JULIE VLAHOU

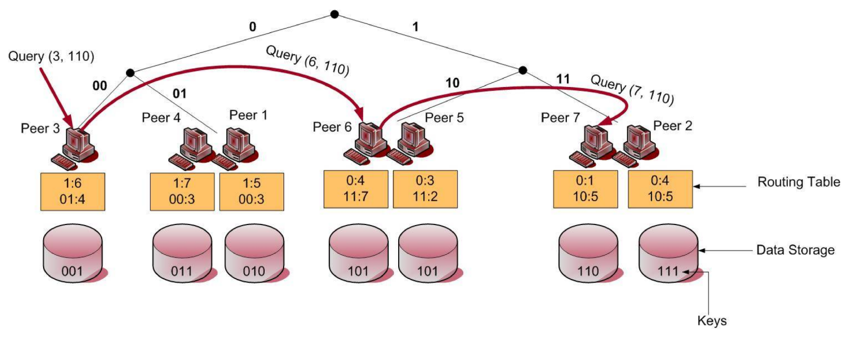
EFFICIENT REROUTING ALGORITHMS FOR TRAFFIC BALANCING IN THE P-GRID P2P PROTOCOL

EFFICIENT REROUTING ALGORITHMS FOR TRAFFIC BALANCING IN THE P-GRID P2P PROTOCOL

TECHNICAL UNIVERSITY OF CRETE

JULIE VLAHOU

SUPERVISOR : VASSILIS SAMOLADAS



Department of Electronic and Computer Engineering

October 2010

Dedicated to my parents Nikos and Glykeria,
my beloved family, friends and comrads
who have been supportive through all these years.

ABSTRACT

In this research we study load balancing methods for the P-Grid Peer to Peer protocol with focus on network traffic. Our main target is to achieve a more uniform load distribution among the network participants. We focus on alleviating the load of the most loaded peers in the network by transferring a portion of their additional load in underloaded peers. The proposed load balancing methods create alternative routing paths during the query search in order to avoid an additional load accumulation in already overloaded peers. We introduce some new metrics for a peer's load and define some new peer load states, which affect and redifine the forwarding process. The most intriguing part in this research is that the routing process now depends not only on the network's topology but also on the load of each network participant.

CONTENTS

LIST OF FIGURES

LISTINGS

1 INTRODUCTION

During the last years the scientific interest about peer to peer networks is growing rapidly due to their innovative structure and philosophy. A peer to peer network is a distributed network architecture composed of participants that make a portion of their resources directly available to other network participants, without the need for central coordination instances. Peers are both suppliers and consumers of resources, in contrast to the traditional client server model where only servers supply, and clients consume. This structure offers the users numerous benefits concerning the common usage of all the available resources such us disk storage, processing power, network bandwidth etc. In other words any peer user which joins the p2p community offers its resources in all the other peers and can simultaneously utilize their resources in return. This unique attribute of P2P networks improves the network performance in a more qualitative and versatile way in comparison with the obsolete client-server technologies.

1.1 ARCHITECTURE OF P2P NETWORKS

A P2P network structure consists of two main topologies, the Overlay network topology and the underlying Internet Topology. The underlying Internet Topology is the network structure in TCP/IP level which is used for transferring and forwarding data directly between the peers using physical addresses. The overlay network topology is a virtual network built on top of the underlying topology. Nodes in the overlay can be thought of as being connected by virtual or logical links, each of which corresponds to a path, perhaps through many physical links, in the underlying network. The overlay network consists of all the participating peers as network nodes. There are links between any two nodes that know each other: i.e. if a participating peer knows the location of another peer in the P2P network, then there is a directed edge from the former node to the latter in the overlay network. Based on the centralization degree of the overlay network we define the following categories [3]:

- Purely Decentralized Architectures: All the nodes of the peer to peer network are allowed to execute the same tasks, acting both as resource providers and consumers without using any central coordination of their activities.

- Partially Centralized Architectures: Similar to the previous architecture with the difference that there are specific nodes which have a more important role, acting as local central indexes for files shared by local peers. These nodes are called Supernodes and do not constitute single points of failure for a peer to peer network, since they are dynamically assigned and in case of failure they are automatically replaced by other network nodes. The assignment of the supernode role to specific nodes is based on criteria which differs at each peer to peer network.

- Hybrid Decentralized Architectures:In these architectures there is a central server coordinating the interaction between peers by maintaining directories of metadata, describing the shared files stored in the network nodes. All the exchange interactions between two peers are partially performed by the central server who implements the lookups and the identification of nodes with the appropriate data files. After the central coordination the two peer nodes complete the exchange interaction directly. Despite the great similarities with the client server architecture there is a significant attribute which differentiates the two architectures. In Hybrid Decentralized Architecture the data is shared between the nodes and not in the central server. Even though central coordination is necessary the central node is more alleviated than the traditional server of client server models. We should not forget that there is still a single point of failure, the central server.

Another classification for peer to peer networks which is based in the overlay network topology is structure. That is if the overlay network is created according to specific rules or is created in more arbitrary way. So in terms of the peer to peer networks structure we have the following types[3]:

- Structured: The main characteristic of structured networks is that any request of data can be efficiently routed to a peer destination that has the answer even if the requested data file is extremely rare. This means that the overlay network is strictly controlled and the data files are placed at precise and specific locations(nodes) in the network using exact mapping between content and location. Hash functions are used for the mapping, which assign data files at specific peers using data file identifiers and node addresses. A great drawback of structured peer to peer networks is that it is hard and expensive to maintain the strict overlay structure which guarantees efficient routing in a transient network, because peers are entering and leaving the network with high or even arbitrary rate. Some representative examples of structured peer to peer systems include Chord, CAN, Tapestry and P-Grid.

- Unstructured: In the Unstructured peer to peer systems the links of the overlay network are defined arbitrarily and also the placement of contents is completely irrelevant with the overlay topology. This type of networks can be easily constructed by continuous joins of new peers in the network, by splitting the data and copying the overlay links of an old node and then form their own over time. Searching mechanisms are more inexplicit here due to the flexibility of the overlay network topology. The most common searching mechanism is the flood algorithm where each query is propagated in the network until a node with the requested data is found. Quite known also is the propagation of query using random walks, where a node instead of forwarding the query at all neighbors chooses a neighbor randomly and propagates the query only at the chosen one. A major drawback is that the searching mechanisms employed in unstructured networks have obvious implications, particularly in regards to data availability, scalability and persistence. Concerning the restricted data availability many data replication mechanisms have been developed and applied with great success in almost all the Unstructured peer to peer networks. Considering the above we can easily realize that the Unstructured networks are more appropriate for accommodating highly-transient node populations. Some prevalent Unstructured networks are Napster, Gnutella, FreeHaven and Edutella.

1.2 DEFINITION OF THE PROBLEM

As we mentioned above all the nodes which affiliate the network provide their resources such us bandwidth, storage space and computing power. As nodes arrive and demand on the system increases, the total capacity of the system also increases. In contrast with a typical client server architecture where clients share only their demands with the system, but not their resources so as more clients join the system, less resources are available to serve each client. This indicates that the biggest drawback of traditional client server model turns into a great advantage for the distributed peer to peer networks. Another great asset is that in distributed systems there is no single point of failure in the system due to lack of centralized index server during search. This fact not only improves the system performance but also increases network robustness. The distributed peer to peer networks are not lacking in comparison with the antecedent common purpose systems but have a long way ahead of them for some additional and unique characteristics that they introduce. A major problem is that the network load is not uniformly distributed among the peers, this happens because the popularity of the available data in the network differs significantly. Indisputably some data are more popular than others and as a consequence the peers that own this data accept more requests, so they have greater load to manage. Under these circumstances some peers are unable to handle the additional load and subsequently crash down. A peer crash down means that the peer leaves the network and so the overlay network links among the peers need to be changed or updated, furthermore the data which were assigned to the departed peer must be distributed to other peers. All these actions have great cost and taking under consideration that the peer might leave the network immediately in some cases there is no time for the above actions to take place before the peer's departure. So the non uniform load distribution among peers is a great weakness which encumbers system's performance and is the reason why the most recent researches in the field of peer to peer networks focalize in studying load balancing methods.

1.2.1 *Our Approach*

This study focuses in overcoming the problem described above by introducing new load balancing methods for a peer to peer network. Our objective is to propose and study efficient algorithms in order to avoid heavy load in specific network participants. Specifically this research aims to:

- Identify the parameters which can lead in overloading a peer in a P2P network.
- Understand and study the impact of the overloaded peers in network's performance.
- Propose techniques which contribute in alleviating peers with popular data and also elaborate the heavy load so as to not have a dramatic impact in the network's performance.

Our theoretical study is mostly based in queuing theory which provide us with useful mathematical tools for the overloaded peers problems. Main focus was given in a peer's receiving, processing and sending mechanisms in order to evaluate and select the determinant parameters which define our load balancing algorithms. Some very important parameters of our study which contribute significantly in detecting and further avoiding saturation problems in the network includes:

- the arrival rate of the messages in the input queue of a peer

- the service rate of peers
- the portion of time at which the peer is busy
- the delay of the network
- the information for the load of the neighboring peers in the network which can be represented by a load-factor. This information was proved useful when choosing the forwarding path for a message in the network in order to avoid the additional load of the overloaded peers. The definition for the load-factor will be described in the chapter 3.

Finally we will describe how the above parameters can be exploited in order to implement efficient rerouting algorithms during query search in the network. We introduce new rerouting algorithms which take into consideration the load of the destination peer and the network topology. In our approach there is additional information in the routing tables of each peer. This information concerns the load of each corresponding neighbor which is used during message forwarding in the network so as to avoid aggravating already overloaded peers.

Load balancing is one of the wick spots of today's P2P networks so any improvement in this field is of major importance. The alleviation of overloaded peers by transferring a portion of their load in less stressed peers, can enhance network's consistency by relieving the peers with increased traffic. In the following chapters we will prove that this can be easily done without applying any additional data balancing methods in the network.

2 RELATED WORK

In this chapter we will present the most prevalent peer to peer networks protocols as well as range queries. Moreover we will talk about searching techniques of each protocol concerning management of range queries.

2.1 DISTRIBUTED HASH TABLES

Distributed hash tables more widely known as DHT's are a category of decentralized distributed systems which provide lookup service similar to hash tables. In every registration stored in a DHT a key is assigned and so we have key-value pairs. The value retrieval can be done by any participating node in the system using the corresponding key during search. The mapping maintenance from keys to values is distributed among the network participants in such a way that any node departure or failure will cause minimal disruption in the mapping mechanism. This shows the ability of DHT's to handle efficiently continuous node arrivals, departures and failures without any serious impact in the system's mapping. Furthermore is able to expend the scalability[1] of the system to extremely large number of nodes.

DHT research was originally motivated, in part, by peer-to-peer systems such as Napster, Gnutella, and Freenet, which took advantage of resources distributed across the Internet to provide a single useful application. In particular, they took advantage of increased bandwidth and hard disk capacity to provide a file sharing service.

The design of Gnutella was the first attempt to create a decentralized indexing scheme using a peer-to-peer network. Each node of this network maintains a list of neighbors which is used in order to route messages across the peers. When a query is instantiated at a peer, the query is forwarded to every node in the neighbor list and recursively in the subsequent nodes. Additionally, the search requests have high probability to be dropped before the whole network has been contacted, therefore the results cannot be reliable. Unfortunately, this flooding scheme is the only valid way to locate data in networks with such infrastructure.

Napster had a central index server: each node, upon joining, would send a list of locally held files to the server, which would

1 scalability is a desirable property of a system, a network, or a process, which indicates its ability to either handle growing amounts of work in a graceful manner or to be readily enlarged

perform searches and refer the querier to the nodes that held the results. This central component left the system vulnerable to attacks and lawsuits.

Freenet was also fully distributed, but employed a heuristic key-based routing in which each file was associated with a key, and files with similar keys tended to cluster on a similar set of nodes. Queries were likely to be routed through the network to such a cluster without needing to visit many peers. However, Freenet did not guarantee that data would be found.

Distributed hash tables use a more structured key based routing in order to attain both the decentralization of Gnutella and Freenet, and the effciency and guaranteed results of Napster. One drawback is that, like Freenet, DHTs only directly support exact-match search, rather than keyword search, although that functionality can be layered on top of a DHT. DHTs characteristically emphasize the following properties:

- Decentralization:the nodes collectively form the system without any central coordination.
- Scalability:the system should function efficiently even with thousands or millions of nodes.
- Fault-tolerance:the system should be reliable even with nodes continuously joining, leaving, and failing.

The infrastructure of DHT's can be used except from peer to peer file sharing systems, in distributed databases, in content distribution systems, cooperative web caching[2], multicast[3], domain name services[4] and instant messaging mechanisms[5]. Noticeable distributed systems which use the DHT infrastructure are BitTor-

2 Web caching is the caching of web documents (e. g. , HTML pages, images) to reduce bandwidth usage, server load, and perceived lag. A web cache stores copies of documents passing through it; subsequent requests may be satisfied from the cache if certain conditions are met

3 Multicast addressing is a network technology for the delivery of information to a group of destinations simultaneously using the most efficient strategy to deliver the messages over each link of the network only once, creating copies only when the links to the multiple destinations split

4 The Domain Name System (DNS) is a hierarchical naming system for computers, services, or any resource connected to the Internet or a private network

5 Instant messaging (IM) is a form of real-time direct text-based communication between two or more people using personal computers or other devices, along with shared software clients

rent's distributed tracker[6], the Kad Network[7], the Storm Botnet[8] and the Coral Content Distribution Network[9].

2.1.1 *CAN*

Content Addressable Network(CAN) which introduced by Ratnasamy is a distributed Internet-scale hash table that maps filenames to their location in the network, by supporting the insertion, deletion and lookup of key-value pairs in the table [8]. The data space in CAN uses a virtual d-dimensional Cartesian coordinate space and any key is deterministically mapped onto a point of this coordinate space. The coordinate space is split into segments which are assigned to the network nodes, and each node is responsible for the pairs key-value whom the points lies in the corresponding segment. CAN nodes also maintain routing tables containing the nodes that hold segments adjacent to theirs in the coordinate space. Figure 1 shows a simple CAN topology with 5 nodes.

