USC Viterbi
School of Engineering

MIND: A Distributed Multi-dimensional Indexing System for Network Diagnosis

Xin Li, Fang Bian, Hui Zhang,
Christophe Diot, Ramesh Govindan,
Wei Hong, Gianluca Iannaccone

ENL
Embedded Networks Laboratory

intel

Multi-dimensional Range Queries for Network Monitoring

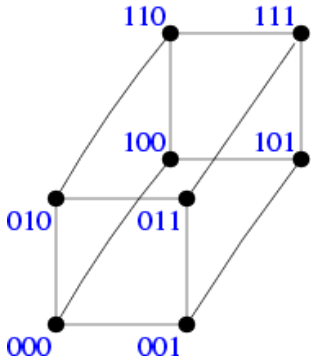
- Internet traffic is described with multi-attribute tuples
  - E.g. Flow records stored in flow-tools format
  - Various multi-dimensional spaces can be defined
    - 0601.00:40:00.628 0601.00:40:00.628 128.125/16 1300
- Multi-dimensional range queries are a natural and power ful approach to access these spaces
  - E.g. was there a flow of size greater than 1MB to customer prefix 128.125/16 in the past 5 minutes
  - More complex multi-dimensional range queries can be construct ed for detecting various network anomalies

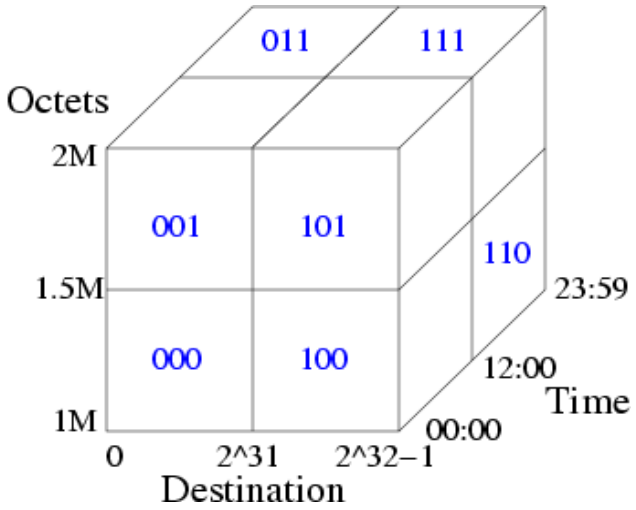
MIND Overview

- A distributed infrastructure to support multi-dimensional range queries
  - Manage data with locality-preserving hashing for effici ent query resolution
- Features
  - Allows user to easily create various indices for different network monitoring tasks
  - Handles large amount of data in near real-time
  - Maintains storage load balance with low cost
    - Histogram-based periodic storage load balancing
  - Provides flexible data availability
    - Adjustable replication factor

MIND Design

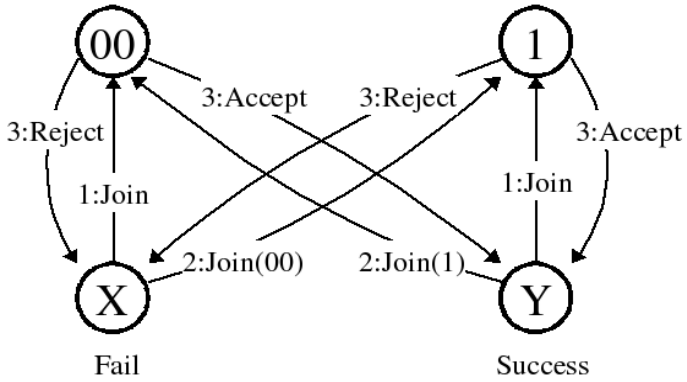
- A logical hypercube overlay
- Divide data space into blocks (hyper-rectangles)
- Encode each block based on its location in the data space
- Map each block to an overlay node that has the same code
  - Tuples in the same data block share the same code
  - Adjacent data blocks are allocated to adjacent overlay nodes





