Data Discovery and Dissemination with DIP

Authors: Kaisen Lin, Philip Levis

Speaker: Giannakakis Spyridon

10-937-548
ETH Zurich, D-INFK

March 22, 2011

Problem

How to efficiently distribute binaries or data to remote nodes.

Problem

How to efficiently distribute binaries or data to remote nodes.

Why?

- Adjust configuration parameters.
- Reprogram nodes for extra functionality.
- Apply patches.

Limitations of wireless nodes: energy, processing power etc.

Trickle

The algorithm behind most protocols.

Operation

- Assigns keys and version numbers to data items.
- Broadcasts periodically summary of the node's data.

Trickle

The algorithm behind most protocols.

Operation

- Assigns keys and version numbers to data items.
- Broadcasts periodically summary of the node's data.
  - If neighbours agree on version number, increase the broadcast interval.
  - Else, broadcast rapidly.

Extra functionality

- Suppress message.
- Nodes can update each other.

Trickle

The algorithm behind most protocols.

Operation

- Assigns keys and version numbers to data items.
- Broadcasts periodically summary of the node's data.
  - If neighbours agree on version number, increase the broadcast interval.
  - Else, broadcast rapidly.

Extra functionality

- Suppress message.
- Nodes can update each other.

Big intervals when data consistent, rapid advertisement in case of differences.

Limitations

- Scales to many nodes but not for many data items per node.
- Detection of new data is problematic.
- Transmission costs scale linearly $O(T)$. [1]

Result:

Broadcast intervals remain big, late detection of asymmetries in the network.

[1] T number of data items

Performance metrics

- Latency.
- Maintenance cost.
- Identify cost.

Related to interval used.

Traditional Trickle: dynamic adjustment of interval propagates fast updates...

Performance metrics

- Latency.
- Maintenance cost.
- Identify cost.

Related to interval used.

Traditional Trickle: dynamic adjustment of interval propagates fast updates...
but does not help with detection itself!

Approaches using Trickle

1. Parallel Scan

   Separate Trickle for each data item. Fixed interval.

2. Serial Scan

   One Trickle (transmission) containing summary over a selection of data items.

	Detection		Identify
	Latency	Cost	
Parallel Scan	$O(1)$	$O(T)$	$O(1)$
Serial Scan	$O(T)$	$O(1)$	$O(1)$

[2]

[2] adapted from [DIP, 2008]

Cost/latency Tradeoff

Cost/latency product scales with O(T). [3]
Administrators have to choose between speed and efficiency.

Parallel scan detects faster but costs more,

serial scan is cheaper but takes longer.

[3] T data items

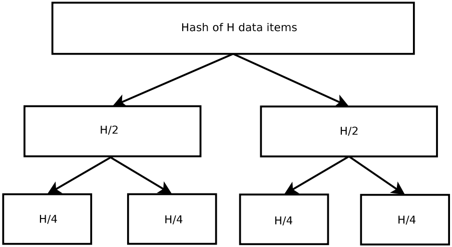
Search Protocol

Similar to a binary search.

Main idea:

Send a hash of version numbers across H data items.

When a node hears a different hash, it sends hashes of sub-range within that hash.



Performance

Detection

When stable, top-level hash is sent $\Rightarrow$ detects all differences.

Identification

Need to traverse the whole tree.

Protocol	**Latency**	**Cost**	**Identify**
Parallel Scan	$O(1)$	$O(T)$	$O(1)$
Serial Scan	$O(T)$	$O(1)$	$O(1)$
Search	$O(1)$	$O(1)$	$O(log(T))$ [4]

[4] [DIP, 2008]

Performance

Detection

When stable, top-level hash is sent $\Rightarrow$ detects all differences.

Identification

Need to traverse the whole tree.

Protocol	Latency	Cost	Identify
Parallel Scan	$O(1)$	$O(T)$	$O(1)$
Serial Scan	$O(T)$	$O(1)$	$O(1)$
Search	$O(1)$	$O(1)$	$O(log(T))$ [4]

Problem

Identifying multiple new items simultaneously.

[4] [DIP, 2008]

DISSEMINATION PROTOCOL

Hybrid protocol

- Starts with *search* to detect a difference.
- When hashes become small enough (threshold), drop to *scan* to identify the item.

[5] [DIP, 2008]

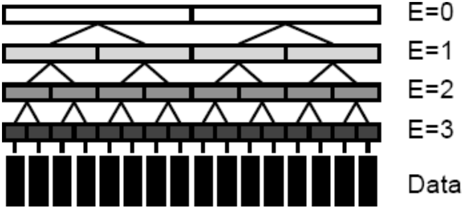
DISSEMINATION PROTOCOL

Hybrid protocol

- Starts with *search* to detect a difference.
- When hashes become small enough (threshold), drop to *scan* to identify the item.

Goal:

Detect and identify which nodes need which updates by increasing Estimate values.



[5]

[5] [DIP, 2008]

Data stored

For each data item:

- Unique key
- Version Number
- Index (memory)
- Estimates: estimate of E means that DIP detected a difference in level E.