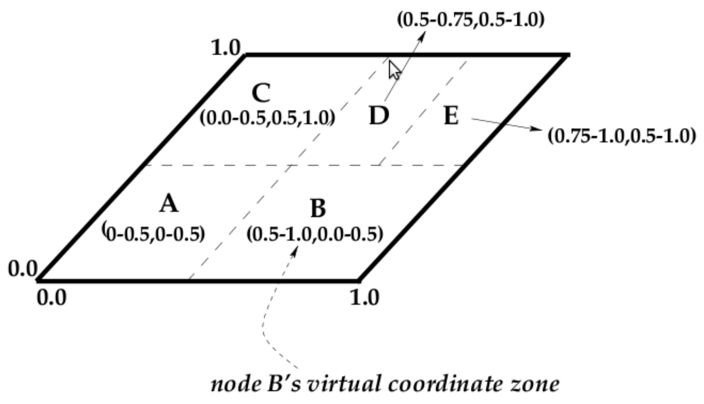


Figure 1: Simple CAN 2-dimensional overlay topology

Routing in CAN works by following the straight line path through the Cartesian space from source to destination coordinates. During node insertion, the joining node must locate a bootstrap node[10], identify a data zone (segment) that can be split

6 BitTorrent is a peer-to-peer file sharing protocol used for distributing large amounts of data

7 The Kad network is a peer-to-peer (P2P) network which implements the Kademlia P2P overlay protocol which were designed by Petar Maymounkov and David Mazieres

8 The Storm botnet or Storm worm botnet is a remotely controlled network of "zombie" computers (or "botnet") that has been linked by the Storm Worm, a Trojan horse spread through e-mail spam

9 Is a peer to peer network that uses the bandwidth of a world-wide network of web proxies and nameservers to mirror web content, http://www.coralcdn.org/

10 Bootstrap node is a node in an overlay network that provides initial configuration information to newly joining nodes so that they may successfully join the overlay network

and finally Update the routing tables of nodes neighboring the newly split zone. To handle a node departing, the CAN must identify that a node is departing, have the departing node's zone merged or taken-over by a neighboring node, and finally update the corresponding routing tables across the network. CAN is designed to be scalable, fault-tolerant and self-organizing and has a routing performance of $O(kN^{1/k})$.

2.1.2 *Chord*

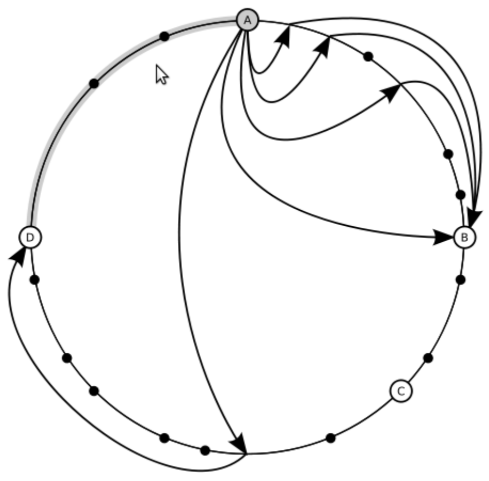


Figure 2: Typical Chord topology: the shaded region is the responsibility area of the shaded peer and the arrows indicate the entries in the finger table.

Another distributed hash table is the Chord peer to peer lookup protocol for Internet applications proposed by Stoica in 2001[13]. In Chord except from data, nodes are also identified by $m - bit$ keys. The $m - bit$ identifier keys are assigned both to files and nodes by means of a deterministic function, a variant of consistent hashing [11]a technique which is designed to let peers enter and leave the network with minimal disruption. All node identifiers are arranged in a "identifier circle" which cannot have more than $2m$ nodes. Key k is assigned to the first node whose identifier is equal to, or follows k, in the identifier space. This node is called the successor node of key k and the node which precedes key k is called predecessor. The only routing information required, is for each node to be aware of its successor node on the "identifier circle". Queries for a given key are passed around the circle via the successors' pointers until a node that contains the key is encountered. A typical Chord topology is shown in Figure 2.

11 Consistent hashing is a scheme that provides hash table functionality using the SHA-1 algorithm in a way that the addition or removal of one slot does not significantly change the mapping of keys to slots. By using consistent hashing, only K/n keys need to be remapped on average, where K is the number of keys, and n is the number of slots.

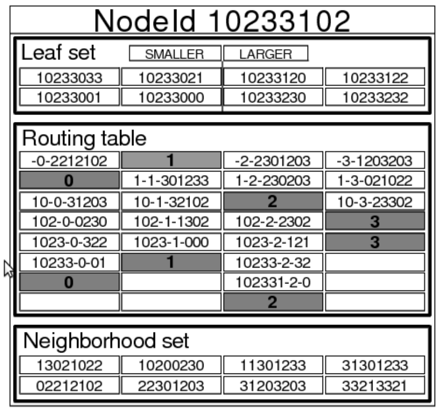
When a new node n joins the network, certain keys previously assigned to n's successor will be assigned to n. When a node n leaves the network, all its keys will be assigned to its successor. The only changes in the the network which need to be done in order to preserve load balance are the above. Also, in order to achieve logarithmic lookup performance, each peer maintains a finger table which points to the peer responsible for every $2^i - 1$ interval from this peer, with $1 <= i <= m$.

2.1.3 *Pastry*

Pastry[10] is a Distributed Hash Table quite similar to Chord. The hash table's key-space is taken to be circular, like the key-space in the Chord system, and node IDs are 128-bit unsigned integers representing position in the circular key-space. Node IDs are chosen randomly and uniformly so peers who are adjacent in node ID are geographically diverse. An attribute that sets apart Pastry is that the notion of network proximity is based on a scalar proximity metric, such us the number of IP routing hops or geographic distance. A node with a lower distance value is assumed to be more desirable. For the purpose of routing, node id's and keys are thought of as a sequence of digits with base 2^b [12]. Pastry routes messages to the node whose id is numerically closest to the given key with the following way:In each routing step, a node normally forwards the message to a node whose id shares with the key a prefix that is at least on digit longer than the prefix that the key shares with the present's node id. If no such node is known, the message is forwarded to a node whose id shares a prefix with the key as long as the current node, but it is numerically closer to the key than the present node's id. We do this recursively until the node with the searching key is encountered. To support this procedure, each node maintains a routing table, a neighborhood set and a leaf set.

The routing overlay network is formed on top of the hash table by each peer discovering and exchanging state information consisting of a list of leaf nodes, a neighborhood list, and a routing table. The routing table of a Pastry node contains one entry for each address block assigned to it. To form the address blocks, the 128-bit key is divided up into digits with each digit being b bits long, yielding a numbering b system with base 2 . This partitions the addresses into distinct levels from the viewpoint of the client, with level 0 representing a zero-digit common prefix between two addresses, level 1 a one-digit common prefix, and so on. The routing table contains the address of the closest known peer for each possible digit at each address level, except for the digit that belongs to the peer itself at that particular level. The leaf node list consists of the $L/2$ [13] closest peers by node id in each direction around the circle. In addition to the leaf nodes there is also the

12 The value of b is defined according to the corresponding Pastry network
13 L in most cases take the value 16 or 32



NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

-0-2212102	1	-2-2301203	-3-1203203
0	1-1-301233	1-2-230203	1-3-021022
10-0-31203	10-1-32102	2	10-3-23302
102-0-0230	102-1-1302	102-2-2302	3
1023-0-322	1023-1-000	1023-2-121	3
10233-0-01	1	10233-2-32	
0		102331-2-0	
		2	

Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Figure 3: State of an hypothetical Pastry node with node id 10233102, $b = 4$, $l = 8$. All numbers are in base 4.

neighborhood list. This represents the M closest peers in terms of the routing metric. Although it is not used directly in the routing algorithm, the neighborhood list is used for maintaining locality principals in the routing table. A typical example of all routing information of a Pastry node is shown in Figure 3.

2.1.4 *Tapestry*

Tapestry[14] is a distributed hash table which provides a decentralized object location, routing, and multicasting infrastructure for distributed applications. It is composed of a peer-to-peer overlay network offering efficient, scalable, self-repairing, location-aware routing to nearby resources. The topology of the network is self-organizing as nodes come and go, and network latencies vary. Each node is assigned a unique nodeID uniformly distributed in a large identifier space. Tapestry uses $SHA - 1$ hash function to produce a 160-bit identifier space represented by a 40 digit hex key. The routing and location information is distributed among network nodes; the topology's consistency is checked on the fly, and if it is lost or destroyed due to failures , it is easily rebuilt or updated. Each node maintain a neighbor map which comprises of multiple levels l. Each level l contains pointers to nodes whose id's must match with l digits with the current node. Each entry in the neighbor map corresponds to pointer to the closest node in the network whose id matches the number in the neighbor map, up to a digit position.

Routing in Tapestry has as follows. Each identifier is mapped to a live node called the root. If a node's nodeID is G then it is the root else use the routing table's nodeIDs and IP addresses to find the nodes neighbors. At each hop a message is progressively routed closer to G by incremental suffix routing. Each neighbor

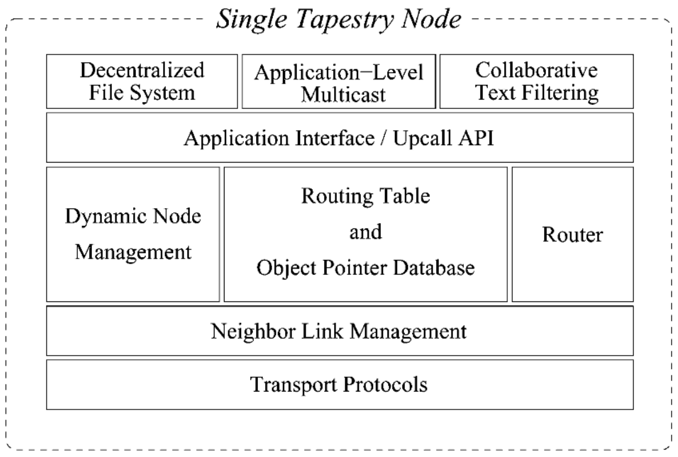

Single Tapestry Node
Decentralized File System
Application–Level Multicast
Collaborative Text Filtering
Application Interface / Upcall API
Dynamic Node Management
Routing Table and Object Pointer Database
Router
Neighbor Link Management
Transport Protocols

Figure 4: Tapestry component architecture. Messages pass up from physical network layers and down from application layers. The Router is a central conduit for communication.

map has multiple levels where each level contains links to nodes matching up to a certain digit position in the ID. Because of this, routing takes approximately $log(BN)$ hops in a network of size N and IDs of base $B(hex : B = 16)$. If an exact ID can not be found, the routing table will route to the closest matching node. For fault tolerance, nodes keep c secondary links such that the routing table has size $c * B * log(BN)$. Figure 4 shows a typical Tapestry component architecture.

2.1.5 *P-Grid*

P-Grid is the peer to peer protocol in which our work is based. It was introduced by Karl Aberer and is an efficient self-organizing infrastructure which can accommodate arbitrary key distributions (and hence support lexicographic key ordering and range queries), while providing storage load-balancing and efficient search by using randomized routing. The main characteristics of P-Grid are the following:

- Arbitrary load-distribution over the key-space by preserving good balance over the stored data.

- Efficient elaboration of range queries while simultaneously supports arbitrary load-distribution over the key-space as in real life scenario.
- Offers decentralized bootstrapping in the network and hence easy merge of multiple P-Grid networks without any deterioration in the load distribution.
- Replication mechanisms and also gossip based algorithms in order to preserve replicated data up-to-date.
- A Self-referential directory is realized using to provide peer identity persistence over multiple sessions.
- Query-adaptive caching is easy to realize on P-Grid to provide query load-balancing where peers have restricted capacity.

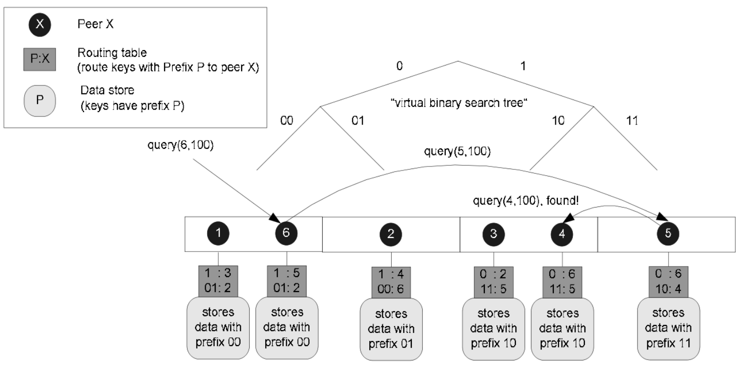

X Peer X
P:X Routing table (route keys with Prefix P to peer X)
P Data store (keys have prefix P)
0
1
"virtual binary search tree"
00
01
10
11
query(6,100)
query(5,100)
query(4,100), found!
1
6
2
3
4
5
1 : 3 01: 2
1 : 5 01: 2
1 : 4 00: 6
0 : 2 11: 5
0 : 6 11: 5
0 : 6 10: 4
stores data with prefix 00
stores data with prefix 00
stores data with prefix 01
stores data with prefix 10
stores data with prefix 10
stores data with prefix 11

Figure 5: P-Grid trie example.

2.1.5.1 *P-Grid Overlay topology*

P-Grid[2][1] abstracts a trie[14] and resolves queries based on prefix matching. The actual topology has no hierarchy. Queries are resolved by matching prefixes. This also determines the choice of routing table entries. Each peer, for each level of the trie, maintains autonomously routing entries chosen randomly from the complementary sub-trees. In fact, multiple entries are maintained for each level at each peer to provide fault-tolerance (as well as query-load management). For diverse reasons including fault-tolerance and load-balancing, multiple peers are responsible for each leaf node in the P-Grid tree. These are called replicas. The replica peers maintain an independent replica sub-network and

14 A trie, or prefix tree, is an ordered tree data structure that is used to store an associative array where the keys are usually strings. No node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are normally not associated with every node, only with leaves and some inner nodes that correspond to keys of interest.

uses gossip based communication to keep the replica group up-to-date. The redundancy in both the replication of key-space partitions as well as the routing network together is called structural replication. Figure 5 shows a P-Grid overlay topology.

2.2 RANGE QUERIES

Range Queries are a type of query in centralized databases that retrieves all records where some value is between an upper and lower boundary, the range boundaries. Distributed hash tables which were described in section 2.1 more extensively , can support exact match lookup operations but due to the hashing that is used for a more uniform load distribution among the network, hobble to implement efficiently range queries. The uniform hashing functions they use to achieve probabilistically good load balance is disastrous for range queries. Range queries are based on spatial locality which is however destroyed when keys are uniformly hashed before entering the network. As a result, DHTs cannot handle natively this kind of queries. Due to this drawback of the widely used DHT's many interest is given in the search of efficient ways for answering range queries in DHT's. Many ways are proposed which target in maintaining the good characteristics that DHT's have established in conjuction with algorithms which can handle the additional complexity of range queries. Most notable are illustrated in following sections.

