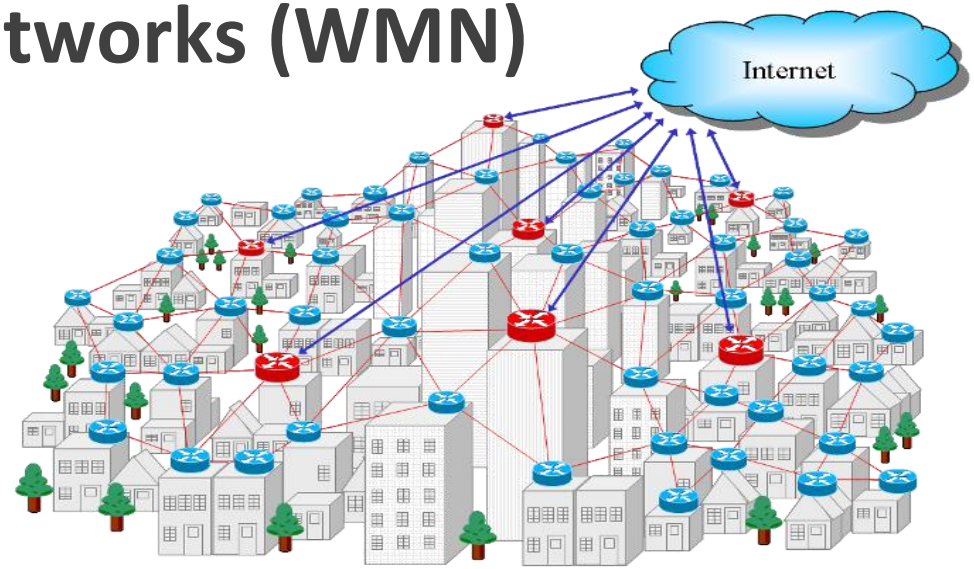
Stateful synchronous

- **Server maintains device cache states, and periodically transmits invalidation report to the concerned devices when update occurs**
- **Client requests for new data on receiving the invalidation report, and sends its new state after caching the new data.**
  - If no report is received till the end of the period, the client requests for the report.
- **Advantages**
  - Enables server to synchronize timely with the client device when invalid data gets modified and becomes valid.
- **Disadvantages**
  - High bandwidth requirement to enable periodic transmission of invalidation reports to each device and updating requests from each client device

Cooperative Caching in Multi-hop Wireless Internet Access

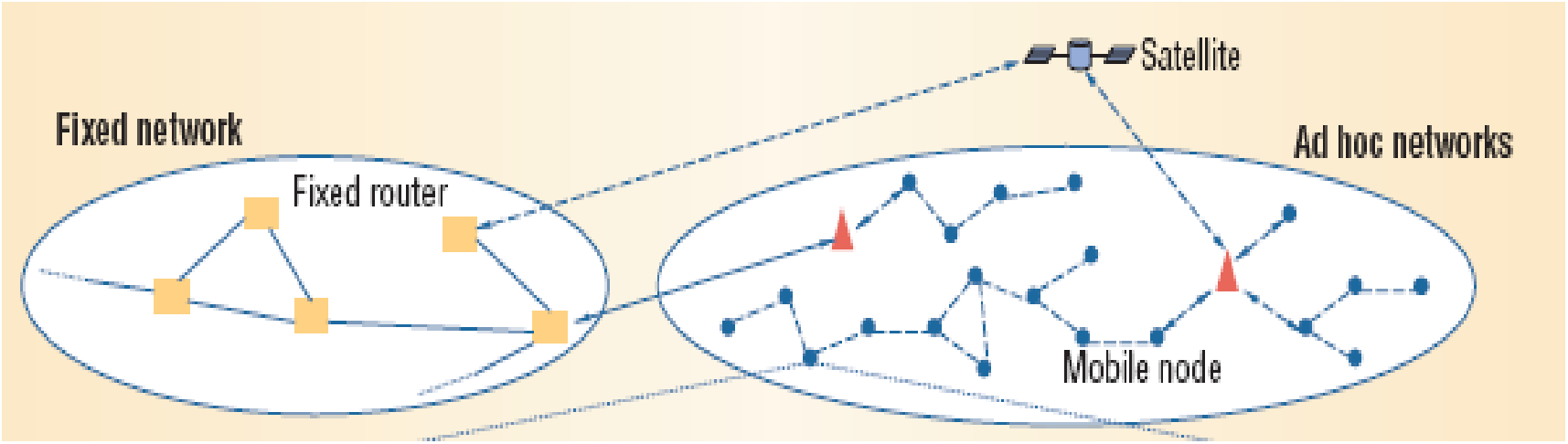
Multi-hop Wireless Internet access

- **Pervasive Internet access: extending access to the Internet**
  - Combining both wired and wireless networks
  - Hybrid of infrastructured and ad hoc wireless networks
- **e.g., wireless mesh networks (WMN)**
  - community networks



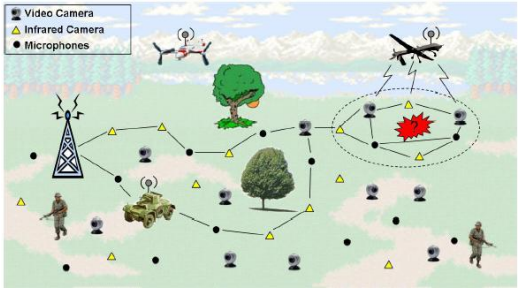
Multi-hop Wireless Internet access

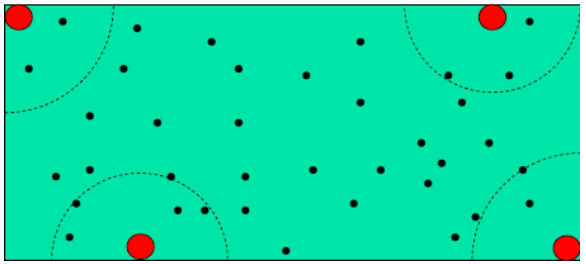
Internet-based MANET (I-MANET)



Data dissemination in I-MANET

- Disseminate data from the **data source** to the **querying node** in multi-hop wireless Internet access
- Scenarios
  - Soldiers equipped with mobile terminals
    - Receiving and exchanging geographic info, enemy info etc.
  - Mobile store consisting of several mobile booths
    - Exchanging info of commodities such as price and sum