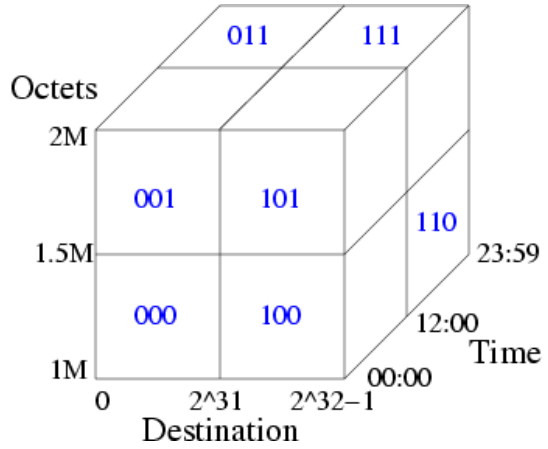
Overlay Construction

- Requirement
  - Balanced hypercube, i.e. to make all node have nearly the same code length ⇒ logarithmic neighbor list size
  - Allow distributed, concurrent node join
- Modified node join algorithm by Adler et al [STOC'03]
  - Randomized node join procedure
  - Pre-emptive serialization of concurrent node join requests



Insertion and Query Processing

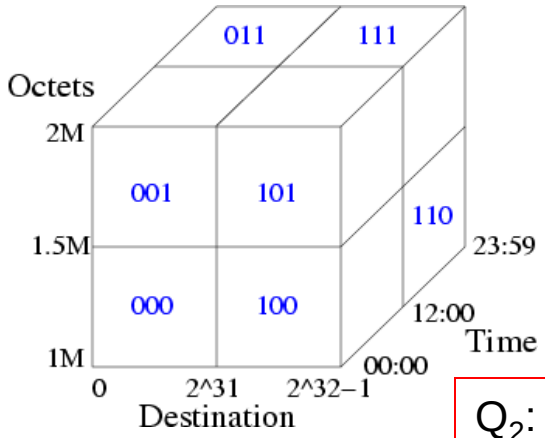
- Insertion
  - Map a tuple to a code
  - Greedy-routing the tuple to the node whose code matches the tuple's code most.

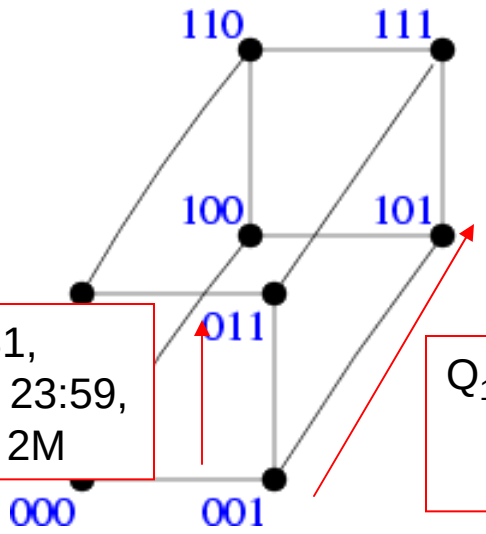


- Query processing
  - Map a query to a code (usually a prefix)
  - Greedy-routing the query to the set of nodes whose code matches the query's code most
  - When the query code is a proper prefix of the code of the node, the node *splits* the query