2.2.1 *Space filling curves*

A space-filling curve[12] is a curve whose range contains the entire $2-dimensional$ unit square or for more dimensions the entire $k-dimensional$ unit hypercube. Intuitively, a "continuous curve" in 2 or 3 (or higher) dimensions can be thought of as the "path of a continuously moving point". To eliminate the inherent vagueness of this notion Jordan in 1887 introduced the following rigorous definition, which has since been adopted as the precise description of the notion of a "continuous curve": A curve (with endpoints) is a continuous function whose domain is the unit interval. In the most general form, the range of such a function may lie in an arbitrary topological space, but in the most common cases, the range will lie in a Euclidean space such as the 2-dimensional plane (a "plane curve") or the 3-dimensional space ("space curve"). The most common space-filling curves widely used in computer science are the following:

Z-ORDERING CURVES :Z-ordering space-filling curves has locality-preserving behavior and it is used in data structures for mapping multidimensional data to one dimension. The z-value of a point in multidimensions is simply calculated by interleaving the binary representations of its coordinate values. Once the data are sorted into this ordering, any one-dimensional data structure can be used such as binary search trees, B-trees, skip lists or (with low significant bits truncated) hash tables.

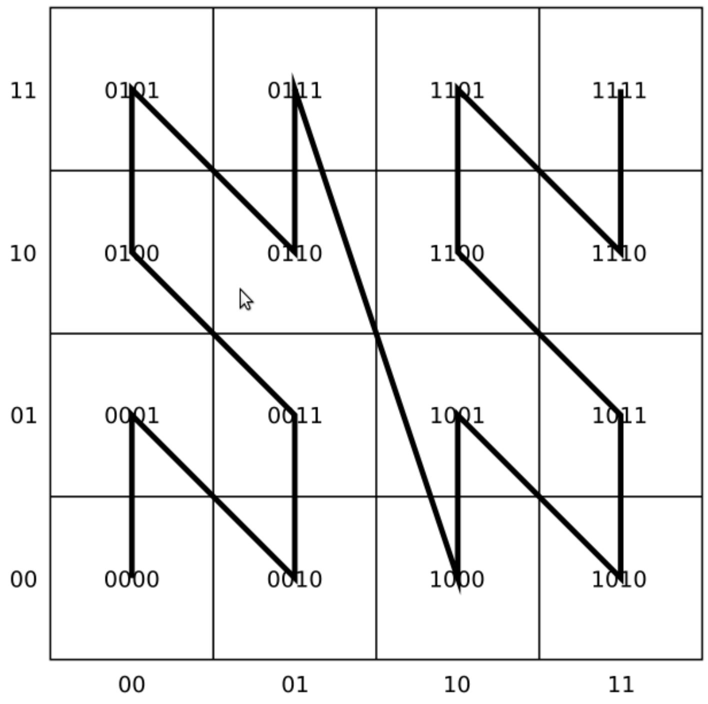


Figure 6: Z-order space-filling curve.

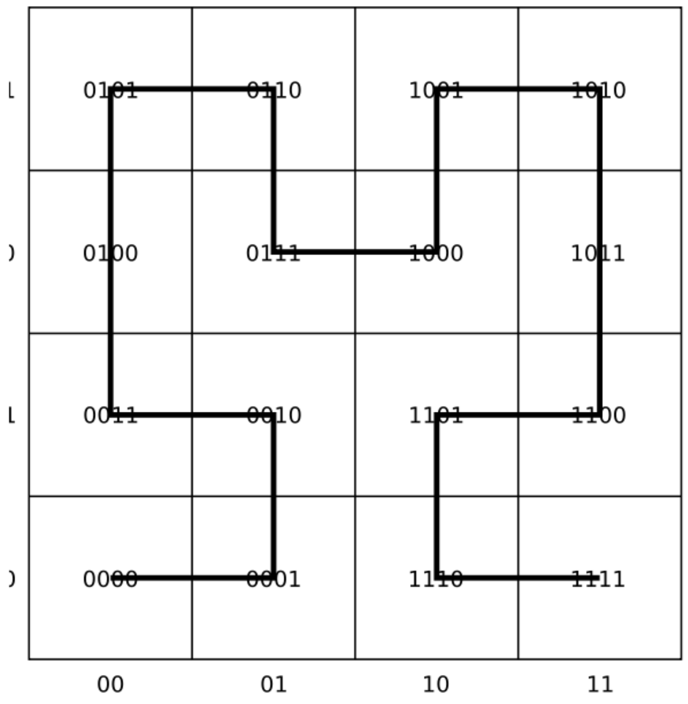


Figure 7: Hilbert space-filling curve.

HILBERT CURVES :A major drawback of z-order curves is that it does not preserve spatial locality. A more satisfying curve on this matter is the Hilbert curve[11]. On the downside, the algorithmic complexity of mapping K-dimensional points to 1 dimension using the Hilbert curve makes z-ordering usually a more desirable choice.

2.2.2 *Range queries in P-Grid*

As it was described above a range query comprises of a lower and an upper bound and the results set is the data within this range boundaries stored in the network. In P-Grid[4] boundaries are represented by binary strings and and the range query retrieves all element with keys between the two binary strings. The P-Grid has to main algortihms in order to perfom a range query, both have the same task but a different manner, the first is a sequential approach and the second a parallel one.

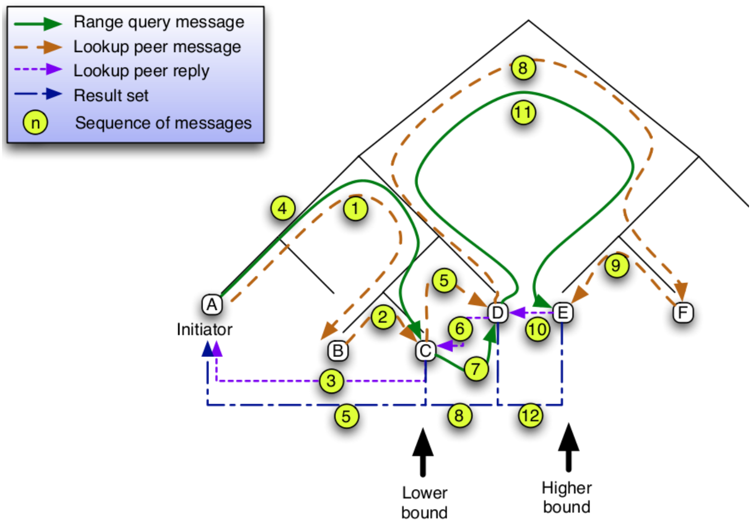


Figure 8: Min-Max Traversal algorithm example.

SEQUENTIAL APPROACH THE MIN-MAX TRAVERSAL ALGORITHM The Min-Max Traversal Algorithm[9] is based in sequential processing of the query by locating the subsets of the query in order. Subsequently only one peer performs a query at the same time. Firstly the peer responsible for the lower bound of the query is located, then the query is forwarded to the peer which handle the next subset of the query until all the query's subsets are found, and so the query is finalized at the peer which has the last subset of keys in which lies the upper bound of the query range. The described algorithm has two major drawbacks which make implementation of range queries difficult. The first is that it requires additional links in the peers' Routing tables except from the existing and so cannot be used directly in the traditional P-Grid infrastructure. Another drawback is that a single peer failure can interrupt a range query search and so the processing of the query may end up incomplete. Figure 8 shows a Min-Max Traversal algorithm example.

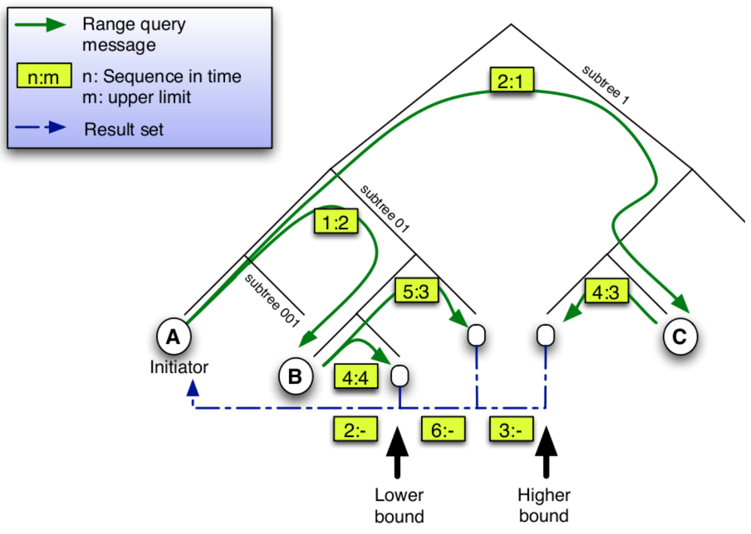


Figure 9: Shower algorithm example.

PARALLEL APPROACH THE SHOWER ALGORITHM The Shower algorithm[9] partially overcomes the drawbacks of the previous and is based in simultaneous processing of a query by multiple peers. The main idea is to split the P-Grid tree in subtrees and then delegate the query to arbitrary peers in each subtree. The initial query is forwarded in parallel to peers who have a part of the query result and then this query is forwarded recursively to the other peers who handle partitions of the query interval using each peer's routing table. Each peer discovers the peers who are registered in his Routing table and are responsible for a subset of the current query, and then forwards the query to them taking into account the relative position of the peer who forwarded the query to him in the virtual binary tree. Although it is possible to forward a query to a peer who does not have part of the result set, the P-Grid infrastructure guarantees that this peer will forward the query back to a peer whose keys are lying inside the query key-range. Despite the fact that this algorithm requires more messages than the previous is less prone to peer failures and gives faster results. Our protocol supports the Shower Algorithm for processing Range Queries. Figure 9 shows a Shower algorithm example.

2.3 LOAD BALANCING

Load balancing in peer to peer networks is a technique to distribute workload evenly across the network, in order to get optimal resource utilization, maximize throughput, minimize response time, and avoid overload in peers. In peer to peer networks load balancing algorithms try to achieve a more uniform distribution of data among peers because an uneven data distribution increases significantly the probability of overloading the peers at which greater amount of data is assigned. Also try to achieve a more uniform distribution of traffic among the peers considering the frequency of requests that each peer accepts, which is defined by the popularity of data that each peer is responsible for. Load balancing is a critical issue for the efficient operation of a peer-to-peer networks as it increases system's resistance in single point failures and network consistency.

2.3.1 *Load Balancing in CAN*

The main load balancing methods on the CAN[8] peer to peer protocol are the overloading of a coordinate zone and the use of multiple hash functions. Both are described below:

Overloading coordinate zone: The basic CAN topology assigns a coordinate zone to a unique node in the network. Now this attribute is modified and it is allowed to many nodes of the network to share the same zone by defining a maximum number of allowable peers per zone. Each node except from its neighboring list maintains a list of the peers that are responsible for its zone and needs to keep information only about one peer of each neighboring zone. Consequently there is no need for a node to keep additional neighboring information but it is only necessary to keep the state of all the peers which are co-responsible for its zone. The overloading zones offer the following advantages:

- Reduce the path length and thus path latency, because placing more nodes per zone has similar effect as reducing the number of nodes in the system.

- Reduce per-hop latency because now a node has multiple choices and at each hop can select the node which is closer in TCP-IP level.

- Greater fault-tolerance because a zone is unaccessible only if all the nodes of this zone had simultaneously failed.

On the other hand the overloading zones increase system's complexity because nodes must additionally track a set of peers.

Multiple Hash functions: Instead of using the traditional one hash function for key mapping the CAN uses k different hash functions in order to map one key in k different points in the coordinate space and simultaneously to replicate a single (key, value) pair at k distinct peers in the network. The use of multiple hash functions offers the following advantages:

- A key is unavailable only if all the nodes that are responsible for the coordinate zones in which the k replicas of the requesting key are lying, have simultaneously failed.
- Reduction of the average query latency because queries for a particular key could be sent to all k nodes simultaneously.
- Queries for a particular key can be sent only to the node which is closest and thus reduces per-hop latency.

The only drawbacks of multiple hashing is increased query traffic in the case of parallel queries and also additional cost by the increased size of the key database.

2.3.2 *Load Balancing in Chord*

As it was described in section 2.1.2 the Chord protocol uses consistent hashing for key mapping which offers a even distribution of keys in the network nodes. Additionally Chord makes the distribution of keys to the nodes more uniform by associating the keys to virtual nodes and then mapping these virtual nodes in the real nodes, thus we succeed a more uniform coverage of the identifier space. Adding virtual nodes as an indirection layer can significantly improve load balance. The tradeoff is that each real node now needs time as much space to store the finger tables for its virtual nodes.

2.3.3 *Load Balancing in P-Grid*

Load balancing in P-Grid, as it is extensively described in [7], is mostly based in storage balance algorithms which exploit the topology of the overlay trie. Load balancing algorithms are based in exchange interactions between two peers of the network and are separated in exchanges between peers with common path and non-common path.

Exchange interactions between peers with common path

- Become mutual replicas: If two peers meet during routing and have common path and the total amount of stored data is within the apprropriate limit they are willing to store, they can create replicas of each other's data.

- Extend paths: If two peers with the same path meet and they are not able to store any more data then they can extend their path by complimentary bits and then share their total data according to their new paths. Similarly, if peers meet where the path of one peer is a prefix of the other peer's path, the peer with the shorter path may decide to extend its path by a complimentary bit, and then this peer can split its data to the new nodes acoording to the new paths.

- Retract paths: If two peers with exactly the last bit of their path different meet, and together store less than their storage ability then they may decide to retract simultaneously their path. Similarly, if one peer's path is prefix of another peer's path, the peer with the longer path might retract by one bit, if it finds an amount of data in the corresponding leaf nodes which does not exceed its storage ability.

Exchange interactions between peers with non common path

- Initiation of further exchanges: The most common case where two peers meet randomly is to have incompatible paths. In this case the peer with the longest path searches for a registration in its routing table with path common with the path of the other peer, and so we can implement one of the previous exchanges between peers with compatible paths. This shows that any exchange interaction can be implemented as an exchange between two peers with compatible paths.