Data dissemination in I-MANET

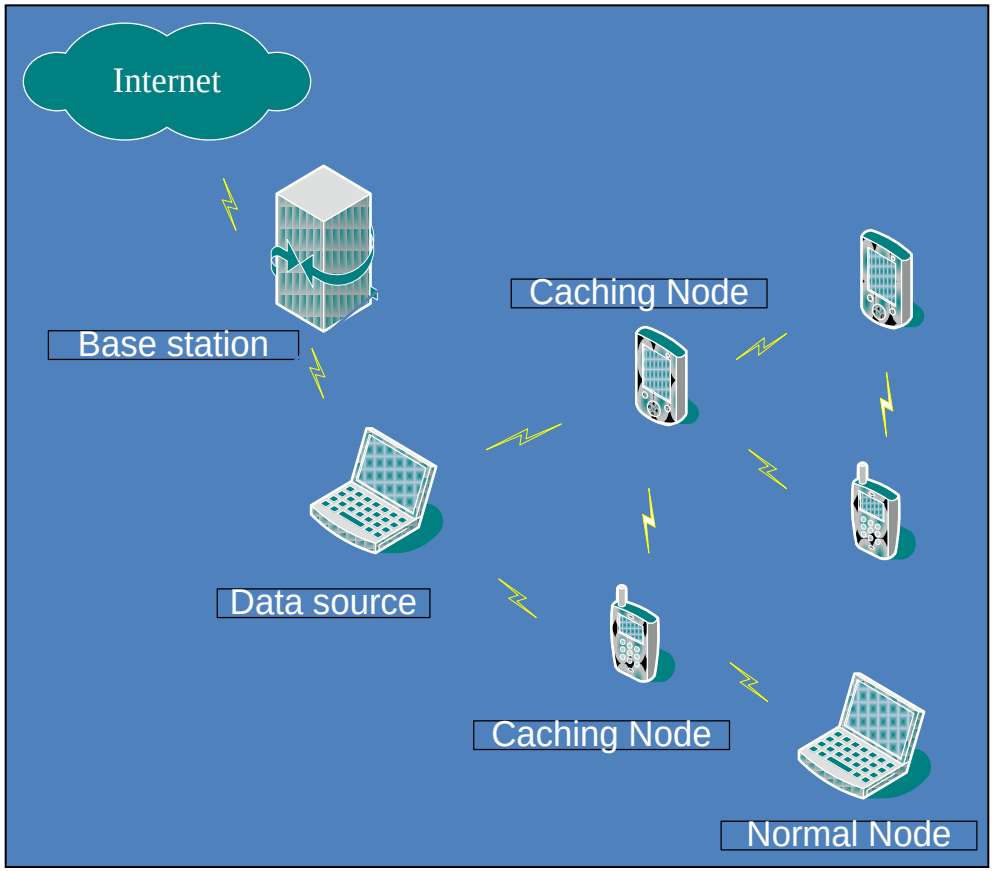
- **Requirements**
  - Short query delay
  - Low traffic overhead
- **Caching**
  - Cache popular data on the querying node
  - Reduce traffic overhead and query delay

Cooperative caching

- **A collection of nodes cooperate to cache and disseminate the data**
  - Cache more popular data, overcoming the limit in size of cache at individual nodes
    - Serve queries without having to frequently send request to data source
    - Shorter delay and less communication overhead
  - Need coordination and sharing of cached data

System model

- **Base station**
  - Gateway to outside world, e.g., Internet
  - Often serve as Data source
- **Caching nodes**
  - Holding the cached data
- **Cooperation among the caching nodes**



Challenges

- Multi-hop reachability from data source
- Bandwidth constraints
- Disconnection
- Topology changes and peer dynamism
- Resource constraints on mobile nodes

→ New protocols are needed

Major issues

- **Cache placement**
  - Determine where to place copies of data item and when to replace them
- **Cache discovery**
  - Direct the query to an appropriate caching peer
- **Caching consistency**
  - Maintain desired level of consistency among data copies

Cache placement

- **Who should cache?**
  - How to select the caching nodes
    - e.g., every node caches
    - e.g., nodes with more power cache, or more stable
- **What should be cached?**
  - Which data items should be cached?
  - **CacheData** vs. **CachePath**: caching the data or caching the path to nodes holding the data?

L. Yin and G. Cao, *Supporting Cooperative Caching in Ad hoc Networks*, IEEE Trans. on Mobile Computing, Vol. 5, No. 1, 2006.

Cache Replacement

- **Some cached data items need to be replaced due to limited cache size**
- **Cache replacement strategies**
  - Temporal locality-based
    - Keep more recently accessed, more likely to be accessed again
    - e.g., LRU: Least Recently Used
  - Benefit-based
    - Minimize the expected cost
    - e.g., PIX: data with less p / x, more likely to be replaced
      p: probability that the data item will be accessed
      x: frequency that the item is updated
  - Semantic-based
    - Semantic relationship among cached data on different nodes

Cache discovery

- **Forward queries to the caching node that holds the requested data or the data source**
- **Most methods are based on existing routing protocols**
  - Broadcast with TTL
  - Expanded Ring
  - Cluster head maintain caching digests of member nodes

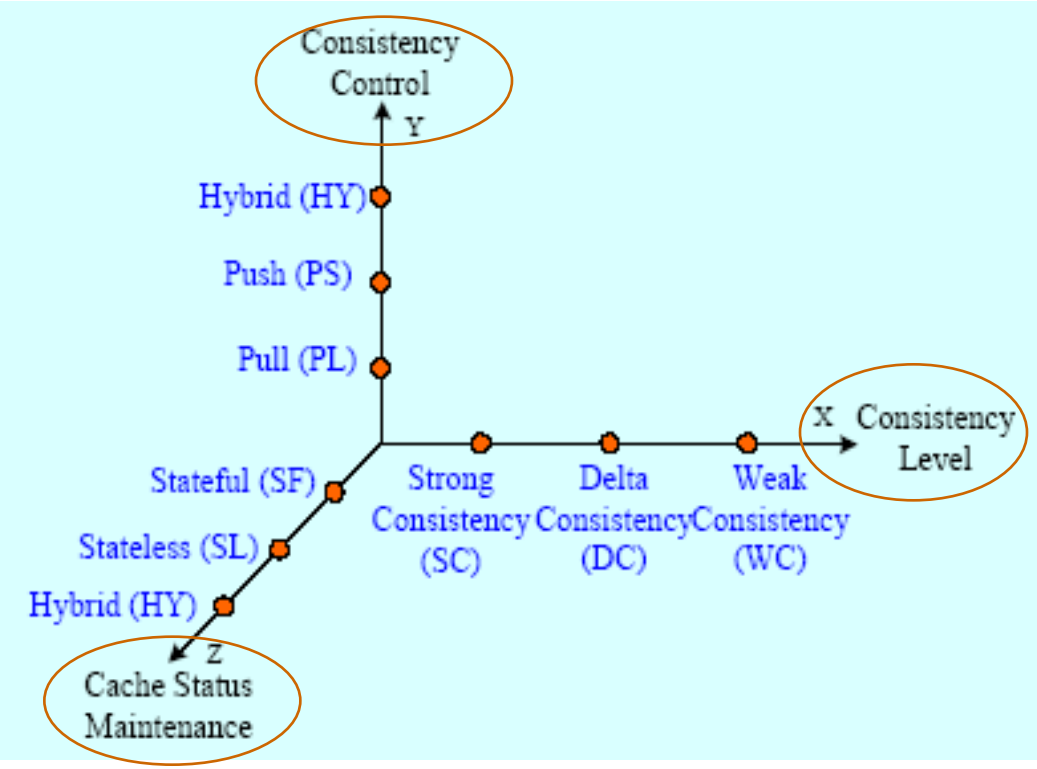
Caching consistency

- **Measure the deviation between the source data and the cached data**
  - Data source updates the data
  - Cached data serves queries on the data
- **Applications have their requirements on consistency between the source data and the cached data, e.g.**
  - the cached weather information can not be stale for more than 1 hour
  - the cached stock price can not be inaccurate by more than $1

Our focus

Cache Consistency

A Design framework



J. Cao, Y. Zhang, L. Xie and G. Cao, *Data Consistency for Cooperative Caching in Mobile Environments*,
IEEE Computer April 2007.