Query Splitting

Q: Destination = *, Time = *, 1.5M ≤ Octets < 2M





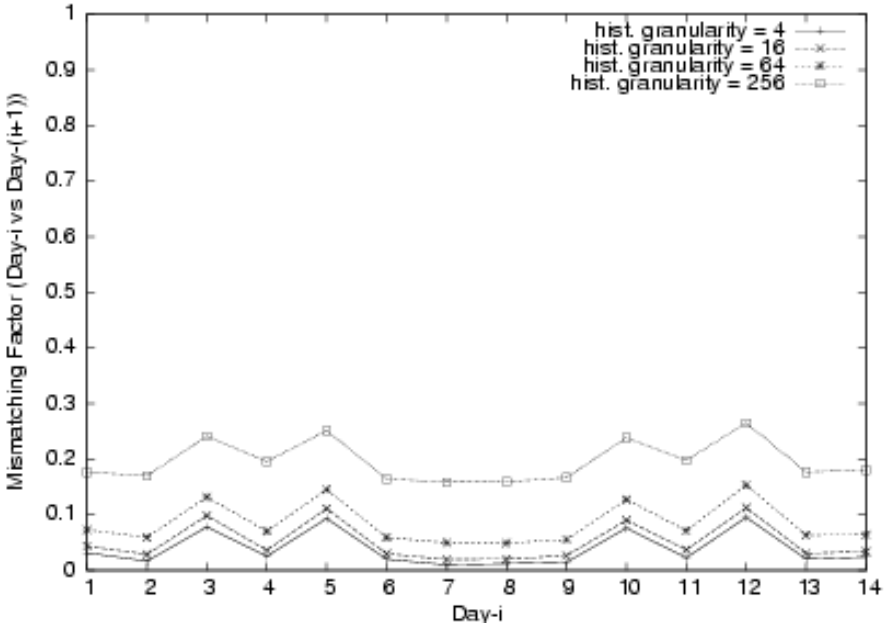
Q_2: 0 ≤ D < 2^31,
12:00 ≤ T ≤ 23:59,
1.5M ≤ O < 2M

Q_1: 2^31 ≤ D < 2^32,
T = *,
1.5 ≤ O < 2M

Q_3: 0 ≤ D < 2^31,
0 ≤ T < 12:00,
1.5M ≤ O < 2M

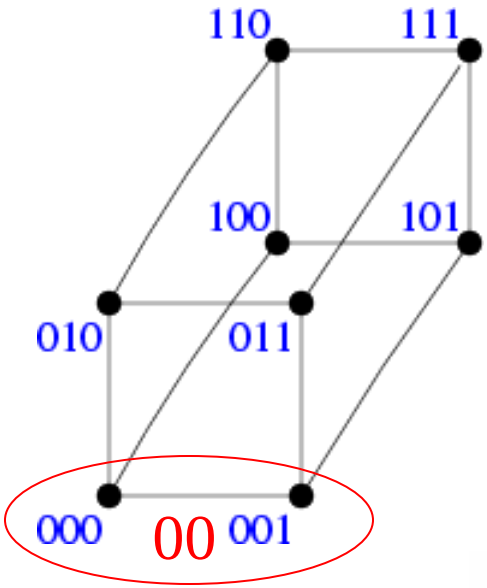
Storage Load Balancing

- Locality-preserving hash under skewed data distributions result in skewed storage load on index nodes
- Leverage two facts
  - Network traffic distribution is stationary on daily scale
  - The data space division in MIND is independent of overlay construction
- MIND approach
  - Daily rebuild indices based on data distribution histograms



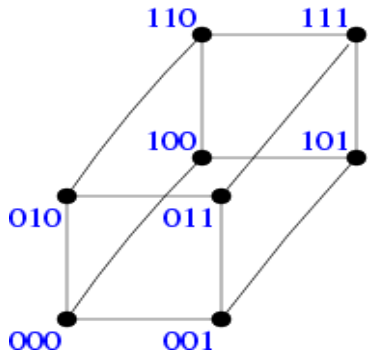
Robustness to Failure

- Link failure
  - Transient failures
  - Recovered by repeated reconnection attempts
- Node failure: Node shutdown, crash
  - Recovered by merging the data space block of the failed nodes with that of its nearest-code neighbor