Estimates

E = 0	constant data
$E_D = log(T)$	certainty of **d**ifference
E_O	neighbour contains **o**lder information
E_N	neighbour contains **n**ewer information

As we traverse down the tree $\Rightarrow$ E grows.

Data stored

For each data item:

- Unique key
- Version Number
- Index (memory)
- Estimates: estimate of E means that DIP detected a difference in level E.

Estimates

$E = 0$	constant data
$E_D = log(T)$	certainty of **d**ifference
E_O	neighbour contains **o**lder information
E_N	neighbour contains **n**ewer information

As we traverse down the tree $\Rightarrow$ E grows.

Overhead: Trickle uses 4 bytes, DIP extra 1 byte.

Messages

1. Data

Data Message

Key	Version Number	Data payload

Sent after detecting *and* identifying the difference, updates the node.

2. Vector

Vector Message

Key	Version Number
Key	Version Number
Key	Version Number

...

Provides detection of differences,
can identify the different item, can't update the node.
Scan part of DIP.

3. Summary:
   Summary elements (branching factor), salt value

Salt s			
Begin_1	End_1	BloomFilter_1	SummaryHash_1
...			
Begin_n	End_n	BloomFilter_n	SummaryHash_n

[6]

Provides detection with the hash (*search part of DIP*).
Salt avoids hash collisions.
Bloom filter helps to identify with probability.

[6] [DIP, 2008]

Bloom filters

"Swiss army knife"

Usage

Test membership of element e.

- Initialization: Set k bits $H(e) = 1$. [7]
- Query: Check if $H(e) = 1$
  - If no, e is not in the set
  - If yes, *might* be.

Purpose

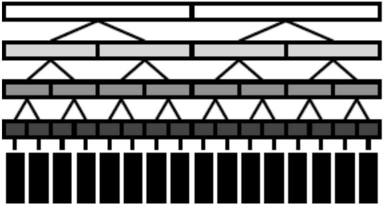
Bypass thorough traversing of tree, sets directly corresponding data items to E_D.

[7] In DIP k=1

Upon receiving

Summary message

- Matching hash: Data is the same, Estimate $= 0$.



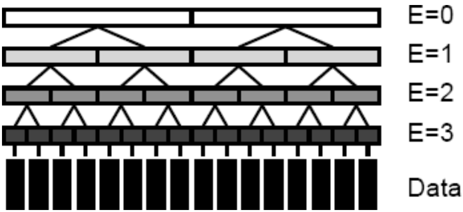
[8]

[8] [DIP, 2008]

Upon receiving

Summary message

- Matching hash: Data is the same, Estimate $= 0$.
- Differing hash: E $=$ max(currentEstimate, $log(T) - log(H)$)



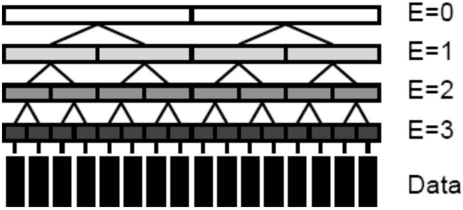
[8]

[8] [DIP, 2008]

Upon receiving

Summary message

- Matching hash: Data is the same, Estimate $= 0$.
- Differing hash: E $=$ max(currentEstimate, $log(T) - log(H)$)
- Differing hash and bloom filter: Difference detected, E_D .



[8]

[8] [DIP, 2008]

Receiving Data or Vector messages

- Same version number: Data is the same, Estimate $= 0$
- Older version: Estimate $= E_O$ unless E_N

Receiving Data or Vector messages

- Same version number: Data is the same, Estimate $= 0$
- Older version: Estimate $= E_O$ unless E_N

Vector with a newer version	E_N
Data with newer version	install update, set E_O.

Transmitting

Decrease Estimate after transmission of a message.

Transmission

Packets with highest Estimate first.
A packet represents a single level in the tree (estimate value).

Algorithm

- if Estimate is E_O, send *data* message[9].
- else:
  - if "cost of using summaries"[10] > "cost of using vectors", send summary
  - else send vectors.

[9] forwarding has higher priority
[10] $(E_D - E)$: levels until the end of the hash tree

Transmission

Packets with highest Estimate first.
A packet represents a single level in the tree (estimate value).

Algorithm

- if Estimate is E_O, send *data* message[9].
- else:
  - if "cost of using summaries"[10] > "cost of using vectors", send summary
  - else send vectors.

Basically, the scan threshold
if Estimate equals E_D or E_N, transmit a vector.

---

[9] forwarding has higher priority
[10] $(E_D - E)$: levels until the end of the hash tree

Example[11]

Node A

Sends *summary*,

each element of H=8 ↘

Node B

[11] Assumptions: Initially both in steady state, 16 Data items, without bloom filters, vectors are sent when E_D (vectors contain one tuple), summaries contain 2 elements (branching factor of 2).

Example[11]

Node A

Node B

Sends *summary*,
each element of H=8 ↘

Sets E=1,
↙ sends *summary*, elements H=4

[11] Assumptions: Initially both in steady state, 16 Data items, without bloom filters, vectors are sent when E_D (vectors contain one tuple), summaries contain 2 elements (branching factor of 2).