Consistency levels

- **Two extremes**
  - Strong consistency (SC)
  - Weak consistency (WC)
- **Strong consistency is too expensive and not always necessary, especially in I-MANET**
- **Weak consistency is a *best-effort* guarantee**

Consistency levels

- **Modeling the consistency spectrum**
  - **Delta consistency (DC)**
    - Deviation in values / time bounded by δ
    - e.g., the cached data is never stale for more than 60s
    - e.g., difference between the cached stock price and the latest price is never more than $1
  - **Probabilistic consistency**
    - Guarantee given consistency level with specified probability *p* – ratio of valid cache access ≥ *p*
    - e.g., guarantees strong consistency with probability of 90% (for all the queries)

Consistency control Initiation

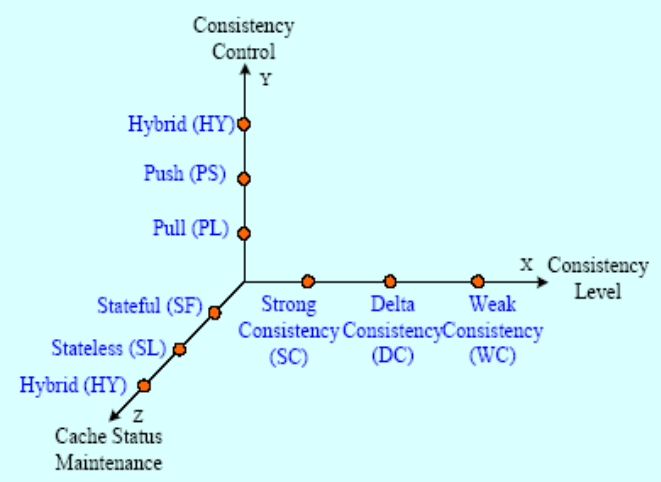
- **Push**
  - Initiated by the data source
  - One-way traffic but long query latency
  - Suitable to a stable network
    - Disconnection causes inconsistency
  - Can be periodical or upon update
  - Can push real data or IR
  - Can be broadcast or multicast (with TTL)
- **Pull**
  - Caching node-initiated
  - Round-trip traffic and may cause large traffic cost if many nodes pull
  - Suitable to dynamic network
  - Consume more battery power on mobile nodes
  - Can pull upon each query or TTR-controlled (and *adaptive* TTR)
- **Hybrid**
  - Combine Push and Pull

Cache status maintenance

- **Stateful**
  - Data source maintains info about caching nodes
  - Heavy-weight on data source
- **Stateless**
  - Data source maintains no info about caching nodes
- **Hybrid**
  - Maintain info of part of the caching nodes, or
  - Migrate between stateful and stateless modes from time to time

Space of protocol design

- **Covering existing protocols, e.g.**
  - SC-PL-SL: Pull upon each query
  - DC-PL-SL: Cached data with TTR, Pull when TTR expires
  - SC-PS-SF: Push IR with ACK
  - WC-PS-SL: Push updated data
  - SC-HY-SF: Lease protocol
- **Guiding the design of new protocols**
  - *-HY-HY
    - How to efficiently combine Push and Pull to support multiple consistency levels?
    - How to support heterogeneous consistency requirements of different users?



Space of protocol design

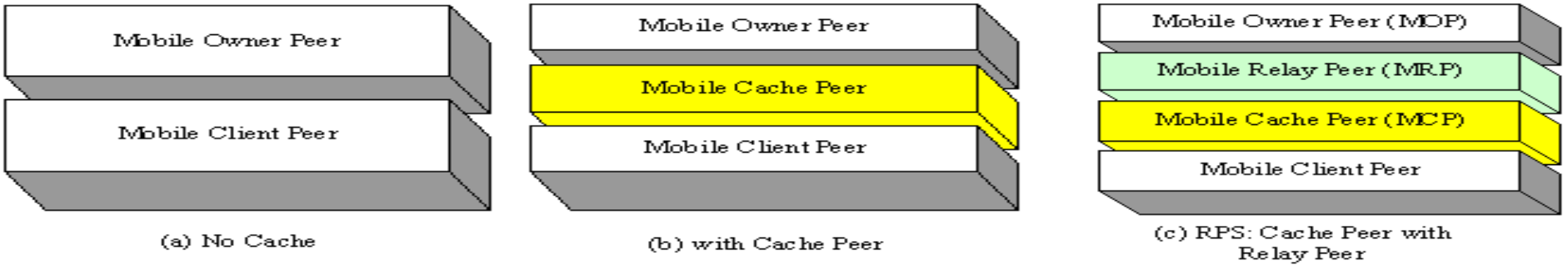
- **Hybrid protocols with combination of Push and Pull**
  - Adaptively adopt Push or Pull
    - Most existing protocols do this
    - Complexity in adaptation causes overhead on MH
  - **Adopt both Push and Pull (our work)**
    - Using a 2-layer hierarchy
      - The Relay Peer-based Approach (RPCC)
    - Using prediction (dynamically adaptive)
      - Predictive Caching Consistency (PCC)
    - Using time-out
      - The Flexible Combination of Push and Pull Algorithm (FCPP)

Relay Peer-based Caching Consistency (RPCC)

J. Cao, Y. Zhang, L. Xie and G. Cao, *Consistency of Cooperative Caching in Mobile Peer-to-peer Systems over MANET*,

Intl. J. of Parallel, Emergent and Distributed Systems, Vol. 21, No. 3, June 2006

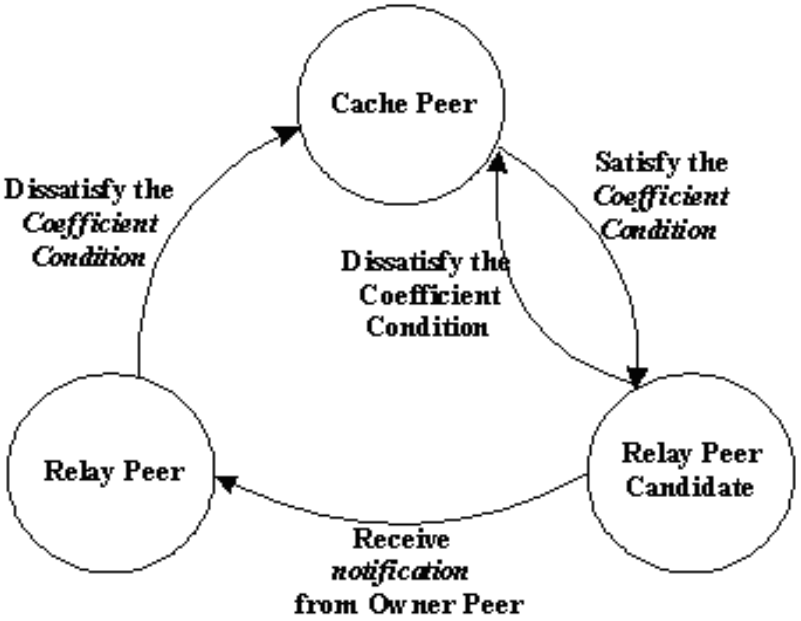
Overview of RPCC



- **Introducing *relay peers* between data source and normal cache nodes**
  - Data source periodically Push IR to relay peers (relay nodes are more stable)
  - Normal caching nodes Pull relay peers

Overview of RPCC

- **Select relay peers, considering**
  - Peer access rate
  - Stability
  - Energy
- **Role transformation**
  - Normal caching nodes <-> Relay peer
- **Consistency control protocol**
  - Source node (TTNotify)
  - Relay peer (TTRrefresh)
  - Normal caching node (TTPull)
- **Meeting different consistency levels by TTR and TTP on relay and cache peers**
- **Handling disconnection / reconnection**



RPCC Algorithms

```
(1)  IF TTN_d =0                    //at invalidation interval
(2)     IF data item is updated during this period
(3)           For all peers ∈ RP_d;
(4)           Send UPDATE to these relay peers ;
(5)     END IF
(6)     Broadcast INVALIDATION;
(7)     Renew TTN_d;
(8)  END IF

(9)  IF receive GET_NEW     //relay peer misses the last update message
(10)    Send back SEND_NEW to such kind of relay peers;
(11) END IF

(12) IF receive APPLY       //candidate wants to promote to relay peer
(13)    Send back APPLY_ACK;
(14)    Add this peer to its relay peer table ;
(15) END IF

(16) IF receive CANCEL or the destination peer of APPLY_ACK unreachable
(17)    Remove the peer from its RP_d;
(18) END IF
```

G%'QRS:H- 1!>D&8&A&?a!F&4DA1!>11DF#21!

```
(1)  IF receive INVALIDATION          //from owner peer
(2)    Compare its LVER_i with VER_d ;
(3)       IF LVER_i < VER_d           //not up-to-date
(4)          Send GET_NEW to owner peer for the latest version ;
(5)       ELSE renew TTR_i;
(6)       END IF
(7)  END IF

(8)  IF receive POLL                  //from cache peer
(9)    IF TTR_i>0                     //up-to-date (at relay peer)
(10)     Compare the LVER_i with VER_i;
(11)        IF LVER_i = VER_i        //up-to-date (at cache peer)
(12)           Send back POLL_ACK_A;
(13)        ELSE                     //stale (at cache peer)
(14)           Send back POLL_ACK_B;
(15)        END IF
(16)   ELSE                           //stale (at relay peer)
(17)     Wait for the INVALIDATION message ;
(18)   END IF
(19) END IF

(19) IF receive SEND_NEW              //from owner peer
(20)    Update the data item ;
(21)    Renew TTR_i;
(22) END IF

(23) IF receive UPDATE                //from owner peer
(24)    Update the data item ;
(25) END IF
```

G%'QRS:H- 1!>D&8&A&?a!K1?$@>11DF#21!