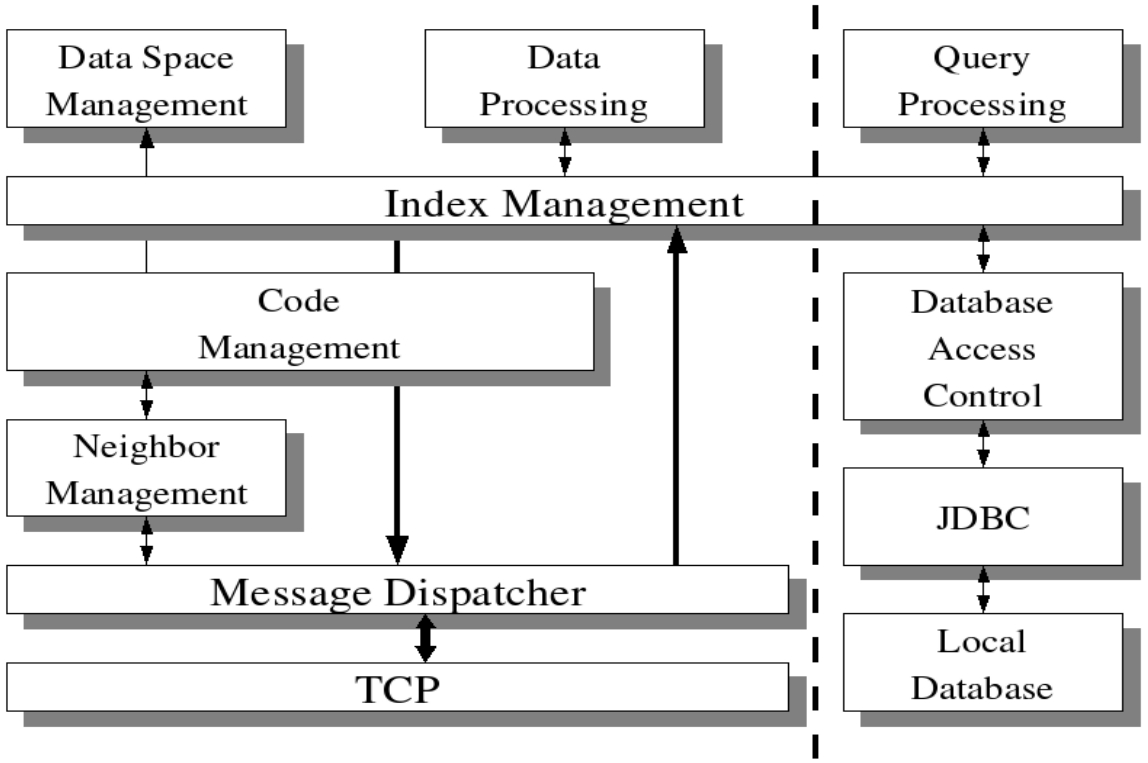


Recovery and Replication

- Routing recovery
  - Stop message forwarding
  - Probe with expanding ring to find new neighbor where greedy routing can continue
  - Rebuild hypercube connections
  - Restore message forwarding
- Adjustable data availability
  - Duplicate data in neighboring nodes
  - N nodes $\Rightarrow$ replication factor: 0 to $\log N$
  - Tradeoff between data availability and replication cost



Implementation: Node Architecture



- Event-driven model
- Java prototype on PlanetLab
- MySQL for node local storage

Performance Evaluation Outline

- Metrics
  - Insertion/query path length, latency
  - Traffic and storage load distribution
- Design workload for detecting real anomalies
  - Use traffic traces from Abilene and GÉANT backbone networks
  - Insert data for different indices in near real time
  - Queries are randomly generated every 5 minutes from all nodes
- Run MIND prototypes on PlanetLab
  - Choose nodes geographically close to the backbone routers
  - Randomly choose ~100 nodes for scalability
  - Robustness evaluation on local cluster

Experiment Workload

- Index-1: Port scan detection
  - Query: Find all the sources that generated more than *F* flows to destination *D* within time period *T*
  - Tuple: <destination, timestamp, fan-out, source, router>
- Index-2: Unusually large flow detection
  - Query: Find all flows destined for *D* that have carried at least *O* octets within time period *T*.
  - Tuple: <destination, timestamp, octets, source, router>
- Index-3: Camouflaged application detection
  - Query: Find all the flows either from source *S*, or to destination *D*, or both, that have flow size more than *X* and/or destination port *P* within time period *T*.
  - Tuple: <destination, timestamp, flow-size, source, destination-port, router>