- Exchange routing tables: This exchange method implements the routing tables' exchange between two peers without no need for path compatibility between them. This method sends a request for exchange RT from a peer to another and then the recipient can accept it or not. If the exchange request is accepted by the recipient then the recipient should send its RT to the sender, and the exchange action is completed.

2.4 P2P NETWORK SIMULATORS

Typical peer-to-peer networks involve thousands of computers. They make heavy use of the network communication capabilities of the underlying hardware and typically computers join and leave continuously. These facts make researching and obtaining experimental results for peer-to-peer networks a very difficult, error-prone and time-consuming process. The only feasible way to approach such problems is to use realistic simulation models to extract useful information about the (virtual) network, before proceeding to real implementations. The academic community has created many specialized simulation tools for a variety of peer-to-peer networks. There is no single widely accepted framework for performing experiments on peer-to-peer networks; each individual solution has its own strengths and weaknesses and often focuses on different aspects of the problem.

2.4.1 *P2PSim*

P2PSim[15] is a discrete event packet level simulator that can simulate structured overlays only. It contains implementations of six candidate protocols: Chord, Accordion, Koorde, Kelips, Tapestry and Kademlia. Event scripts can be setup to simulate churn but neither the churn north node failure statistics are exhaustive. P2PSim can simulate node failures and both iterative and recursive lookups are supported. Node IDs are generated by consistent 160-bit SHA-1 hashing.vDistributed simulation, cross traffic and massive fluctuations of bandwidth are not supported. The C++ API documentation is poor, but implementation of other protocols can be built by extending certain base classes. Custom event generators can also be implemented by extending a base class. The P2PSim code suggests support for a wide range of underlying network topologies such as end-to-end time graph, G2 graph, GT-ITM, random graph and Euclidean graph, which is the most commonly used. P2PSim developers have tested its scalability with a 3, 000- node Euclidean ConstantFailureModel topology.

2.4.2 *PeerSim*

PeerSim[16] is an event-based P2P simulator written in Java, partly developed in the BISON project and released under the GPL open source license. It is designed specifically for epidemic protocols with very high scalability and support for dynamicity. It can be used to simulate both structured and unstructured overlays. Since it is exclusively focused on extreme performance simulation at the network overlay level of abstraction, it does not regard any overheads and details of the underlying communication network, such as TCP/IP stack, latencies, etc. Its extensible and pluggable component characteristics allow almost all predefined entities in PeerSim to be customized or replaced with user-defined entities. For flexible configuration, it uses a plain ASCII file composed of key-value pairs.

15 http://pdos.csail.mit.edu/p2psim/
16 http://peersim.sourceforge.net/

2.4.3 *OverSim*

OverSim[5]is a flexible overlay network simulation framework based on OMNeT++, designed by Ingmar Baumgart and Bernhard Heep and Stephan Krause in 2007. It was designed to fulfill a number of requirements that have been partially neglected by existing simulation frameworks. OverSim includes several structured and unstructured peer-to-peer protocols like Chord, Kademlia and Gia. These protocol implementations can be used for both simulation as well as real world networks. To facilitate the implementation of additional protocols and to make them more comparable OverSim provides several common functions like a generic lookup mechanism for structured peer-to-peer networks and an RPC interface. Several exchangeable underlay network models allow to simulate complex heterogeneous underlay networks as well as simplified networks for large-scale simulations.

2.4.4 *Rangesim++*

Rangesim++ is a simulator based on PeerSim that supports cycle-driven and event-based simulations. It is implemented by V. Samoladas in C++ , which greatly improves the time of experiments (about 100 times faster than its predecessor which is written in Java). We have used this simulator to get our experimental results for our proposed algorithms. Rangesim++ is described more thoroughly in chapter 4.

3 OUR APPROACH

In our approach the main target is to achieve a more uniform load distribution among peers, we try to alleviate the overloaded peers by transferring a portion of its load to the underloaded peers of the network. In order to achieve that, we need a metric which will represent a peer's traffic. By traffic here we mean the frequency that a peer accepts query requests. Consequently for the needs of our research we introduce a load-factor for each peer. The definition of load-factor is extensively presented in the following sections. This load factor information is registered for each peer neighbor as additional information in the routing tables of a peer. In our implementation, in the routing tables except from the existing information for each neighbor that the traditional P-Grid overlay has, such as the peer's path and its psychical address, a load factor also stored for each neighbor. Every time that a peer looks for the appropriate neighbor to forward the query except from the position in the overlay network takes under consideration and the neighbor's load factor so as to avoid increasing the load in already overloaded peers. The rerouting algorithms used in this case are thoroughly presented in the following sections.

3.1 ASSUMPTIONS

In order to focus in congestion control in P2P networks we need to make some simplifying assumptions of the studied protocol.The assumptions concern two major problems in the field of P2P networks which concern fault tolerance in the topology level and in application level.So our assumptions are the following:

- The underlying topology of the network is static, the network size and its topology is predefined and so the peers cannot enter or leave the network arbitrarily.
- The alternation of the P-Grid protocol which is used during the experiments does not support any data replication mechanisms, so every data item is assigned only to one peer.

3.2 DEFINITION OF LOAD FACTOR

Load factor is a quantity which represents the load at each peer in the network according to the frequency of the query requests that each peer receives. Each peer has an Input and an Output channel which are simulated as two PSQueues queues [1] ,queues with processing sharing policy. So the load factor of each peer must reflect the load at both queues. In our approach we define a load factor for the input and output channel of a peer respectively, and then we calculate the peer's load factor in relative with this two quantities. We introduce two approaches for calculating a channel's load, the time average and the time average with damping.

3.2.0.1 *Time average utilization*

In this approach we define as a channel's load factor the utilization[6] of a queue(a peer's channel is simulated as a PSQueue as it was described above) which is the fraction of the time that the queue is busy to the total time at which the system is observed. As long as the input and output channel of the peer are simulated as two PSQueues this definition is valid.

$$U = \frac{B}{T}$$

U : Utilization
B : The length of time which the queue is observed to be busy, namely the time at which there are packages in the queue waiting to be serviced
T : The length of time that we observe the queue, in our case is the simulation time

Although this definition of the Utilization is quite representative for the traffic of a queue we encountered some problems which may lead to inaccuracies. As the time passes and we observe the utilization of the queue, this quantity loses the ability to reflect satisfactory the current state of a queue. For example if we observe the queue for 1000s and the time at which the queue is busy is 200s, we conclude at a first glance that the queue does not have any difficulties of serving the packages. But we do not know if the time of 200s at which the queue is busy is at the first 250s of observation, or at the last seconds of the observation or if it is evenly distributed through the total observation time. So

1 This is a queue with Processor Sharing (PS) service policy. In our simulator the implementation is a slight approximation. An accurate implementation cannot be implemented, because the type at which simulation time is counted is an integral type. For example, assume that we have two concurrent customer arrivals with demands 2 and 3 respectively. Then, at time 1, another customer arrives, with demand 2. The first customer's exact departure time would then be 2.5, which is non-integral!) In practice, if service times have large values (in the thousands), the simulation is highly accurate. Thus, if accuracy is required, the time units of the simulation must be chosen appropriately.

it is possible that the queue did not have any packages at the first 750s of observation and at the last 250s started to receive packages constantly. This means that the utilization would be $U = 200s/1000s = 0.2$ value which indicates that there isn't any increased service demand while it is possible to be.

3.2.0.2 *Time average utilization with damping*

In order to overcome previous approach's problem we introduce a time-weighed estimation of a queue's utilization. In this second approach we take into consideration except from the observed value of the utilization, and the time at which this value is measured, in order to give increased importance in the most recent observations. So each observation of a peer's utilization is assigned with a weight according to the value of observation time, the most recent observations are assigned with greater weights and as the oldness of an observation is increasing the weights decrease. This approach offer us the ability to have a more representative view of a queue's load state due to the fact that reflects more effectively the current state of the queue without having great distortions from the old observations of the studying quantity. In order to assign the weights in the observed values of the queue's utilization we use an exponential frame with damping rate λ, which decreases according to the observation time. We define the time average utilization with damping [2], as follows:

Let $\phi(t)$ be a real function defined for $t \geqslant 0$.

Let $\mu_\varphi(t)$ be the time average at t, with damping rate λ, ie.

$$\mu_\varphi(t) = \frac{\lambda}{1 - e^{-\lambda t}} \int_0^t \varphi(\tau) e^{-\lambda(t-\tau)} d\tau$$

Now, assume that for $t' < \tau \leqslant t$, $\phi(t)$ **is constant** i.e $\phi(t) = x$. Let $\Delta t = t - t'$. Also, let $\mu = \mu_\varphi(t)$ and $\mu' = \mu_\varphi(t')$.

Then,

$$\mu = \mu' + (x - \mu')\frac{1 - e^{-\lambda \Delta t}}{1 - e^{-\lambda t}}$$

and

$$s = e^{-\lambda \Delta t} s' + (x - \mu)(x - \mu')(1 - e^{-\lambda \Delta t})$$

with

$$\sigma^2{}_\varphi(t) = \mu_{\varphi^2}(t) - \mu_\varphi^2(t) = \frac{s}{1 - e^{-\lambda t}}$$

According to the above we consider as the queue's load factor the mean value μ of the observed values of the queue's Utilization.

2 http://en.wikipedia.org/wiki/Moving_average

3.2.1 *Definition of a peer's load factor*

The load factor of a peer is defined in relation with the load state of its input and output channel. So in order to calculate a peer's load factor we check the values of each peer's input and output queues and then we conclude which is the load state of the peer.So our approach is case-oriented and we define the following four load states of a peer which are represented by the values 1,2,3 and 4 respectively:

1. None of the two queues is overloaded or about to become overloaded.
2. One or both queues are not overloaded but are about to become.
3. One of the peer's queues is overloaded.
4. Both input and output queues are overloaded.

By saying overloaded we mean that the utilization value tends to become equal to 1 which means that from the beginning of the observation until now a queue has constantly packages which are waiting to be served.

3.2.2 *Adjustment of the load factor for real P2P networks*

In an experimental process at which simulations take place, the observation of the quantities **B** and **T** is possible but in real life scenarios the length of time that we observe a channel is not easily measurable. In order to overcome this problem and to make our approach more applicable to real P2P networks we can substitute the variant **T** with the following alternative measures:

- The length of time at which a peer is in the network, the time interval at which a peer joined the network until current time. This definition might not be very efficient due to the fact that the portion of time that each peer is observed can be different with great variations, since the peers join and leave the network constantly, and furthermore is technically difficult to measure a quantity for so big length of time. In this case the time average utilization metric as time passes becomes less representative of a peer's traffic while the time average utilization with damping does not encounter similar problems.
- The second alternative is to define a time interval at which all the peers' load factor will be reseted and then will start calculating of a peer's utilization from the beginning. Since our experiments' duration is approximately 500s, and we

observe a good performance considering the estimation of utilization, we assume that the second alternative is efficient and the appropriate time-interval duration easy to be chosen. Additionally the second alternative overcomes the problems which appears the first approximation. In this approach both definitions of utilization perform satisfactory.

3.3 UPDATE ALGORITHMS

For each registration at a peer's Routing Table we store the load factor of the corresponding neighbor peer and also the time that this load factor is estimated so as to know how recent is the value of the stored load factor. This information about the neighboring peer's load must be up to date so as to take optimized decisions about a more effective routing path without having or creating additional load. Our aspect is that the registered values of neighboring peers load must be as consistent as possible to the real values of each neighbor load. So we introduce 5 algorithms for efficient updating of the neighboring peers load information which is stored in each peers Routing Table.

3.3.1 *Piggy Bag*

Every time a message is sent the sender peer sends with the message its load factor. When the message will be received and will be processed, the recipient will check if there is a registration of the sender peer in its Routing Table, if there is the stored value of load factor and the corresponding timestamp will be replaced by the new.

3.3.2 *Polling Periodically*

We define a specific time interval which we call polling interval in which each peer informs its neighbors about its recently estimated load factor. When this time interval is elapsed the peer sends a message at each peer registered in its Routing Table, which contains the sender's load factor. When a neighbor receives a polling message from a peer it first checks its RT and if there is a registration matched with the sender the stored load factor is updated. Then sends its load-factor back to the sender of the polling message. This offers us the ability to update the stored values as often as we consider appropriate but creates additional traffic to the network for the additional messages which need to be sent. We also have a slight alternation of the polling periodically algorithm where the neighbors are not obliged to send back their load factors. We introduced this alternated algorithm in order to reduce the quantity of additional messages which are created by the polling, by keeping at the same time the ability to attempt for an update as often as we wish.

3.3.3 *Back of Routing Path*

When a query is created, it must be forwarded inside the network to be answered. During the forwarding the query goes through many peers of the network which constitute the query's routing path. This algorithm stores the load factor of each peer which is added to the routing path, and for each new added peer in the routing path checks if there is a registration in the RT which matches with a registration of the current routing path. If there is the corresponding load factor is updated. This algorithm offers updating without creating additional messages due to the fact that the information is enclosed to the original messages.