RPCC Algorithms

```
(1)  IF receive query request
(2)    IF (CL_d=搶eak□)
(3)          Answer the query immediately ;
(4)    ELSE
(5)      IF (CL_d=撮elta? and TTP_d>0)
(6)          Answer the query immediately ;
(7)      ELSE
(8)          Broadcast POLL;
(9)      END IF
(10)   END IF
(11) END IF

(12) IF receive POLL_ACK_A          //unchanged
(13)   Answer the query immediately ;
(14)   Renew TTP_d;
(15) END IF

(16) IF receive POLL_ACK_B          //changed
(17)   Update the data item ;
(18)   Answer the query;
(19)   Renew TTP_d;
(20) END IF

(21) IF want to be a relay peer     //candidates
(22)   Send APPLY to owner peer;
(23) END IF

(24) IF receive APPLY_ACK           //acknowledgment from owner peer
(25)   Change its status from cache peer to relay peer ;
(26) END IF

(27) IF receive UPDATE              //from owner peer
(28)   IF relay peer candidate      //it misses the APPLY_ACK
(29)     Change its status to relay peer ;
(30)     Update its data item ;
(31)     Renew TTP_d;
(32)   Else                         //the owner peer misses the CANCEL
(33)     Update its data item ;
(34)     Send CANCEL to owner peer;
(35)     Renew TTP_d;
(36)   END IF
(37) END IF
```

Advantages of RPCC

- Explore heterogeneity of mobile nodes and cooperation among caching nodes
- Reduce query delay & communication overhead (push & pull done asynchronously and simultaneously)
- Meeting different consistency levels (SC, DC, WC)
- Adaptive and scalable

Predictive Caching Consistency (PCC)

Y. Huang, J. Cao, B. Jin, “Cooperative cache consistency maintenance for pervasive internet access”,

WIRELESS COMMUNICATIONS AND MOBILE COMPUTING, 2010; 10:436–450

Overview of PCC

- **Query log can be used to predict future requests**
- **How can predictive methods help?**
  - Pull only when the cache copy is probably stale
  - Push only when large # of caching nodes will pull
  - Achieve dynamic adaptability
- **The PCC algorithm**
  - Predictive Push and Pull
  - Weak consistency
    - Sacrifice consistency to save cost
    - Mobility and resource constraints in I-MANET

Prediction in PCC

- **Overview of the prediction method**
  - Asymptotically perfect prediction
    - also used in the LeZi-Update* algorithm
  - Represent the history of Pull and Push by *strings*
  - Build a dictionary of *terms* from the strings
  - Record how many times every term/symbol appears
  - Calculate probability of occurrence of each term/symbol
  - Predict probability that given event will happen

* A. Bhattacharya and S. Das, *LeZi-Update: An Information-Theoretic Framework for Personal Mobility Tracking in PCS Networks, Wireless Networks*, Kluwer Academic Publishers, vol. 8, 2002.

Predictive push on data source

- **Data source maintain history of pull from caching nodes**
  - e.g., 102310 in the figure
- **Predicts expected number of pulls between the latest and the forthcoming update**
- **Decides to push if the ratio of pulling caching nodes is more than given threshold**
  - e.g., Push if more than 80% of the caching nodes are expected to pull

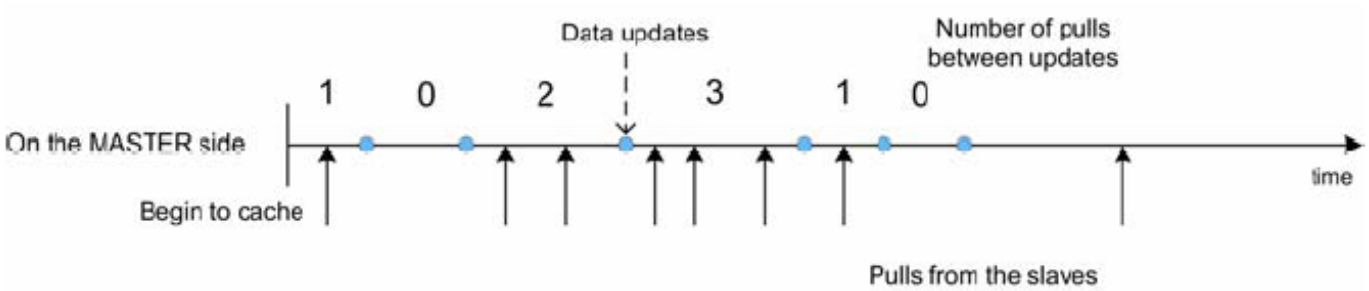


Fig 1. Maintaining the history of updates on the master copy and pulls from the slaves

Predictive pull on caching node

- **Caching node maintains history of updates on the source data**
  - e.g., 120121 in the figure
- **Predicts "no updates will occur between *n+1* consequent queries"**
  - the term of *n* consequent *zero*
- **Decides not to pull but directly answer for the next *n* queries if probability of the n-zero-term is great enough**

Fig 2. Maintaining the history of updates on the master copy and queries on the cache copy

Performance evaluation

- **Performance metrics**
  - *Consistency*: how long the cache copy has been stale
  - *# of relay nodes*: traffic overhead (hop count of control messages) and energy consumption on mobile nodes
- **Evaluate the performance under different conditions**
  - Varying the physical environment
    - Network size
    - Speed of node movement
  - Varying the logical environment
    - Average frequency of source data update
    - Cached data access pattern

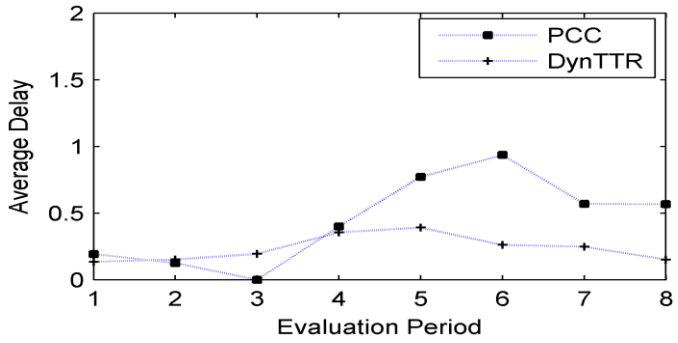
Performance comparison

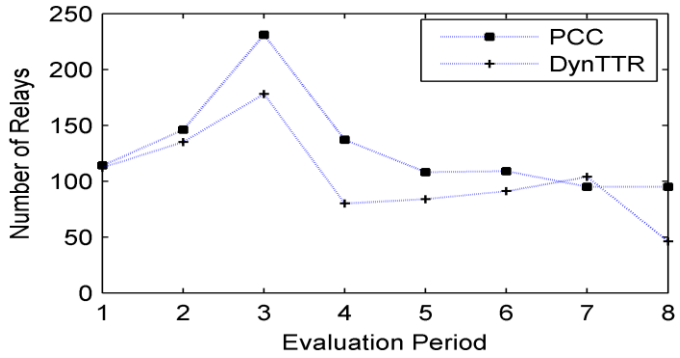
- **Compared with DynTTR***
  - Used in traditional P2P networks
  - Asynchronous push by data source
  - TTR (Time to Refresh) for each cache copy
  - Use a simple linear model to predict the updates and queries
    - If source data updated during TTR
      - TTR decreased by a multiplicative factor
    - If source data not updated during TTR
      - TTR increased by a constant value

* J. Lan, X. Liu, P. Shenoy and K. Ramamritham, *Consistency Maintenance in Peer-to-Peer File Sharing Networks*, 3rd IEEE Workshop on Internet Applications, 2003.