Real-world Anomaly Detection

- Take Lakhina et al's [SIGCOMM'05] work as benchmark

Anomaly Time Window	Result Size	Actual Anomaly Size	Avg Response Time (s)
15:45	43	2 alpha flows	2.09
15:50	38	2 alpha flows	1.38
15:55	55	2 alpha flows	1.47
19:50	10	2 DoS, 1 Scan	0.81
19:55	13	2 DoS	0.79

- Easily correlate query results from different nodes, e.g. at 19:55

DoS 1	Chicago, Denver, Indianapolis, Kansas City, Los Angeles, Sunnyvale

Data Volume

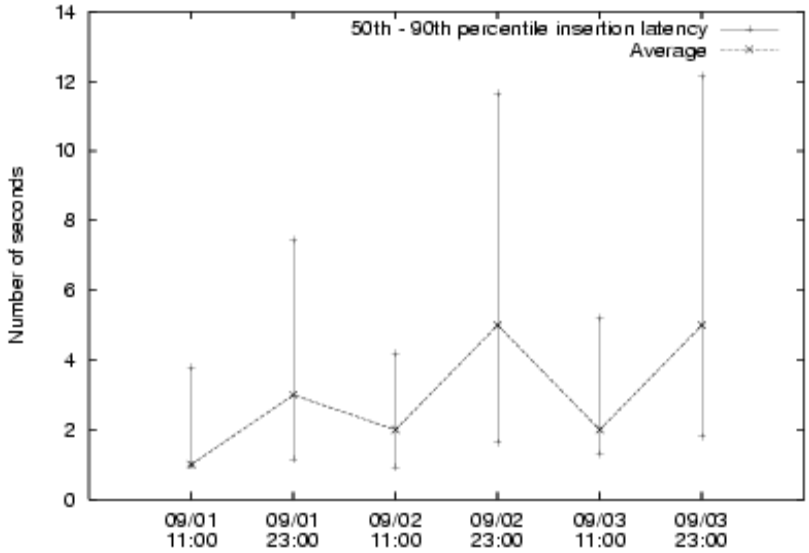
- Abilene

Days	Raw data (MB)	Index-1 (M records)	Index-2 (M records)	Index-3 (M records)	Total (M records)
2004-09-01	863	2.18	2.70	1.54	6.42
2004-09-02	877	2.66	2.66	1.50	6.82
2004-09-03	815	2.67	2.74	1.43	6.84

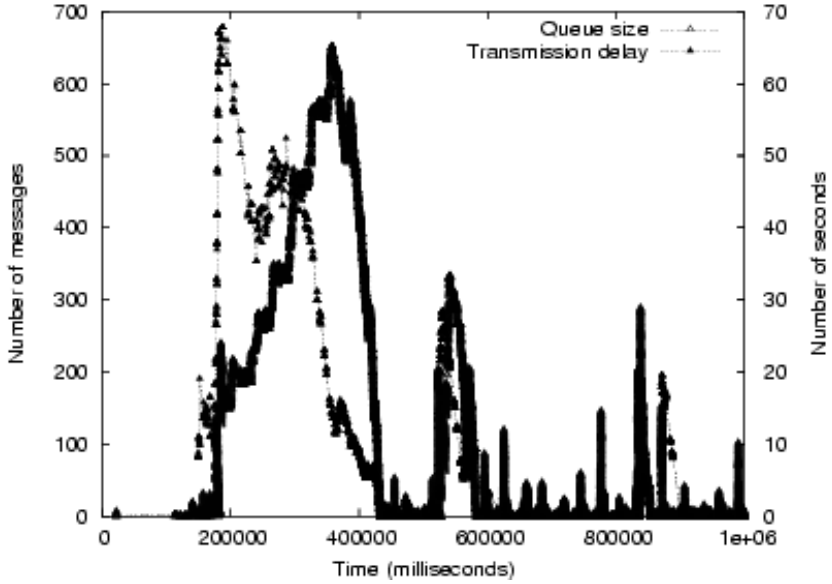
- GÉANT

Days	Raw data (MB)	Index-1 (M records)	Index-2 (M records)	Index-3 (M records)	Total (M records)
2004-09-01	394	1.21	0.74	0.44	2.39
2004-09-02	388	1.21	0.75	0.46	2.42
2004-09-03	367	1.14	0.77	0.45	2.36