3.3.4 *Requesting for Common Neighbors*

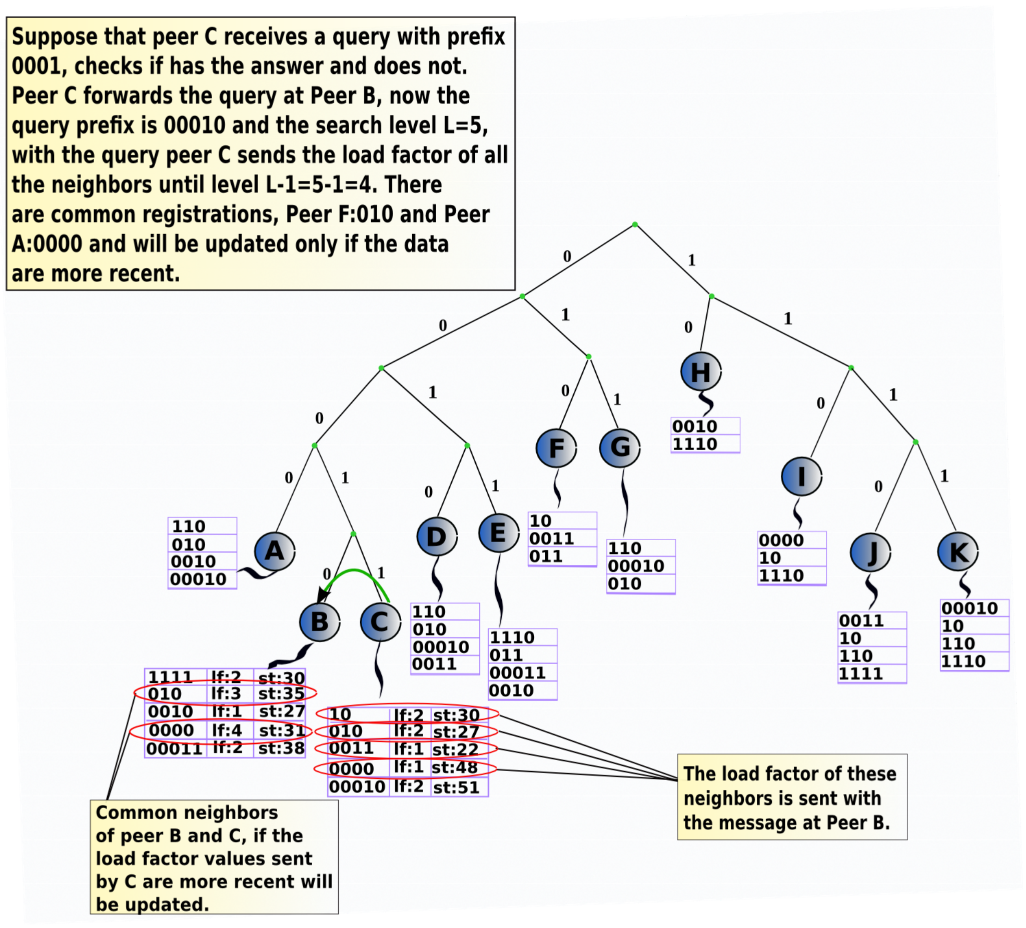
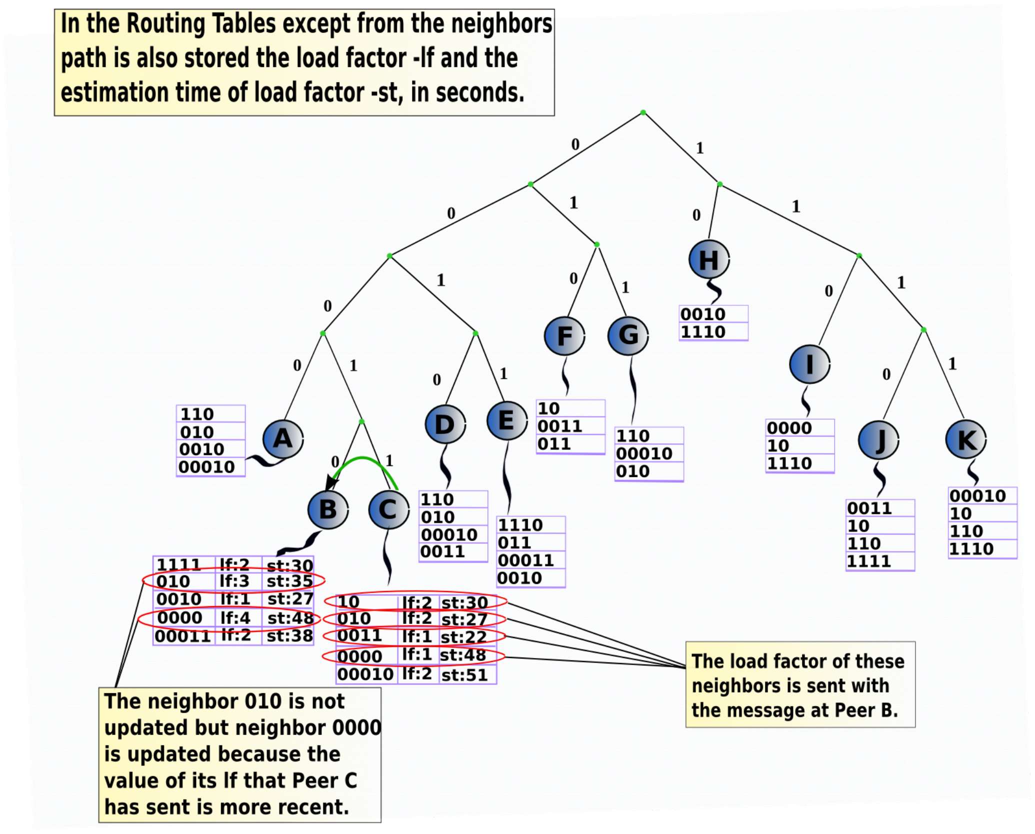

Suppose that peer C receives a query with prefix 0001, checks if has the answer and does not. Peer C forwards the query at Peer B, now the query prefix is 00010 and the search level L=5, with the query peer C sends the load factor of all the neighbors until level L-1=5-1=4. There are common registrations, Peer F:010 and Peer A:0000 and will be updated only if the data are more recent.
0
1
H
0010
1110
F
10
0011
011
G
110
00010
010
I
0000
10
1110
J
0011
10
110
1111
K
00010
10
110
1110
A
110
010
0010
00010
D
110
010
00010
0011
E
1110
011
00011
0010
B
1111 lf:2 st:30
010 lf:3 st:35
0010 lf:1 st:27
0000 lf:4 st:31
00011 lf:2 st:38
C
10 lf:2 st:30
010 lf:2 st:27
0011 lf:1 st:22
0000 lf:1 st:48
00010 lf:2 st:51
The load factor of these neighbors is sent with the message at Peer B.
Common neighbors of peer B and C, if the load factor values sent by C are more recent will be updated.

Figure 10: Example of common neihgbors update algorithm part 1.

According to P-Grid protocol a peer has at least one neighbor for each different subtree of each different level of the underlying trie. So every time a message is sent we also sent back of the message information about the load of the sender's neighbors which correspond to the subtrees of levels smaller than the current search level. This means that if the message is in search depth l we send information for the neighbor peers until depth l-1, namely for all the neighbors of all the existing subtrees except the one that both sender and recipient belong to. We do this because there is possibility some of the sender's and recipient's neighbors for the corresponding levels to be common, and so the load factor of the common neighbors is updated with the new which is sent back of the message. This algorithm offers updating without creating additional network traffic due to the fact that the information is enclosed to the original messages. Figures 10 and 11 present a simple example of the algorithm.



In the Routing Tables except from the neighbors path is also stored the load factor -lf and the estimation time of load factor -st, in seconds.

1111	lf:2	st:30
010	lf:3	st:35
0010	lf:1	st:27
0000	lf:4	st:48
00011	lf:2	st:38

10	lf:2	st:30
010	lf:2	st:27
0011	lf:1	st:22
0000	lf:1	st:48
00010	lf:2	st:51

The neighbor 010 is not updated but neighbor 0000 is updated because the value of its lf that Peer C has sent is more recent.

The load factor of these neighbors is sent with the message at Peer B.

Figure 11: Example of common neighbors update algorithm part 2.

At this point we need to notice that this algorithm does not increases the message size significantly. Suppose that we have a network that constitutes of 100000 peers. In the mean case a peer has a Routing Table with $O(\log N)$ registrations, where N is the network size. Now think that we are just one hop away from the peer that has the answer and we need to store all load information of the possible common neighbors' of the current peer with the destination peer, in the message we are sending. A peer's load factor is represented by a double variable of $8bytes$, and its store time by a long long int $64bits = 8bytes$. A peer's pathid length differentiates according to the trie and can be from 1 to $Nbits$ where N the network size, but in the mean case is $logN$ $bits = log100000 = 16bits$. In order to store all the necessary information we need :

$$RT\_size * (pathid\_size + load\_factor\_size + store\_time\_size) =$$

$$logN * (logNbits + 8bytes + 8bytes) =$$

$$log100000 * (log100000bits + 16bytes) =$$

$$16 * (16bits + 16bytes) =$$

$$16 * 18bytes =$$

$$288bytes =$$

$$2.88kbytes.$$

So it is feasible to support $2.8kbytes$ more per message.

3.3.5 *Exchange Routing Tables*

The exchange method implements the routing tables' exchange between two peers. This method sends a request for exchange RT from a peer to another and then the recipient can accept it or not. If the exchange request is accepted by the recipient then the recipient should send its own RT to the sender, and the exchange action is completed. The exchange requests can only be sent in specific times during the simulation ,so we define an exchange time interval, when this exchange interval is elapsed we check if we need to do an exchange request to a neighbor. In other words this exactly is the common exchange method of the P-Grid protocol with the difference that the exchange take place only in the two following cases:

- When the majority of the registered neighbors in the RT have load factor values which indicate that they are overloaded.
- When the majority of the registered values of the neighbors load factor are old and thus unrepresentative of the neighbors load state, and so need to be updated.

3.4 REROUTING ALGORITHMS

Before introducing the new rerouting algorithms we must note that we have two categories of forwarding. The first is the already known common forwarding and the second is the just-routing forwarding. The second takes place when the destination peer is overloaded and the receiving of additional messages will hinder the peer's load state, in this case we just forward the message without implementing any search mechanism until a properly loaded peer in the corresponding search level is found. Until this happens the forwarding messages transmit load information among the peers which belong to the routing path and so we don't forward the messages with no purpose.

3.4.1 *Dynamic Routing*

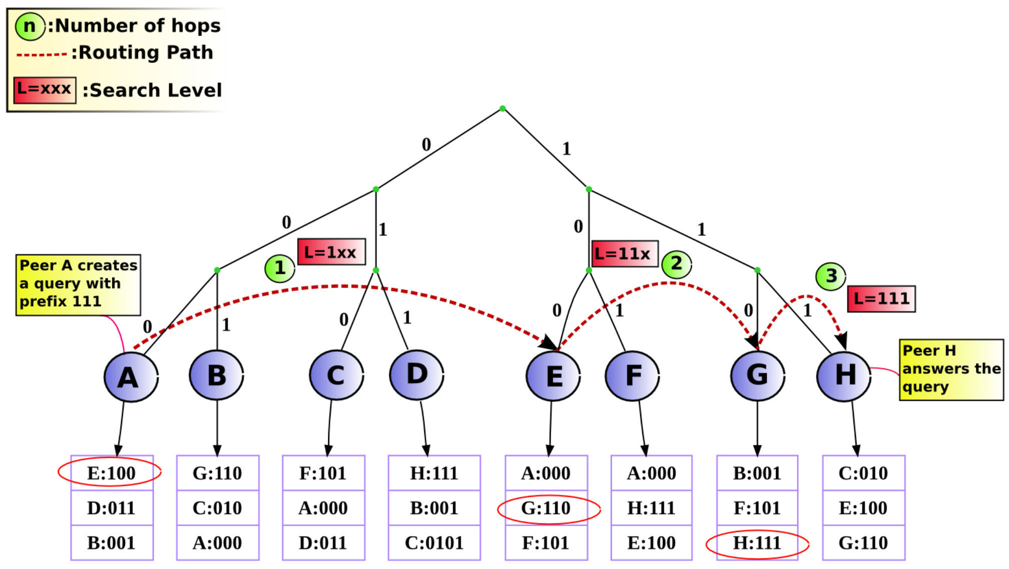


Figure 12: Regular Routing in P-Grid. Initiating, forwarding and answering a query.

In Dynamic Routing algorithm, when the destination peer[3] is overloaded or is about to become, the least loaded neighbor of the levels greater than the message's current search level is selected as destination peer instead of the original destination peer. Here we need to mention that we cannot select neighbors which correspond to level smaller than the current search level as alternative destination peer, due to the fact that the search would go back and forth and the search algorithm would not operate at all. When the regular destination peer is overloaded, we select an alternative destination peer and so the dynamic routing is triggered, this routing continues until a neighbor of the appropriate search level will be found not overloaded. When

3 Is the peer which is selected by the search algorithm as the appropriate peer for the query to be forwarded, according to the range of the query

this happens the dynamic routing stops and we go on with the regular forwarding. During the dynamic routing when the search stops at the current level, we just transfer load information among the peers using the update algorithms described above until a neighbor of the corresponding level at which we stopped the search will be found not overloaded. The functions below implement Dynamic Routing and are methods of struct Peer :

Listing 3.1: Pseudocode of Dynamic Routing

```
%This function just sends a message at a peer.Has 4 arguments
    , the query, the destination peer,
%the current search level and a boolean argument. The last
    argument of the forwardMessage function is triggered if
    the routing
%path has changed and so when the peer will process the
    message, will act accordingly.


void ForwardMessage(Query query, Peer* destination, size_t
    level, bool dynamic_routing_flag){
        Message* msg=new Message(query);
        %sends the message to a peer
        send_message(destination, level, dynamic_routing_flag
            );
}

%This functions checks if the regular destination peer is not
     overloaded and acts properly.
smart_forward(Peer* destination, size_t level){
        if (destination is saturated) then
                Peer* new_destination=find_least_saturated_
                    neighbor(level);
        if (new_destination==NULL) then
                ForwardMessage(Destination, level, false);
        %if we find a underloaded peer we forward the message
             at it
        else if (new_destination->load_factor<destination->
            load_factor)
                ForwardMessage(new_destination, level, true);
        %if we cannot find an underloaded neighbor we send
            the message to the regular destination
        else
                ForwardMessage(destination, level, false);
}


%This function finds the least saturated peer of the Routing
    Table for level greater than the argument level.
%If all the neighbors  in level grater than the argument
    level are overloaded returns NULL
Peer* find_least_saturated_neighbor(size_t level){
    bool flag=false;
    Peer* best_neighbor=new Peer();
    best_neighbor->load_factor=worst_value;
   %We search for an underloaded neighbor in level greater
       than the current search level,
   %because we don't want the search information we have
       until this search level for the current query
   for(int i=level+1;i<RoutingTable.size();i++)
       if(RoutingTable[i]->load_factor<best_neighbor) then
          best_neighbor=RoutingTable[i];
          flag=true;
    if(flag) then
          return best_neighbor;
    else
          return NULL;
}
```

We present a simple example of the dynamic routing algorithm in the figures below:

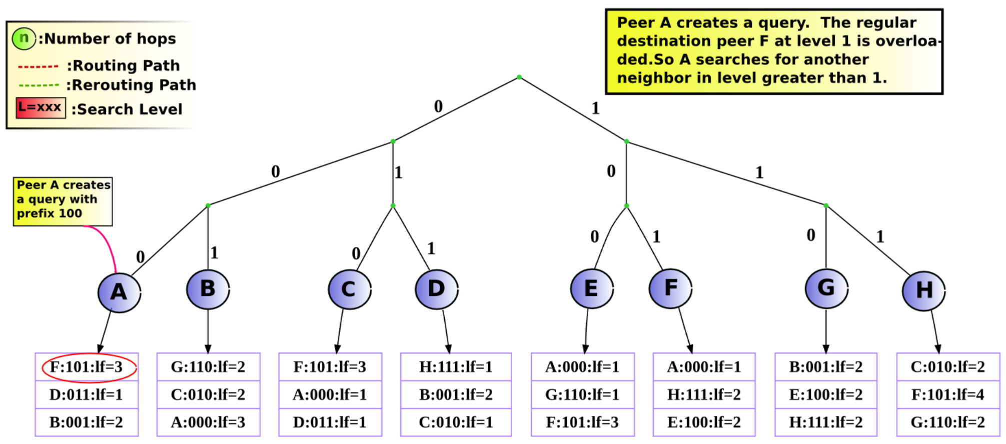


Figure 13: Initiating a query.