Performance comparison

- **The "anchor" experiment**
  - Network size: 80
  - Node velocity: 1.5 m/s
  - Average update interval: 10 s
  - Query ratio: 90% (ratio of querying nodes between two updates)
- **DynTTR is specially tuned for this given environment**
  - Better performance in both consistency and relay counts

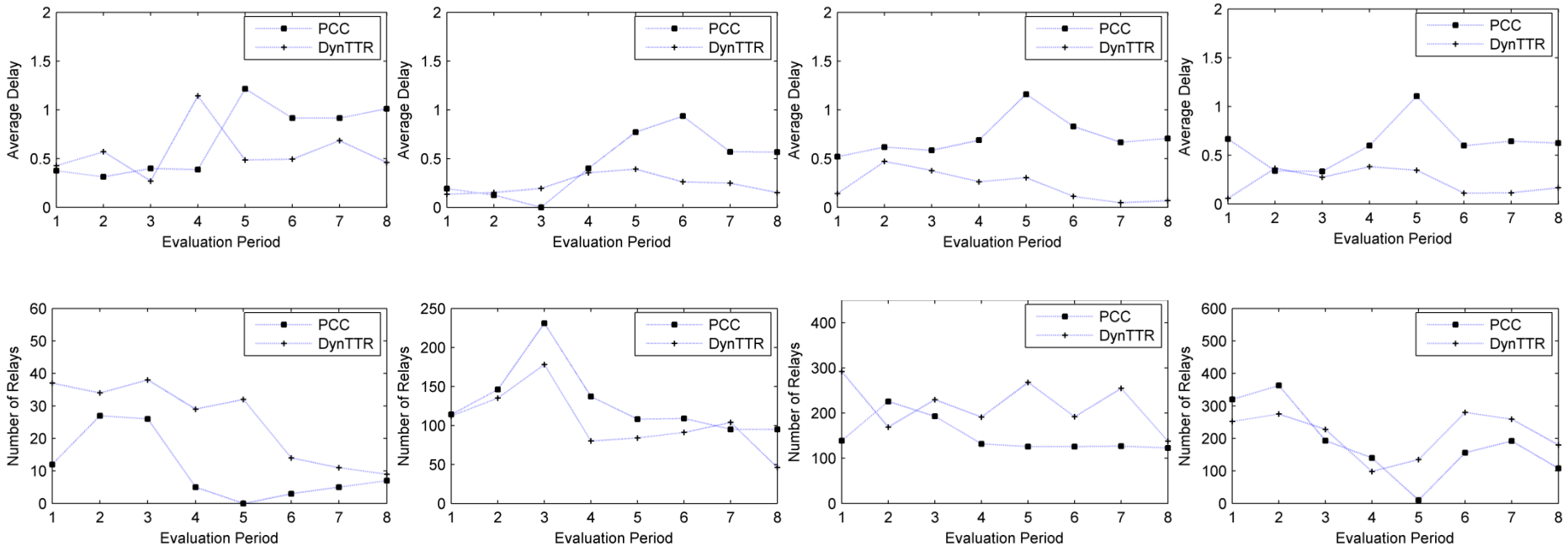




- **What if the environment conditions change?**

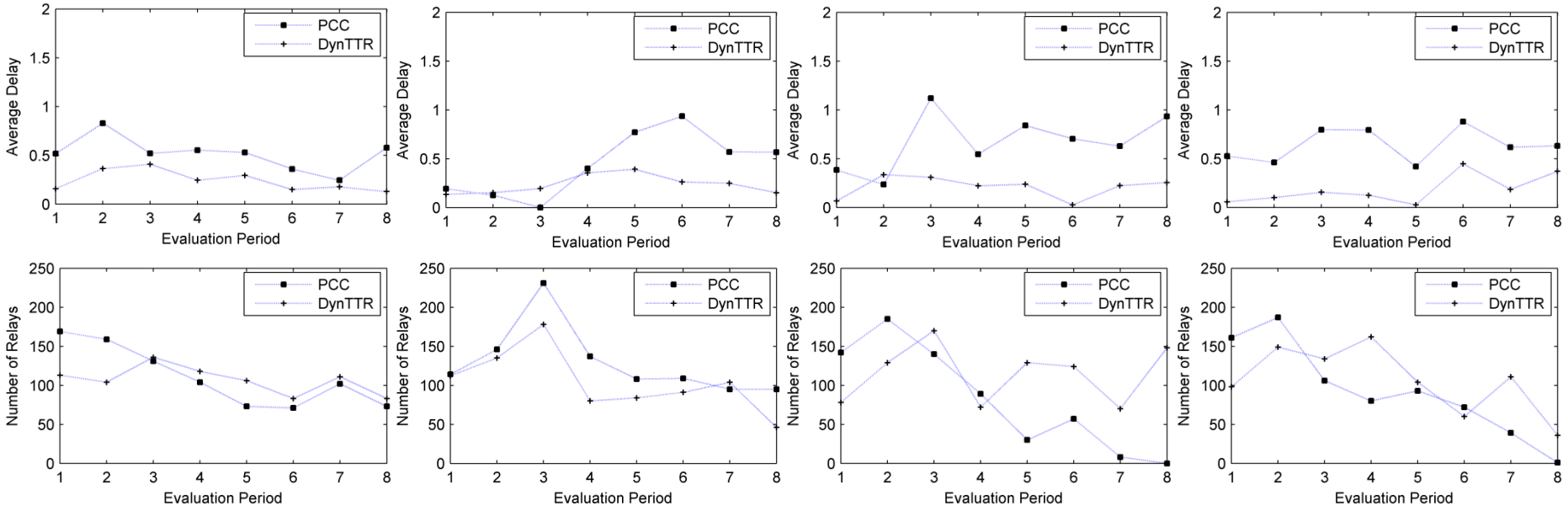
Performance results

- As *network size* changes
  - 40, 80, 120, 160
- PCC becomes better in relay counts