Insertion latency

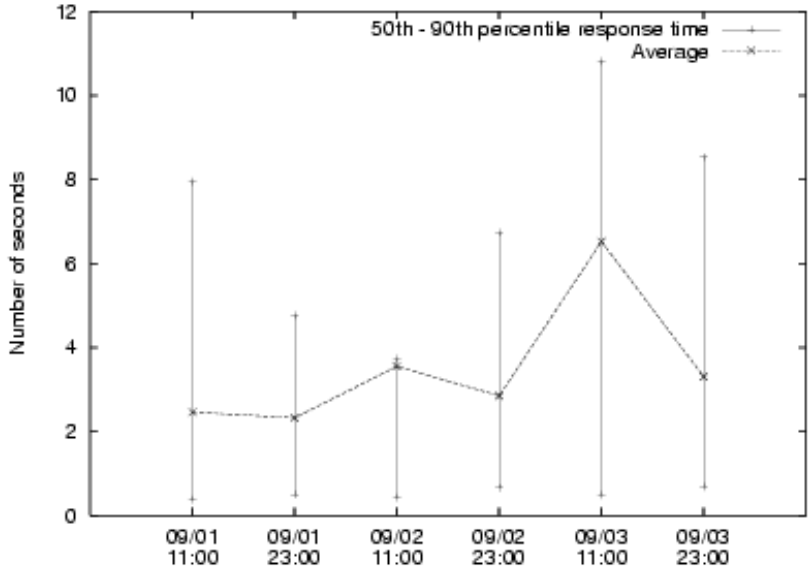


50th ~ 90th percentile insertion latency distribution for six snapshots in three days.

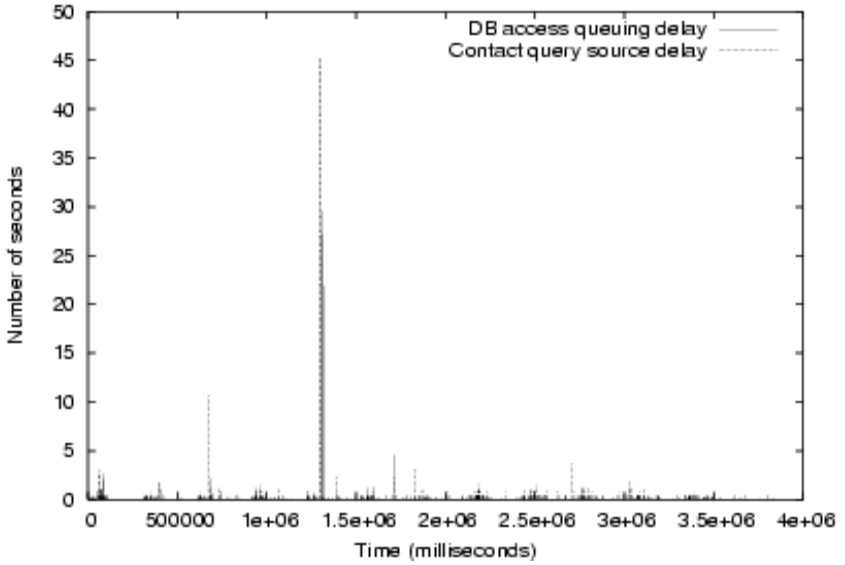


Examination of the slowest link on the path of a long-delayed insertion during a hour time. Dramatic sending queue piling up was resulted from transient increase of transmission delay.