Example[11]

Node A

Node B

Sends *summary*,
each element of H=8 ↘

Sets E=1,
↙ sends *summary*, elements H=4

Sets E=2,
sends *summary* of H=2 ↘

[11] Assumptions: Initially both in steady state, 16 Data items, without bloom filters, vectors are sent when E_D (vectors contain one tuple), summaries contain 2 elements (branching factor of 2).

Example[11]

Node A

Node B

Sends *summary*,
each element of H=8 ↘

Sets E=1,
↙ sends *summary*, elements H=4

Sets E=2,
sends *summary* of H=2 ↘

Sets E=3,
↙ sends *vector* of 1 item

[11] Assumptions: Initially both in steady state, 16 Data items, without bloom filters, vectors are sent when E_D (vectors contain one tuple), summaries contain 2 elements (branching factor of 2).

Example[11]

Node A | Node B

Sends *summary*,
each element of H=8 ↘

Sets E=1,
↙ sends *summary*, elements H=4

Sets E=2,
sends *summary* of H=2 ↘

Sets E=3,
↙ sends *vector* of 1 item

Sets E=4(E_O, N),
sends *data* →

[11] Assumptions: Initially both in steady state, 16 Data items, without bloom filters, vectors are sent when E_D (vectors contain one tuple), summaries contain 2 elements (branching factor of 2).

Overview

Breaks the latency/cost tradeoff $O(T)$.

Overview

Breaks the latency/cost tradeoff $O(T)$.

Link layer broadcasts
Trickle timer
Hierarchical suppression
Based on local information

Overview

Breaks the latency/cost tradeoff $O(T)$.

Link layer broadcasts
Trickle timer
Hierarchical suppression
Based on local information

Contribution

- Bloom filter
- Scan after search

Evaluation

Methodology

1. Improvements

   DIP stand alone protocol.

Evaluation

Methodology

1. Improvements

   DIP stand alone protocol.

2. Comparisons

   With a serial scan and a pure search protocol.

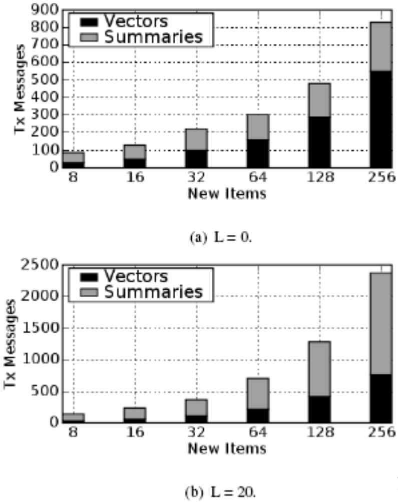
   Simulation and testbed deployment.

DIP Implementation

TinyOS 2.0, 3K code

2 elements/summary

2 tuples/vector

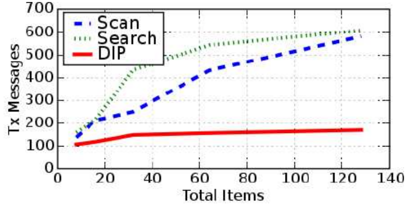


12

Figure: T = 256 total items, D = 32 nodes, L loses

[12] [DIP, 2008]

Simulations



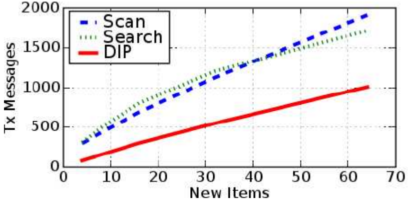


Figure: Default: N=8, T=64, D=32, L=40%[13]

[13] [DIP, 2008], N new items, T total items, D nodes, L losses

Simulation and Experiment

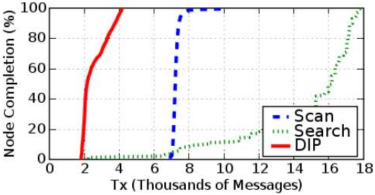


Figure: Tx messages, simulation on TOSSIM, 15 by 15 grid, T = 64

[14] [DIP, 2008]

Simulation and Experiment

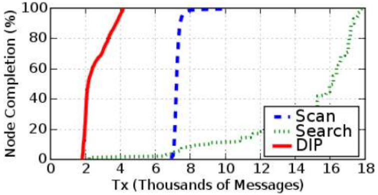
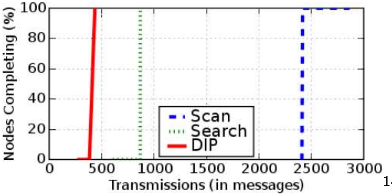


Figure: Tx messages, simulation on TOSSIM, 15 by 15 grid, T = 64



[14]

Figure: Tx costs, experiment on Mirage, T = 64

[14] [DIP, 2008]

References

Lin, Kaisen and Levis, Philip.
Data Discovery and Dissemination with DIP.
IPSN '08.

http://en.wikipedia.org/wiki/Hash_tree,Bloom_filter