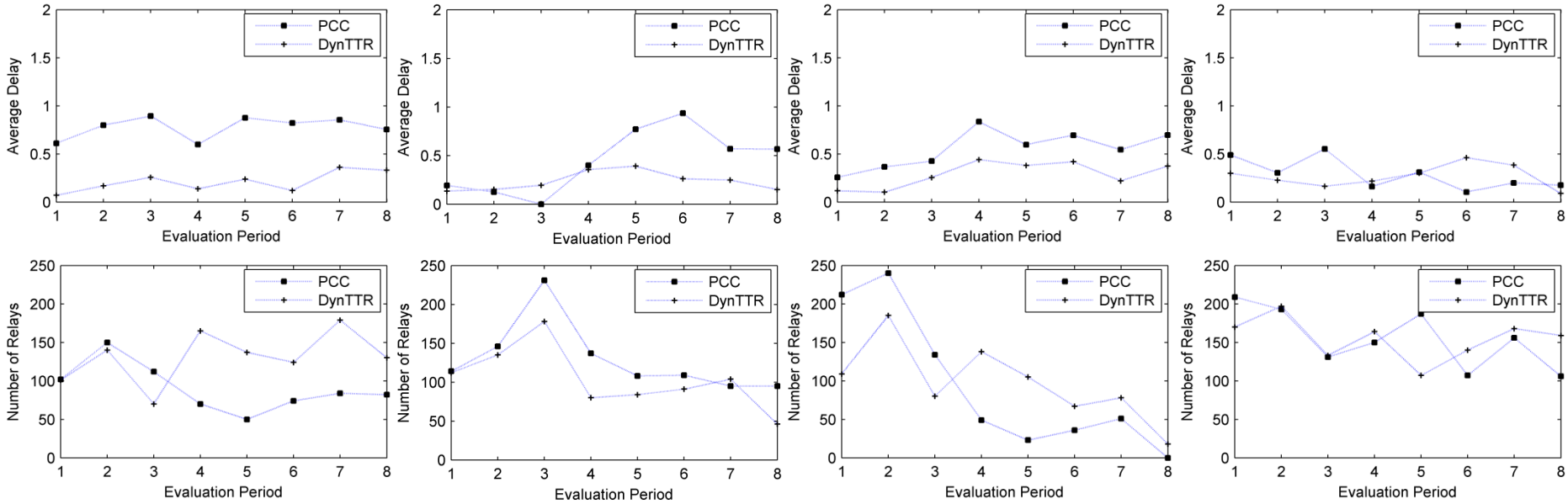
Performance results

- As *node velocity* changes
  - 1, 1.5, 2, 2.5 (m/s)
- PCC becomes better in relay counts
  - Similar performance in low speed scenarios



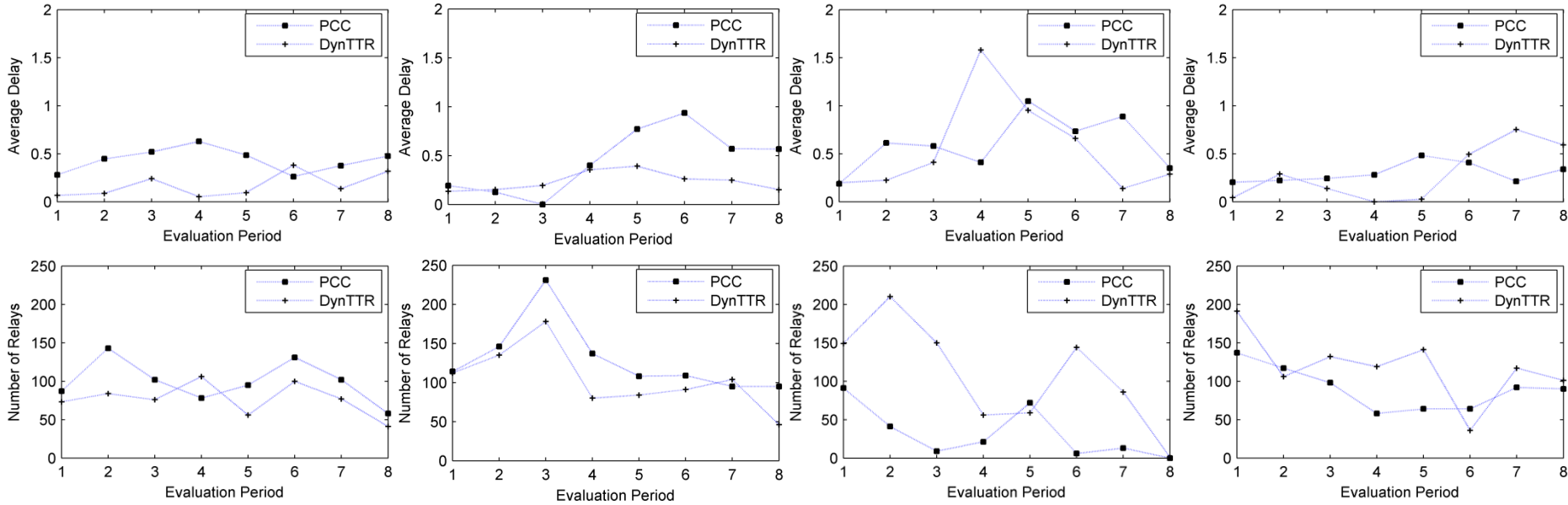
Performance results

- As *query ratio* changes
  - 60%, 90%, 120%, 150%
- PCC becomes better in relay counts
  - PCC is better for high query ratio



Performance results

- As *average update interval* changes
  - 5, 10, 15, 20 (s)
- PCC is better in relay counts except for frequent-update scenarios



Flexible Combination of Push and Pull (FCPP)

Yu Huang, Jiannong Cao, Zhijun Wang, Beihong Jin, Yulin Feng, “Flexible Cache Consistency Maintenance over Wireless Ad Hoc Networks”

IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol. 21, No. 8, August 2010, pp. 1150-1161

(5th Annual IEEE International Conference on Pervasive Computing and Communications (Percom 2007), March 19-23, 2007. New York, USA)

Probabilistic Delta Consistency (PDC)

- A general continuous consistency model
- User-specified (δ, p)

  $\forall t, \forall j, Pr\ (\exists \tau, 0 \leqq \tau \leqq \delta, \text{s.t. } S^{t-\tau} = C^t_j\ ) \geqq p;$

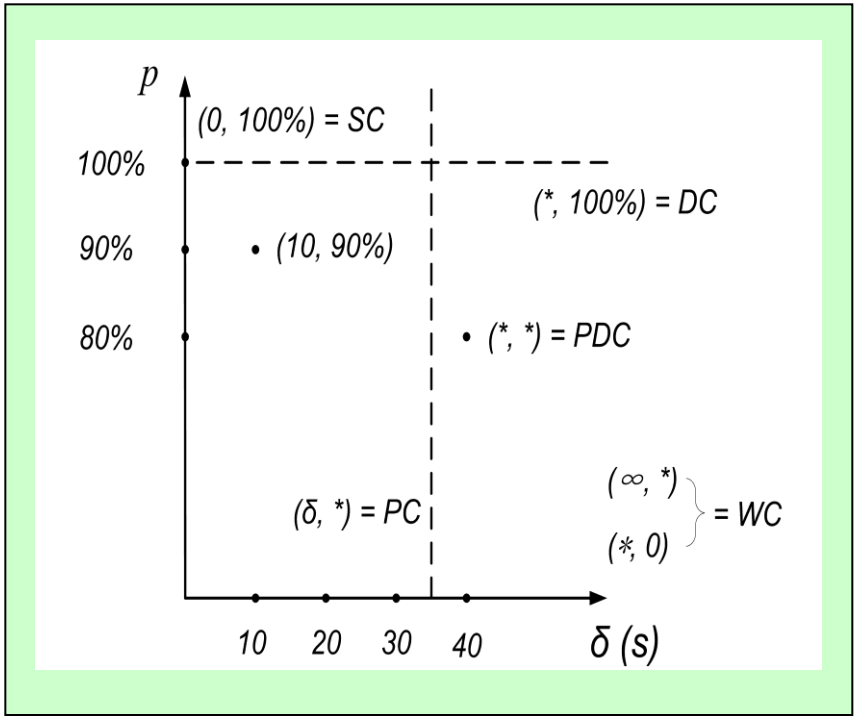
- Combining DC and PC: from

  Consistency Spectrum

  to

  Consistency Plane

- Tuning δ and p



Overview of FCPP

- **Cache copies associated with *timeout l***
  - *l* is granted by the data source as the permission period for caching nodes to directly serve cache queries
- **Upon data update:**
  - *INV* and *ACK* messages
  - Update if
    - all *ACK*s received, or
    - timeout of all non-responding caching node expires, or
    - a maximum tolerable update delay ***D*** is reached.