Query Latency

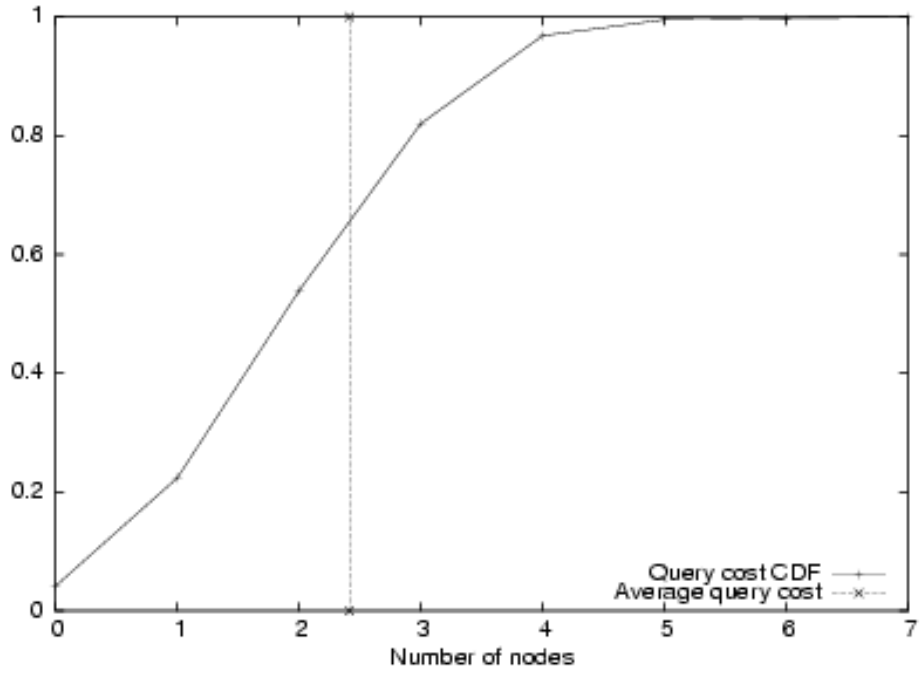


50th ~ 90th percentile query response time distribution for six time snapshots in three days.



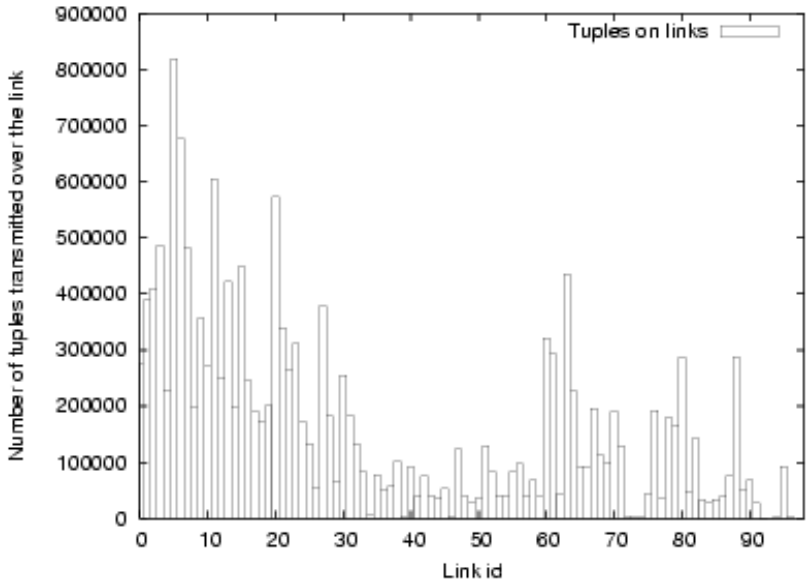
Processing delay per query at the query hot spot node. Failure in connecting the query source resulted in reconnection attempts and delayed all following query result delivery.