In figure 13 we see a small P2P network based in the P-Grid overlay with 8 peers, the trie is balanced but this is of minor importance. We observe that in the Routing tables the load factor of each neighbor is also registered , this information is now used during routing. Peer A creates a query and wants to forward it in a subtree with prefix 1xx. As we observe the only neighbor of A in the subtree with prefix 1xx is overloaded[4]. So A looks for the least saturated neighbor in levels smaller than the current search level which is level 0 for the moment. Peer A chooses Peer D and forwards the query to it. We should notice that the search has not started yet, A sends the query at D by informing D that the current search level remains 0, see figure 14.

Peer D receives the query and check if it has the answer. Observes that it does not have the answer and looks for a neighbor

4 In section 3.2.1 we present the values of the peer's load factor and we defined that the value 3 indicates that at least one of the peer's channel is overloaded

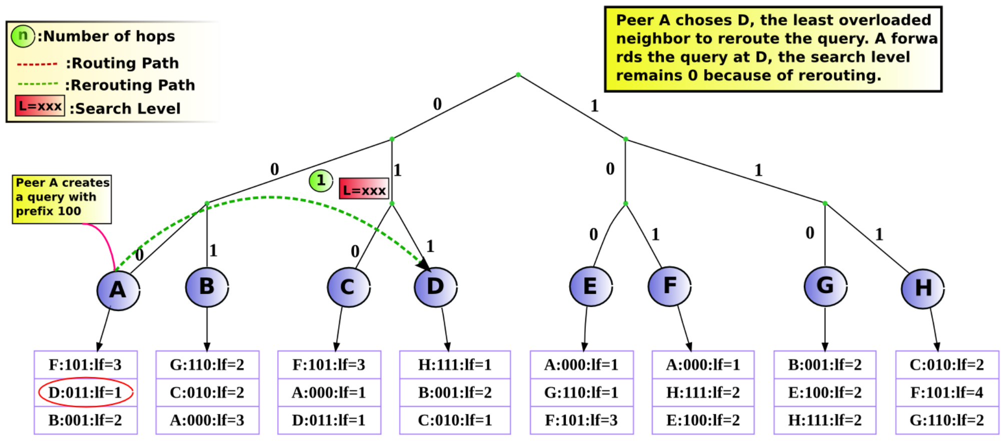


Figure 14: The regular destination is overloaded, we do rerouting.

to forward the query. The query must be forwarded at peer H , D sees that neighbor peer H is not overloaded and forwards the query at H with search level 1. The query now has a prefix 1. See figure 15.

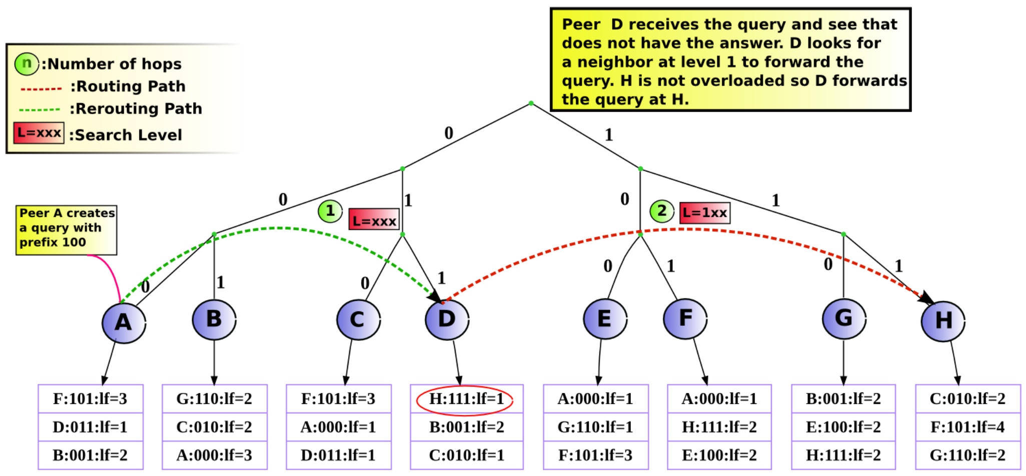


Figure 15: Regular Routing, D forwards a query at H.

Peer H receives the query, checks if it has the answer, it does not and so the query needs to be forwarded at a neighbor in level 2. The neighbor F in level 2 is overloaded (see figure 16), so we look for an underloaded neighbor at levels greater than 2.

The only neighbor in level greater than 2 is G and is not overloaded. H forwards the query at G (see figure 17) by preserving the search level to 1 because of rerouting. If G was overloaded the query would be forwarded to regular destination F because all the alternative destination peers would be overloaded.

Peer G receives the query, checks if it has the answer and it does not, so it looks for a neighbor at level 2 to forward the

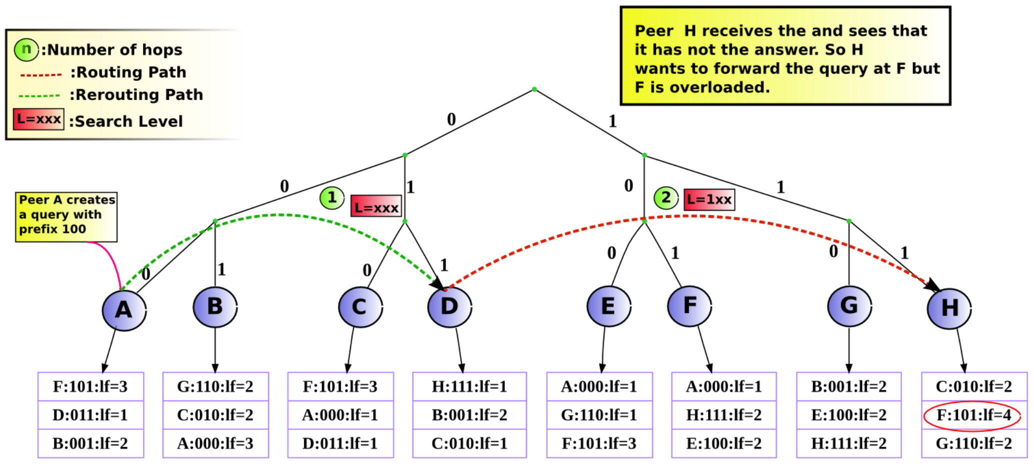

n :Number of hops
:Routing Path
:Rerouting Path
L=xxx :Search Level
Peer H receives the and sees that it has not the answer. So H wants to forward the query at F but F is overloaded.
Peer A creates a query with prefix 100
1 L=xxx
2 L=1xx
A B C D E F G H
F:101:lf=3 D:011:lf=1 B:001:lf=2
G:110:lf=2 C:010:lf=2 A:000:lf=3
F:101:lf=3 A:000:lf=1 D:011:lf=1
H:111:lf=1 B:001:lf=2 C:010:lf=1
A:000:lf=1 G:110:lf=1 F:101:lf=3
A:000:lf=1 H:111:lf=2 E:100:lf=2
B:001:lf=2 E:100:lf=2 H:111:lf=2
C:010:lf=2 F:101:lf=4 G:110:lf=2

Figure 16: Regular destination is overloaded.

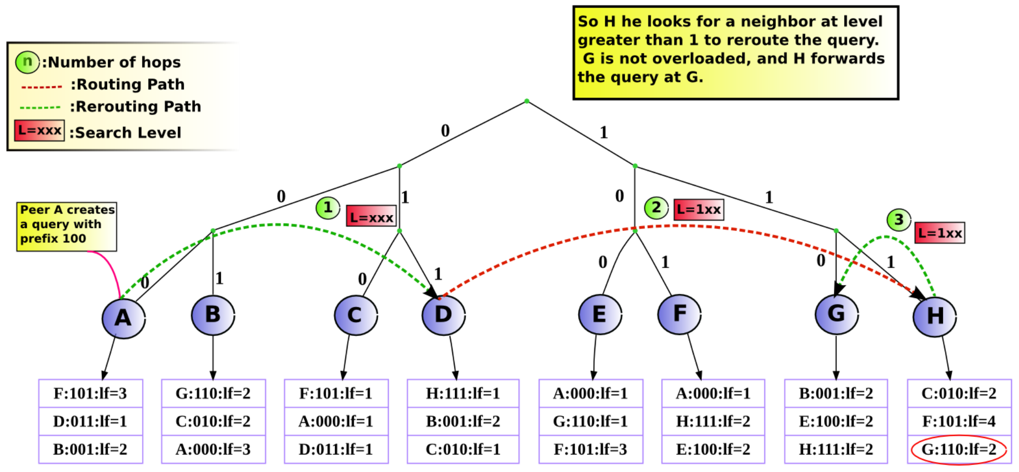

n :Number of hops
:Routing Path
:Rerouting Path
L=xxx :Search Level
So H he looks for a neighbor at level greater than 1 to reroute the query. G is not overloaded, and H forwards the query at G.
Peer A creates a query with prefix 100
1 L=xxx
2 L=1xx
3 L=1xx
A B C D E F G H
F:101:lf=3 D:011:lf=1 B:001:lf=2
G:110:lf=2 C:010:lf=2 A:000:lf=3
F:101:lf=1 A:000:lf=1 D:011:lf=1
H:111:lf=1 B:001:lf=2 C:010:lf=1
A:000:lf=1 G:110:lf=1 F:101:lf=3
A:000:lf=1 H:111:lf=2 E:100:lf=2
B:001:lf=2 E:100:lf=2 H:111:lf=2
C:010:lf=2 F:101:lf=4 G:110:lf=2

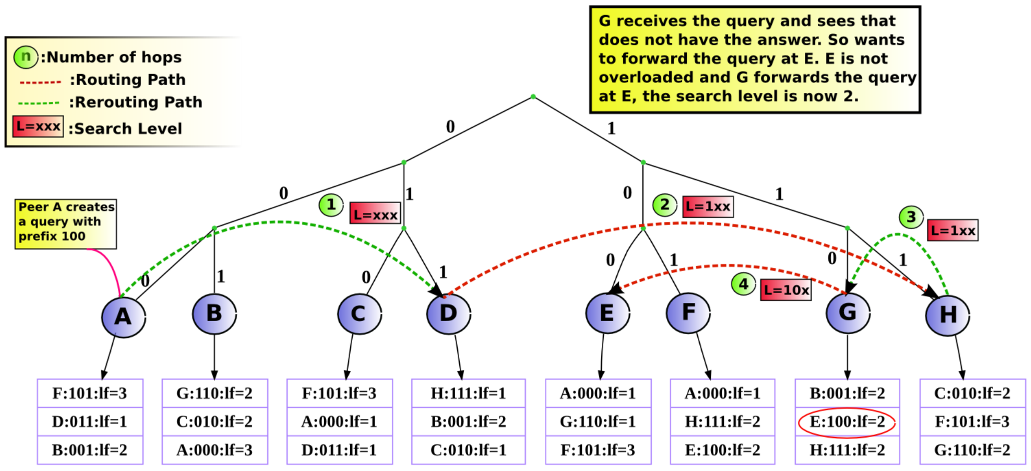
Figure 17: Peer H reroutes the query at G.

query. Peer E the neighbor for level 2 is not overloaded and so G forwards the query at E by increasing the search level at 2. Now the query has the prefix 10x, see figure 18.

Peer E receives the query and checks if it can answer it. Peer E sees that it has the answer for the query with prefix 100 and the search is completed (figure 1). **We observe that every time we reroute a query the P-Grid overlay guarantees that the alternative destination peer will have a neighbor for the level at which we stopped the search and so it will continue normally after one additional hop.**

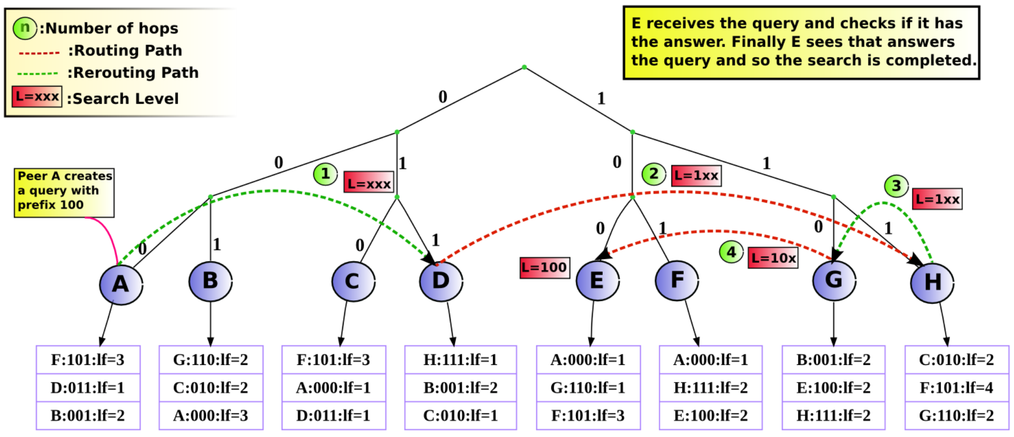
3.4.2 *Dynamic Routing with restricted number of additional hops*

This algorithm is similar to the previous. The difference is that the query's regular routing can change only for a specific number



n :Number of hops
:Routing Path
:Rerouting Path
L=xxx :Search Level
G receives the query and sees that does not have the answer. So wants to forward the query at E. E is not overloaded and G forwards the query at E, the search level is now 2.
Peer A creates a query with prefix 100
1 L=xxx
2 L=1xx
3 L=1xx
4 L=10x
A B C D E F G H
F:101:lf=3 D:011:lf=1 B:001:lf=2
G:110:lf=2 C:010:lf=2 A:000:lf=3
F:101:lf=3 A:000:lf=1 D:011:lf=1
H:111:lf=1 B:001:lf=2 C:010:lf=1
A:000:lf=1 G:110:lf=1 F:101:lf=3
A:000:lf=1 H:111:lf=2 E:100:lf=2
B:001:lf=2 E:100:lf=2 H:111:lf=2
C:010:lf=2 F:101:lf=3 G:110:lf=2

Figure 18: Regular routing, Peer G forwards the query at E.



n :Number of hops
:Routing Path
:Rerouting Path
L=xxx :Search Level
E receives the query and checks if it has the answer. Finally E sees that answers the query and so the search is completed.
Peer A creates a query with prefix 100
1 L=xxx
2 L=1xx
3 L=1xx
4 L=10x
L=100
A B C D E F G H
F:101:lf=3 D:011:lf=1 B:001:lf=2
G:110:lf=2 C:010:lf=2 A:000:lf=3
F:101:lf=3 A:000:lf=1 D:011:lf=1
H:111:lf=1 B:001:lf=2 C:010:lf=1
A:000:lf=1 G:110:lf=1 F:101:lf=3
A:000:lf=1 H:111:lf=2 E:100:lf=2
B:001:lf=2 E:100:lf=2 H:111:lf=2
C:010:lf=2 F:101:lf=4 G:110:lf=2

Figure 19: Peer E answers the query.

of times which are defined by the variable maximum additional hops per message which is given from each experiment's configuration file. This means that if the destination peer is overloaded then we can select another neighbor as destination peer, but we can to this successively for a restricted number of times which is defined by the variable *max_hops*. The pseudocode below implements the current type of routing.