FCPP Algorithms

Algorithm 1: FCPP on a caching node

Upon receiving a query

(1) IF($l > 0$) serve the query with the cache copy;

(2) ELSE {* *l has decreased to zero* *}

(2.1) Send a RENEW message to renew the timeout from the data source node and update the cache copy if necessary;

(2.2) After *l* is renewed, serve the query with the updated cache copy;

FCPP Algorithms

Algorithm 2: FCPP on data source node

When the source data needs to be updated

(1) Send an INV message to each caching node with positive *l* ;

(2) IF (receive INV_ACK for all INV messages OR has waited for *D* seconds)

(2.1) Update the source data;

Upon receiving a RENEW message from a caching node

(3) Grant timeout with duration *l* and send the source data to the caching node;

FCPP properties

- **Flexibility and generality of FCPP: adjusting *l* , *D***
  - $l = 0$
    - Pull each Read
  - $l = +\infty$
    - Push with ACK
  - $l \leqq D$
    - Strong consistency, covering the *Lease* protocol
  - $l \leqq D + \delta$
    - Delta consistency
  - $l > D + \delta$
    - Probabilistic Delta consistency

FCPP properties

- **Decide the optimal value of *l***
  - satisfying the given consistency requirement PDC (δ, p), while
  - imposing minimum cost (traffic overhead and query delay)
- **Analytical model for deriving**
  - relationship between *l* and (δ, p)
  - relationship between *l* and traffic overhead *C*

Analytical model

Table 1. Notations used in the analytical model

(δ, p)	user-specified consistency requirement
D	maximum time the data source can wait before updating the source data
l	timeout duration in FCPP
N	number of caching nodes
N_k	the k^{th} caching node, k = 1, 2, …, N
$\overline{h}$	average path length between the data source and the caching nodes
w	average frequency of data update
r	average query frequency of a cache copy
t_u	time instant of one source data update

Analytical model

- **Deriving the relationship between *l* and (δ, p)**
  - Number of Stale Cache Hits on a Single Caching Node

$$S = \frac{1}{l}\int_{\delta+D}^{l}(x-\delta-D)r\,dx = \frac{r}{2l}(l-\delta-D)^2$$

  - Expected number of Stale Cache Hits on All Caching Nodes

$$ES \le \sum\nolimits_{1\le k\le N} p_k I(k) S \le \sum\nolimits_{1\le k\le N} p_k I(k)\frac{r}{2l}(l-\delta-D)^2$$

  - Satisfying PDC(δ, p)

$$p \le 1-\frac{w\cdot ES}{r\cdot N}$$

$$l \le (D+\delta)+\frac{(1-p)N}{w\sum\limits_{1\le k\le N} p_k I(k)}+\sqrt{\frac{(1-p)^2N^2}{(w\sum\limits_{1\le k\le N} p_k I(k))^2}+\frac{2(D+\delta)(1-p)N}{w\sum\limits_{1\le k\le N} p_k I(k)}}$$

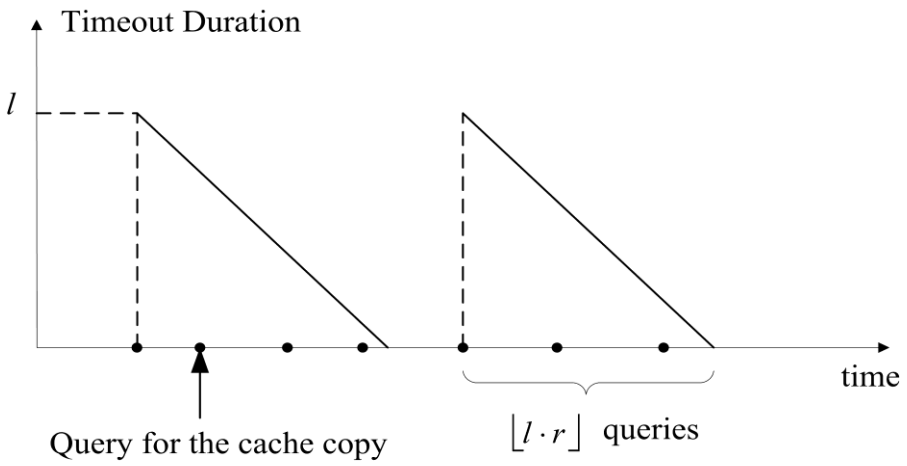
Analytical model

- **Deriving the relationship between *l* and traffic overhead *C***
  - Traffic for Timeout Renewal

$$C_1 = \frac{r}{\lfloor l \cdot r \rfloor + 1} 2 \cdot \bar{h} \cdot N$$

  - Traffic for INV&ACK

$$C_2 = w \frac{\lfloor l \cdot r \rfloor}{\lfloor l \cdot r \rfloor + 1} 2 \cdot \bar{h} \cdot N$$



- **d(*C1*+*C2*) / d*l* < 0**
  - Traffic overhead decreases as *l* increases

Analytical model

- **Calculating the optimal value of timeout duration *l***
  - satisfying User-specified Consistency Requirements
    - *l* should be bounded
  - minimizing Traffic Overhead
    - Increase *l* as much as possible

$$l = (D+\delta) + \frac{(1-p)N}{w\sum_{1\le k\le N} p_k I(k)} + \sqrt{\frac{(1-p)^2 N^2}{(w\sum_{1\le k\le N} p_k I(k))^2} + \frac{2(D+\delta)(1-p)N}{w\sum_{1\le k\le N} p_k I(k)}}$$

Experimental evaluation

- **Evaluate FCPP under different conditions - varying important parameters:**
  - δ, p
  - Average data update interval
  - Number of caching nodes
- **Compare FCPP with DynTTR (*Pull with Dynamic TTR*)**
  - Updating TTR based on a Simple Linear Model
  - Configured to achieve specified consistency requirements
- **Performance metrics**
  - Consistency Requirement: PDC(δ, p)
  - Traffic overhead: number of hops
  - Query delay

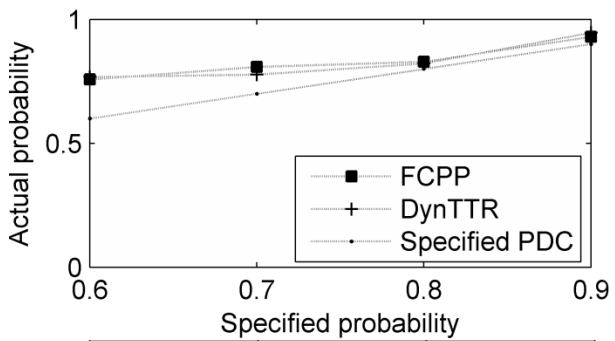
Experimental evaluation

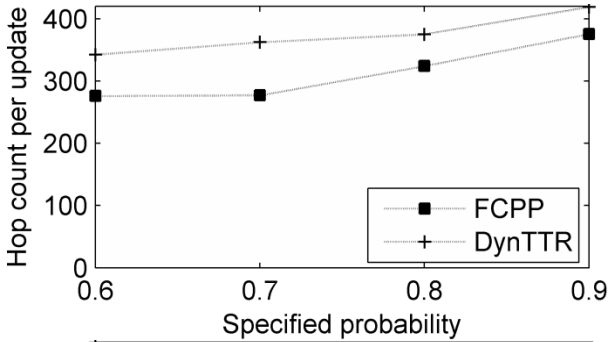
Table 2. Experimental configurations

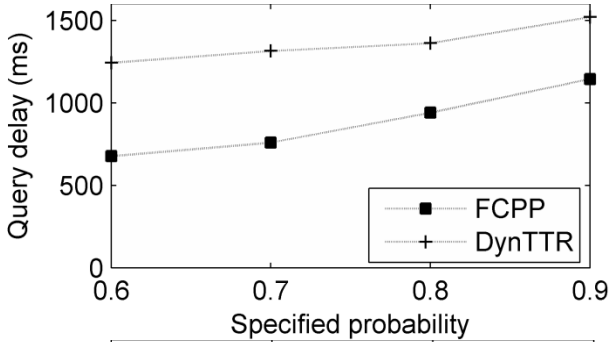
Network area	200×200 m^2
Size of network	80
Transmission range	15
Mobility model	*Random way point* [18]
Average speed	0.5 m/s
Patter of date updates and queries	*Poisson process*
Maximum portion of crashed nodes	10%
Probability of message loss per hop	5%
Maximum delay before data update	2 s
Average interval between queries	5 s