Query Cost

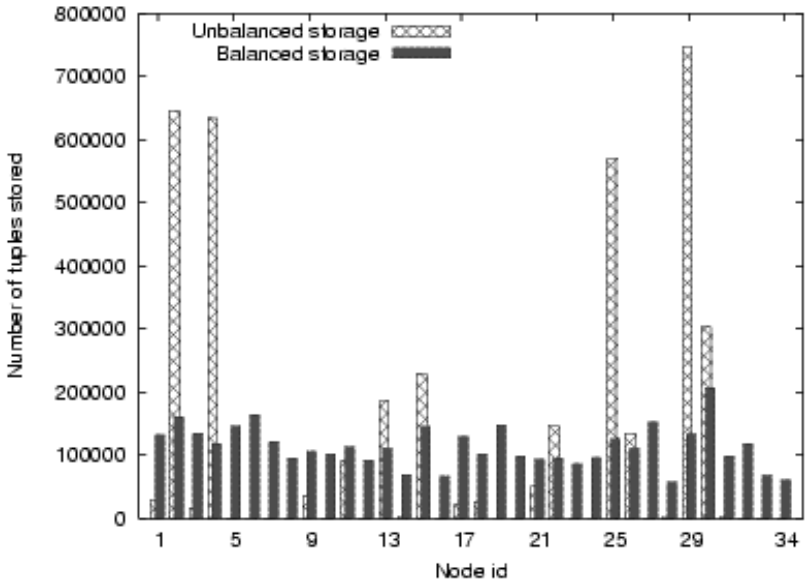


The overall query cost distribution for all three indices measured during the 11:00 to noon hour on the first day. Over 90% of the queries involved 4 overlay nodes or less.

Load Evaluation



The number of tuples traversing each MIND link for one day. The busiest link carried an order of magnitude fewer tuples than what a centralized data repository would observe, about 9 million tuples!

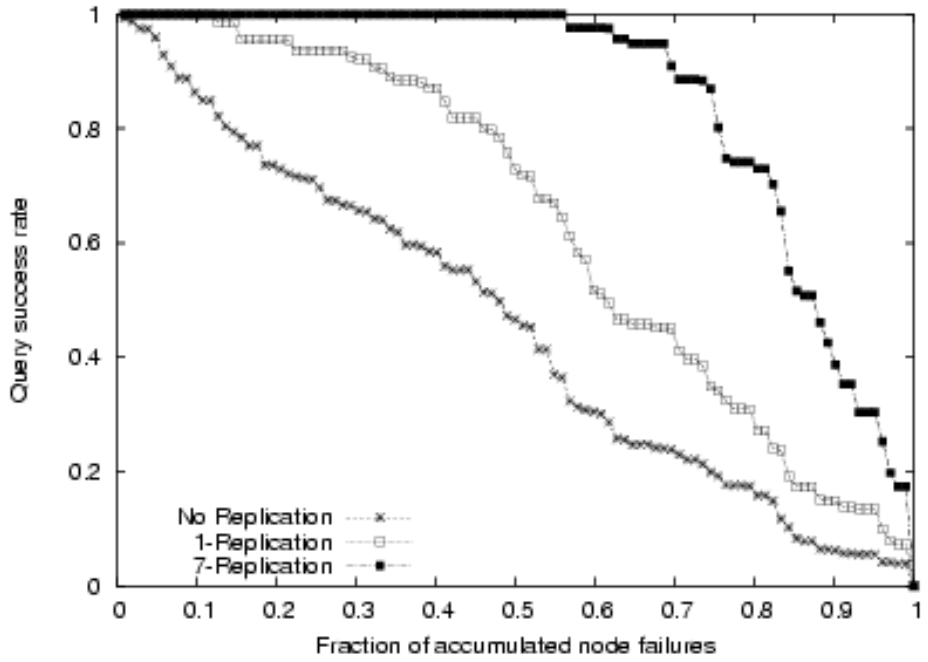


The number tuples stored at each node for Index-1 on a day. Without load balancing the majority (90%) of the tuples are stored on just 8 nodes while 17 nodes are nearly empty!

ENL Embedded Networks Laboratory

intel

Robustness Evaluation



The results of our experiments when using 0, 1, and full replication. Without replication, the fraction of successful queries decreases almost linearly with the number of node failures. With 1-replication, MIND can sustain 15% node failure without loss of data availability. With full replication, MIND can survive over 50% node failure.

Conclusion

- MIND is a distributed infrastructure to support efficient network-wide anomaly detection and is robustness to network failure
- Our evaluations suggest that it is feasible to use MIND to perform on-line near real-time anomaly detection in large networks.
- Further questions to answer
  - how to design queries for periodic monitoring?
  - how to automate the process of drilling down to find p otential anomalies?