Listing 3.2: Pseudocode of Dynamic Routing with Restricted number of additional hops per query

```
%The variable max_hops is global and it is defined from the
    configuration file for each experiment respectively
smart_forward(Peer* Destination,size_t level,Process* proc){
       if (Peer_is_saturated(Destination)&&current_hops<max_
           hops) then
              Peer* NewDestination=find_least_saturated_
                  neighbor();
              proc->current_hops++;
       if(NewDestination==NULL) then
              ForwardMessage(Destination,level,false);
              proc->current_hops=0;
       else if(NewDestination->load_factor<Destination->load_
           factor)
              ForwardMessage(NewDestination,level,true);
              proc->current_hops++;
       else
              ForwardMessage(Destination,level,false);
              proc->current_hops=0;
}
```

The other functions remain as were described above.

3.4.3 *Best Neighbor Dynamic Routing*

This algorithm is quite simple but similar to the previous. It can be implemented under the condition that we have 2-dimensional Routing Tables. Every time that we should forward a query we select the least loaded neighbor for the corresponding search level, instead of a random neighbor selection, and forward the message to it. Also we need to notice that we still use the load information that we store for each neighbor so as to get a more optimized forwarding without creating additional routing messages (as we did in the two previous algorithms) and this is a case that we consider interesting to study.

4 SIMULATION FRAMEWORK

A brief overview of the total work will be presented in this chapter. At first we will present the Datasets and the corresponding Querysets used during the experimental process. We will then give an abstract description of the Rangesim simulator which was used during the simulation process. Finally we will thoroughly describe the simulation framework for evaluating our proposed P-Grid protocol with range query support.

4.1 DATASETS / QUERYSETS

The Datasets used during the simulation are the following:

Greece: This dataset was constructed from real geographic data retrieved from the R-treeportal[1]. It contains random points distributed throughout the road network of Greece.

Hypersphere: This dataset contains points distributed on a hypersphere[2].

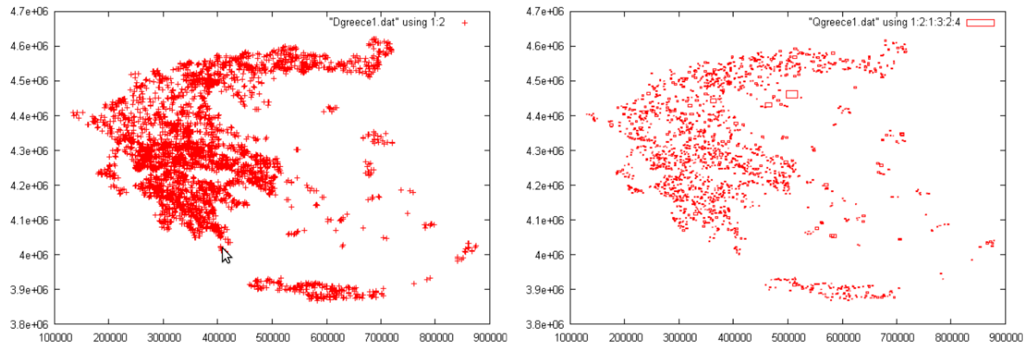


Figure 20: Dataset and Queryset - Greece.

For each of these datasets, three querysets were created respectively containing 30000 queries. Each query returns approx-

1 http://www.rtreeportal.org/

2 The n-hypersphere (often simply called the n-sphere) is a generalization of the circle (called by geometers the 2-sphere) and usual sphere (called by geometers the 3-sphere) to dimensions $n >= 4$. The n-sphere is therefore defined (again, to a geometer; see below) as the set of n-tuples of points

$$(x_1, x_2, ..., x_n)$$

such that

$$x_1^2 + x_2^2 + ... + x_n^2 = R^2$$

where R is the radius of the hypersphere.

imately 50 keys as an answer. A query is a rectangle, centered around a randomly chosen point of the dataset. Thus, the distribution of query ranges is similar to the distribution of datasets.

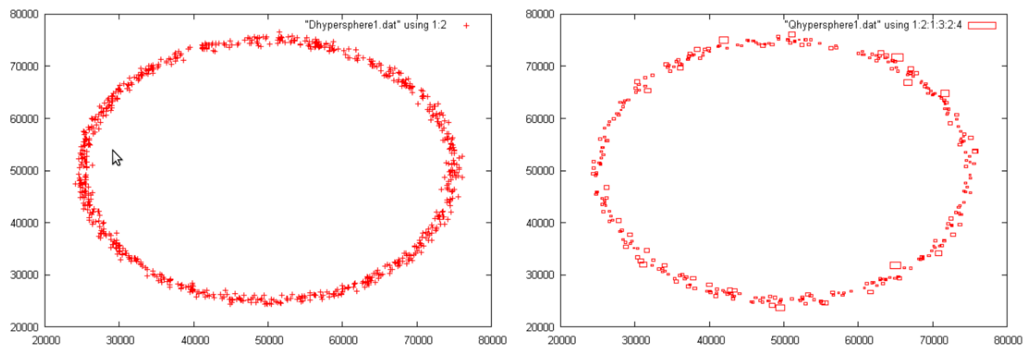


Figure 21: Dataset and Queryset - Hypersphere.

4.2 NETWORK SIMULATION

In this section we will present the simulation framework that we implement in order to obtain our experimental results. Our framework is based on the Rangesim++ simulator which is based in PeerSim and fully supports peer to peer network experiments. Rangesim supports some basic DHT's including a simple implementation of the P-Grid protocol in which our research focuses on. The main features of the Rangesim++ framework are:

- A convenient logging library to export information and results during the simulation process.
- A library used to seed statistical metrics, valuable for each experiment, during the simulation process.
- Supports both event-driven and cycle-based simulations.
- Random number generation facilities.
- Data point, Range and Queryset classes for Multidiamensional Range Search.
- It is fully configurable.
- Supports data balancing such as volume balancing algorithms concerning data distribution among the network.

4.2.1 *Cycle-based and Event-driven simulation*

As it was mentioned above our framework supports both cycle-based and event-driven simulation models which are presented below.

In the cycle-based model there is direct communication between peers in order to alternate the control of the simulation on a sequential order. When a peer has the simulation control can act arbitrarily, perform computations or call methods of other objects. This model is far simpler than the second but lacks on realism according to the real peer to peer networks behavior and functionality.

In the event-driven model we have a global simulation time and each operation is associated with a time delay. Each event carries a timestamp, a time-related information, and the event handling is conducted according to the time priority of the events which are sorted in a queue. The event whose the timestamp is closer to the global simulation time is processed first, then we perform the predefined operation and finally we define a new timestamp which is the sum of the old timestamp and the corresponding delay of the executed operation. A process can be enacted from any peer in the network and at any simulation time, peers may receive messages from other peers as part of this process. After processing an incoming message, the peer produces a number of outgoing messages to continue the process and goes on. The event-driven model is far more realistic than the cycle based because simulates effectively the synchronous behavior of the real peer to peer networks where all the peers can operate in parallel. Also continuous time delays such as communication latencies can be simulated with great accuracy.

In our experimental framework we use the event-driven simulation due to its great similarity with the real life scenarios. Also due to the fact that our research focuses in network traffic load it was more convenient to compute and measure our metrics in this simulation model where all the peers operate in parallel.

4.2.2 *Rangesim++ Design Structure*

The Rangesim++ simulator is designed to be fully configurable in order to make modular programming easier and to avoid repetitive recompilation processes. A simple configuration file is the argument to the simulator, given as a command line parameter. The configuration contains all the simulation parameters concerning all the objects involved in the experiment. The simulator reads the configuration file and sets up the network, initializing all the simulation parameters which were given through the configuration file. The input parameters defined in the configuration file are the following:

- Definition of Datasets and Querysets.
- Network size meaning the number of peers which constitute the network, we can define multiple sizes.

- Repetition number for each experiment.
- Network construction algorithm and topology.
- Total simulation time.
- Boolean flags for triggering the update algorithms of our protocol.
- Time variables which define how often the peers will do exchange and polling.
- For the dynamic routing algorithm with restricted number of hops the variable which defines the maximum number of continuous hops.
- The dimensions of the peers' Routing Tables, depth and length.
- The channel latencies for the peers' input and output channel respectively.
- Report path where the results of each experiment are stored.

Listing 4.1: An example of a configuration file

```
simulator = "PGridDES";
 #   rndgen_seed = 342898233;
%Information for the dataset and the queryset
    workload = {
            dimension=2;
            datafile = "workloads/D1M.dat";
            queryfile = "workloads/Q10K.dat";
    };
%Information for the wokload's parameters
    simulation = {
            % N.B. this is a double for precision reasons (
                internally time is 64-bit integral)
            maxtime = 500.0e6;  #  (in microseconds)
            polling_back=false;
            polling_interval=5000000;%5 sec
            exchange_interval=50000000;%50 sec
            process_path=true;
            common_neighbors=true;
            max_no_hops=100000;
            load_factor=false;
            utilization=true;
            damping_flag=true;
            damping_rate=50000000;%50 sec
    };
%Information for the network model
    network = {
            repeat = 1;
            size = [100000];
            topology = "volume";
            peer_model = {
                    input_channel = {
                            message_latency = 500.0E3;
                    };
                    output_channel = {
                            message_latency = 1000.0E3;
                    };
                    user_think_time = 3.E5;
            };
            rt_depth=32;
            rt_length=3;
    };
%Information for the result path
    reports = {
            path = "results/damping_50s_think_time_0.3s/id_
                pgrid_damping_50s_first_hops_unlimited_volume
                _Q10K_100000_8_outp$Nr$R";
            perrun_path = "results/";
            fileset = {
                    Rformat = false;

                   Perrun = true;
                    Topology= false;
            };
    };
```

4.2.3 *Classes Description*

A brief description of the classes our framework consists of, is given in the following table:

Class	Description, Attributes and Methods
Simulation	Holds all the configuration parameters of each experiment. Has the simulation time, a vector with pointers to all the peers of the network. A vector with all the Datasets and a vector with all the Querysets. Also holds a pointer to the P-Grid trie root.

Peer	Is a virtual peer. Holds a 2-dimensional Routing Table, a pointer to the corresponding trie node, and an input and output channel object. Also has all the variables necessary for the computation of the load factor. Finally holds statistic variables regarding the messages, the processes and the load factor. Also has methods that implement all the update and rerouting algorithms of our proposed protocol.
SimOrchestration	Main class orchestrating a whole P-Grid simulation. Reads the input parameters creates the simulation objects and initiates all the necessary variable regarding each experiment.
InputChannel & OutputChannel	Holds variable concerning the channel latency and also all the variables needed for the computation of the channel's utilization. Also has methods regarding the scheduling of the queue, the receiving and the sending of the messages.
TrieNode	Holds a bitstring denoting its path from the root and a pointer to the trie root. Also holds the data range that occupies and pointers to its parent and children.
PeerNeighbor	Implements a registration in a peer's RT. Holds a pointer to the corresponding neighbor, its load factor and the calculation time for it.
RoutingTable	Is a 2-dimensional table. Implements all the methods, for finding, updating a neighbor's load factor, replacing a neighbor with another etc.
InfoTraffic	It is the class used for transmiting information for the peers' load state. It is used by the update algorithms in order to transmit as less information as possible. Contains the unique pathid of the trie node, the load factor of the corresponding peer and its calculation time.
Process	A process object is responsible for initiating, searching and answering a query. Creates all the necessary number of messages in order to complete a query. Holds a pointer to the initial peer, a pointer to the query and a pointer to the corresponding simulation object.

Message	A message is sent in the context of a process. Holds a pointer to the recipient, a pointer to the sender and a pointer to the corresponding process. Also holds information about the creation time, the transmition time, the load state of the sender, and all the flags concerning the triggering of the update and rerouting algorithms.
Statistics	This class implements the statistic collection for each experiment, concerning the number of messages, the process duration, the load of the peers, the load of the network etc.

4.2.4 *Topology Construction*

Before starting the simulation the network needs to be constructed according to the P-Grid protocol topology, each peer must be assigned with links to other peers in order to perform searching operations in the network. Each protocol has specific algorithms which are executed when a peer wants to join the network. These algorithms in general involve at least one peer of the network, sending a join request to him, taking a part of the network data under responsibility and gradually obtaining network links to other peers so as to forward queries to other peers.

Our research focuses on the stable state of the system, the network size is given from the configuration file, the underlying topology of the network is static and is constructed at the beginning of each experiment and so the peers cannot join or leave the network arbitrarily. Although in peer to peer networks peers join and leave continuously and thus a global stable state is impossible to achieve, the assumption of a stable system provides some significant advantages when analyzing each network's performance.

First of all, the performance achieved in the stable state reflects the optimal ability of the network to retrieve answers to queries. Thus this assumption enables us to identify the upper bound associated with each network's performance. More importantly, when we compare two peer to peer systems in their stable state, we actually only compare the effectiveness of their algorithms for locating data items. If we were comparing the systems in their transient state, not only would the results be inevitably affected by the effectiveness of the algorithms for node join, node removal and node failure but would also be affected in an unforeseeable and unmeasurable way.