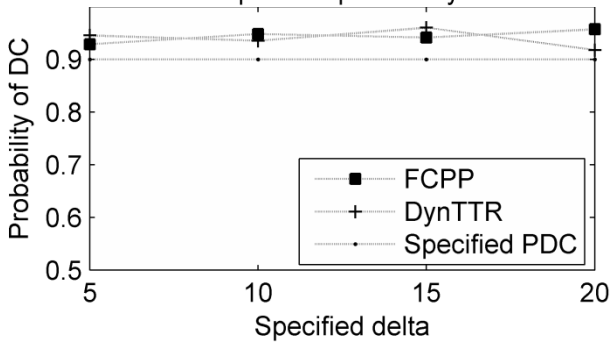
Performance results

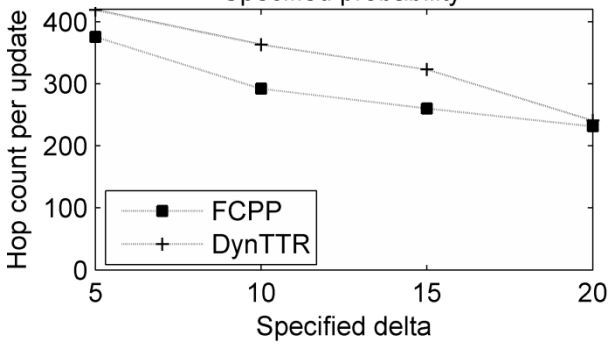
- Tuning *Consistency Requirements PDC(δ, p)*
  - FCPP is better with smaller p, smaller δ

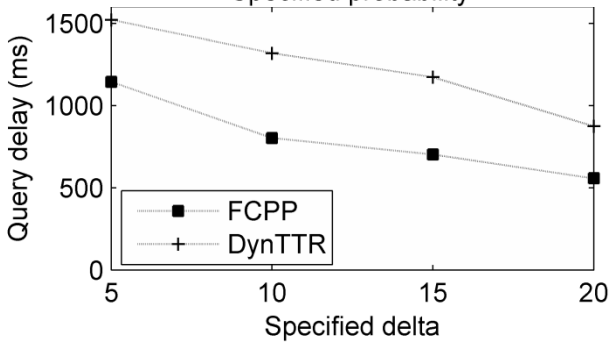






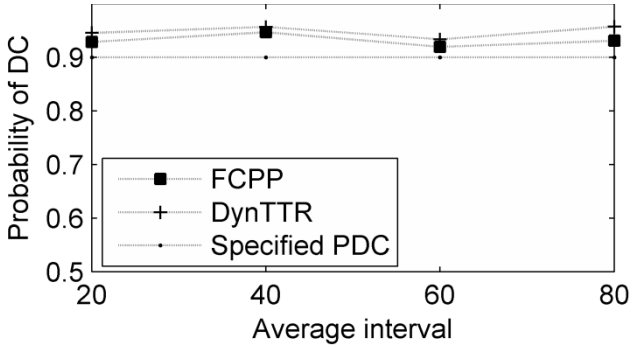


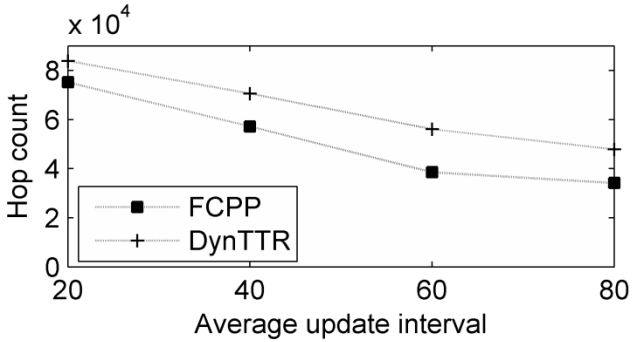


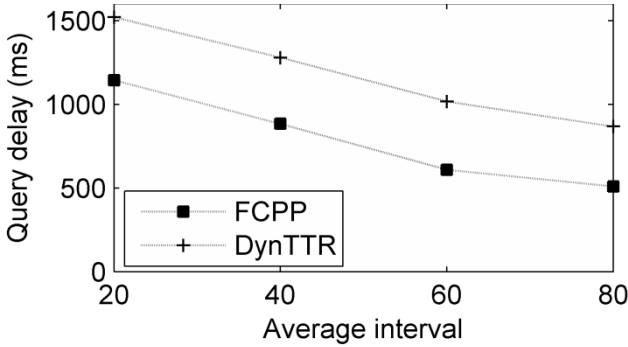


Performance results

- Tuning the *Average Update Interval*
  - FCPP is better with less frequent data updates

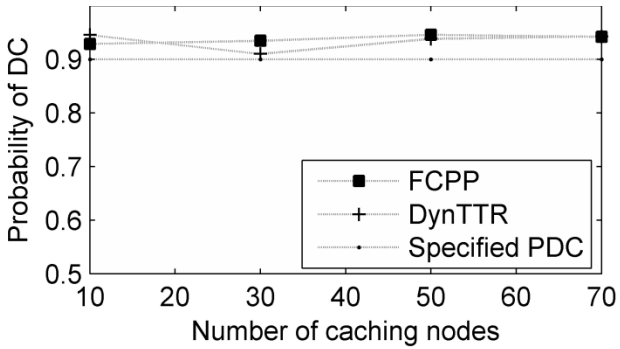


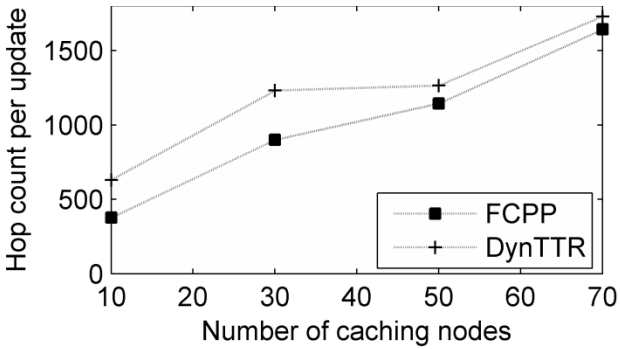


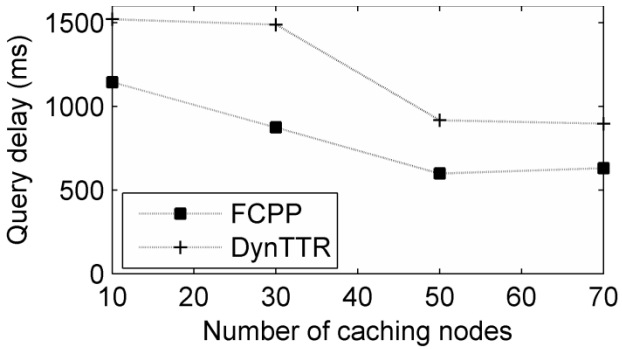


Performance results

- Tuning the *Number of Caching Nodes*
  - FCPP is better with an appropriate number of caching nodes







Summary

- Cooperative caching: a set of interesting issues
  - How to place cache in mobile nodes (cooperatively)?
    - Which node should cache which data item for which other nodes?
    - What to cache (data or path)?
  - How to find a cache node (cooperatively)?
    - Not only by user but also a cooperating cache node
  - How to replace cached data (cooperatively)?
    - Not only based on self's interest but also other caching nodes.
  - How to maintain cache consistency (cooperatively)?

Further reading

Cache placement

Xiaopeng Fan, Jiannong Cao, Weigang Wu, “Contention-Aware Data Caching in Wireless Multihop Ad Hoc Networks”, to appear in *Journal of Parallel and Distributed Computing*.

Xiaopeng Fam, Jiannong Cao, “Altruistic or Selfish: Divide-and-Rule Cooperative Caching in IMANETs”, *submitted for publication*