In our framework we used two different topology construction algorithms the Data-Balanced and the Volume-Balanced which are presented below:

Data-Balanced Algorithm: Suppose that we start from a network aparted from two peers, the total datakey space is already splitted in half and so each one of them holds half of the total amount of the datakeys. Each time that a peer wants to join the network selects uniformly at random a key from the **indexed dataset**. Then the new peer through the P-Grid trie locates the peer that holds the selected key, this peer is called mate, and asks to join the network. When this happens the mate peer splits the datakey space that he owns and shares it with the new joined peer. After that both peers , the mate and the new peer, are assigned with new pathid's in order to obtain a new position in the underlying P-Grid trie, the mate's pathid comes from his old concatenated by a 0 at the end and the pathid of the new peer comes from the mate's old pathid concatenated by a 1 at the end.As a result the mate peer becomes left child and the new peer becomes a right child of the P-Grid trie. This procedure is repeated until the network size becomes the required (which is defined in the configuration file). Thus, existing nodes are selected for splitting with probability proportional to the amount of data they store. This strategy tends to equalize the amount of data assigned to nodes, but is an open question whether it equalizes load as well.

Volume-Balanced Algorithm: Is similar in structure to the Data-Balanced algorithm with the great difference that the new peer instead of selecting a key from the **indexed dataset** selects uniformly at random a point from the **multidimensional search space**. The rest of the steps are the same with the previous algorithm. Thus, existing nodes are selected for splitting with probability proportional to the volume of their assigned space partition. This strategy tends to equalize the volume of space assigned to nodes, but possibly not the load among peers.

5 ANALYSIS

In this chapter at first we will present a description of the metrics used to evaluate our proposed protocol. Then we will present the workloads which where executed during the experimental process. We will thoroughly examine the efficiency of the proposed rerouting and update algorithms and we will present and comment the graphs with the results of our experiments.

5.1 METRICS

Before the description of the metrics which were studied we need to introduce the reader to the types of messages of our protocol.

1. **Regular Messages :** the messages which are created and forwarded during the search of a query in the network such us in the traditional P-Grid protocol.
2. **Rerouting Messages :** the messages which are created and forwarded by the rerouting algorithms.
3. **Polling Messages :** the messages which are created by the polling periodically update algorithm.
4. **Exchange Messages:** the messages which are created by the exchange update algorithm.

Due to the fact that our research is based in event driven simulation our metrics are classified in the two following types:

1. Metrics which emerge from the mean values of the observed measurements at the end of each experiment.
2. Metrics which are observed and measured during the execution of an experiment.

In the first category we observed and study the following metrics:

- **The mean response time per process**: For an experiment we measure the duration of all the completed processes and then we calculate the mean process duration which we call mean response time per process. With this metric we can observe how the additional messages created by the rerouting algorithms can influence the duration of a process.

- **The mean response time of the peers' input channel**: Each message waits an amount of time in the input channel. This time is computed from the time that a message enters the channel until the time that is serviced. We calculate the mean response time of the input channel of each peer, and then the mean response time of the input channel of all the peers in the network. The observation of this quantity can show us if our protocol can influence the response time of a channel.

- **The mean response time of the peers' output channel**: is measured the same way as in the input channel.

- **Mean number of messages per process**.

- **Total number of regular, rerouting, polling and exchange messages**. These quantities can show us the amount of the extra messages that our protocol uses in order to achieve good load balancing.

- **Total number of exact matches for each update algorithm**. As it was described in section 3.3 the update algorithms were designed in order to diffuse information about the peers load state by transmitting their load factors in their neighbors. With this metric we can measure and compare the effectiveness of the proposed update algorithms by examining the registered values which were updated by each algorithm.

- **Network Throughput** Throughput is the maximum rate of queries that a P2P network can sustain without any peer becoming overloaded.

In this category we study and observe the metrics and also their variations during the simulation time. This mean that we observe and keep their values for many different times during the simulation in order to study more thoroughly their fluctuations. These metrics are the following:

- **The number of overloaded peers in the network.** This metric shows the number of the peers in the network which are overloaded or are about to become in relation with the total number of the participant peers, for a specific simulation time. The observation of this quantity at very close times can give us a representative view considering the peers which confront increased traffic load, for each moment during the experiment execution. We calculate this number based in each peer's load factor (the peers' load states are thoroughly described in section 3.2.1). This metric reflects significantly the load in the network. Moreover by using this metric we can easily examine the efficiency of our protocol and get significant information about the network's consistency.

- **The mean utilization of the input channel**. For the calculation of this metric we observe the utilization of the input channel of all the peers in the network and then we calculate the mean value of the observed samples for a specific time during the simulation. This metric offers us the same observation abilities with the previous and can also show us the portion of the load that accumulates in the input channel of the overloaded peers.
- **The mean utilization of the output channel**. This metric is calculated as the previous. Moreover it offers us equivalent with the previous observation abilities but for the output channel.

5.2 WORKLOADS

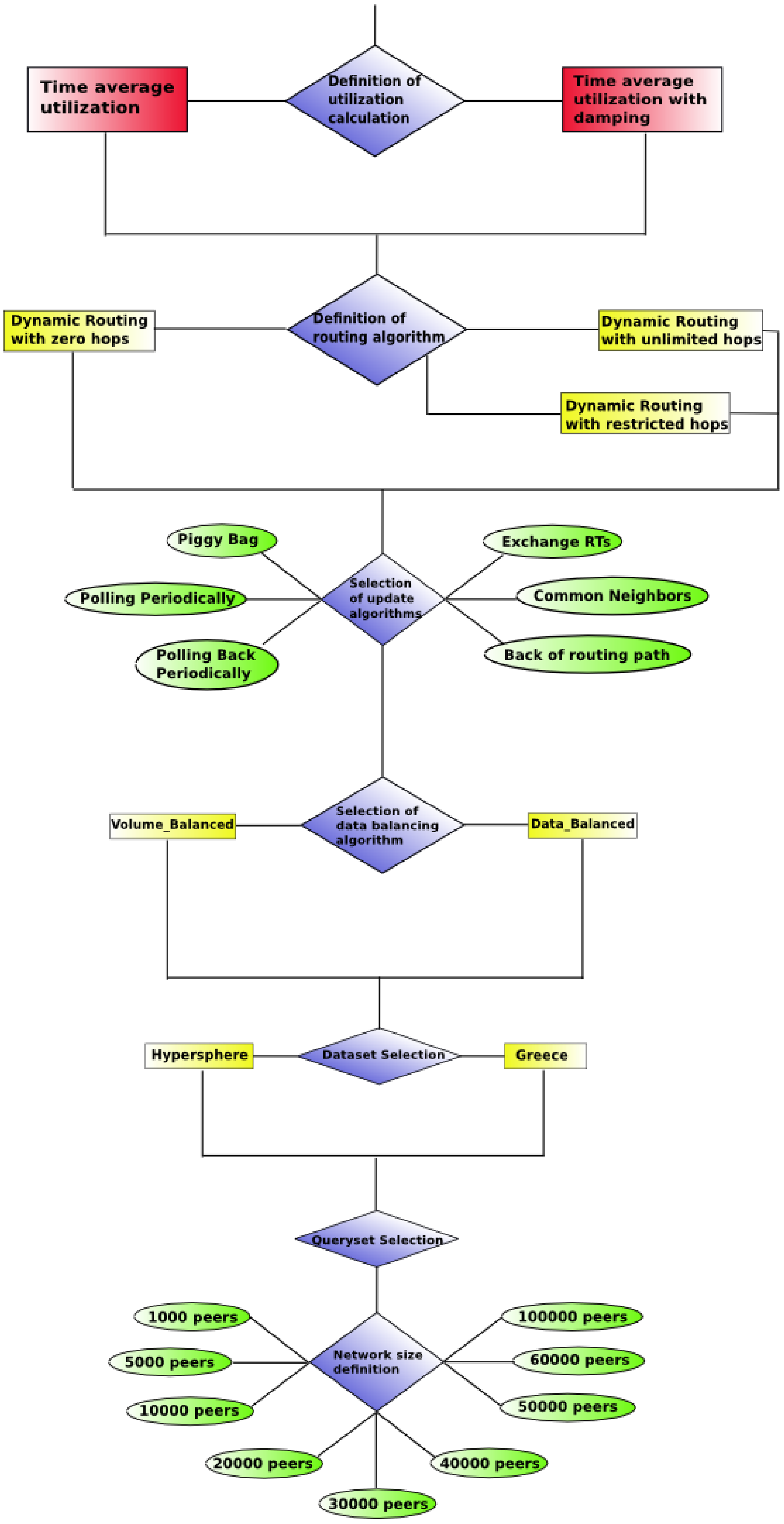


Figure 22: Workload schema.

In order to evaluate the performance and the effectiveness of our protocol it was necessary to simulate two batches of experiments. At first we simulated the traditional P-Grid protocol with all the different datasets and querysets changing simultaneously the rest of the configuration parameters such as channel's latency, network size etc. Then we simulated our protocol for the same scenarios, testing at the same time it's performance, using different combinations of our update and rerouting algorithms. In order to achieve as optimized network performance as possible we performed a wide number of experiments. Here we need to mention that we implemented also many experiments so as to obtain optimized values for some measurements which constitute network's characteristics. For example it was necessary to perform many experiments in order to have an efficient classification of a peers load state, state which will reflect satisfactory the peer's load without exaggerating or underestimating its load. Also we examined the frequency that the peers should implement an exchange request. The exchange could not be done very frequently because this would result in great additional load because of the increased amount of the created exchange messages. It could not be done rarely either because it would be inefficient. The execution of many different experiment scenarios was necessary for the definition of similar values, which during the design of the protocol were inexplicit and needed to be estimated through the experimental process in order to obtain optimized performance.

Our experiments span in the following domains:

- Parametres of the proposed protocol
    1. The two approaches of calculating a channel's load and peer's load (section 3.2).
        - Time average utilization.
        - Time average utilization with damping.
    2. The three rerouting algorithms (section 3.4).
        - Dynamic Routing.
        - Dynamic Routing with restricted number of additional hops.
        - Best Neighbor Dynamic Routing.
    3. The six update algorithms (section 3.3).
        - Piggy bag.
        - Polling Periodically.
        - Polling Periodically and back.
        - Back of routing path.
        - Requesting for common neighbors.
        - Exchange routing tables.

- Parametres of Scalability and network characteristics.
  1. Dataset.
     - We used two different datasets as they were described in section 4.1.
  2. Queryset.
     - We simulated 3 different querysets for each dataset.
  3. Data Balancing Algorithms. We used the following data balancing algorithms (see section 4.2.4).
     - Data-Balanced.
     - Volume-Balanced.
  4. The network size.
     - For each experiment we simulated **9 different network sizes 1000, 5000, 10000, 20000, 30000, 40000, 50000, 60000 and 100000 peers**.
  5. The repetition number for each experiment.
     - In order to get reliable and realistic results we run each experiment for **30 repetitions**.

Due to the fact that we have a great number of protocol and scalability parameters we executed a wide number of experiments and consequently we had a great amount of results to process. In order to avoid any confusion we present the workload schema (see figure 23) which includes all the experimental scenarios. We would also like to add that for each of the following presented cases the results and the graphs are representative of the total network's behavior.

5.3 PROTOCOL EVALUATION

In this section we will present and comment on the graphical plots which were generated from our experiments. We will see how our protocol performs in comparison with the traditional P-Grid protocol . This procedure includes the evaluation of the two methods of utilization calculation and also of the proposed rerouting and update algorithms for each method respectively. Then we will measure and comment the perfomace of the update algorithms. Finally we will evaluate the protocol's scalability according to the scalability parameters which where described in the previous section.

5.3.1 *Rerouting algorithms evaluation.*

In our protocol we introduced three rerouting algorithms Dynamic Routing, Dynamic Routing and Best Neighbor Dynamic

Routing . At first we will present the results for the time average utilization. In the figure 24 we see the percentage of overloaded peers for the traditional P-Grid protocol and our proposed protocol for the Dynamic Routing algorithm.

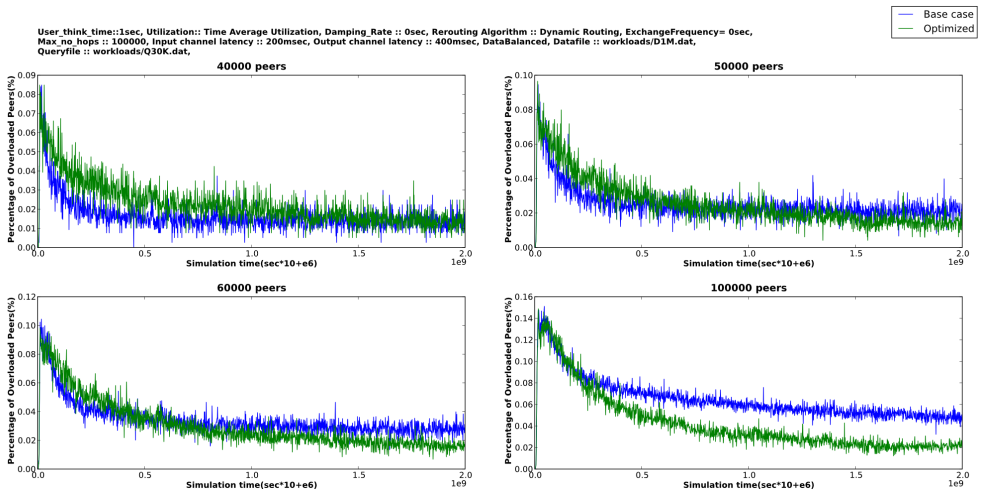


Figure 23: Number of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing algorithm . All the update algorithms are activated except from polling and exchange.

Examining the figure 24 we can observe the following

- The percentage of overloaded peers is barely higher in our protocol at the start but then converges gradually to smaller than the base case.
- The converge of our protocol to smaller percentage of overloaded peers than the base case is faster for greater network sizes.
- The final reduction in the percentage of overloaded peers is about 25 to 30 percent.

In order to study the distribution of a peer's load in the input and output channel and the way that the reduction in the number of overloaded peers is reflected in a peer's load we plotted the mean utilization for the input and output channel respectively. In figures 25 and 26 we see the mean utilization of the peers' input and output channel for the Dynamic Routing algorithm with time average utilization.

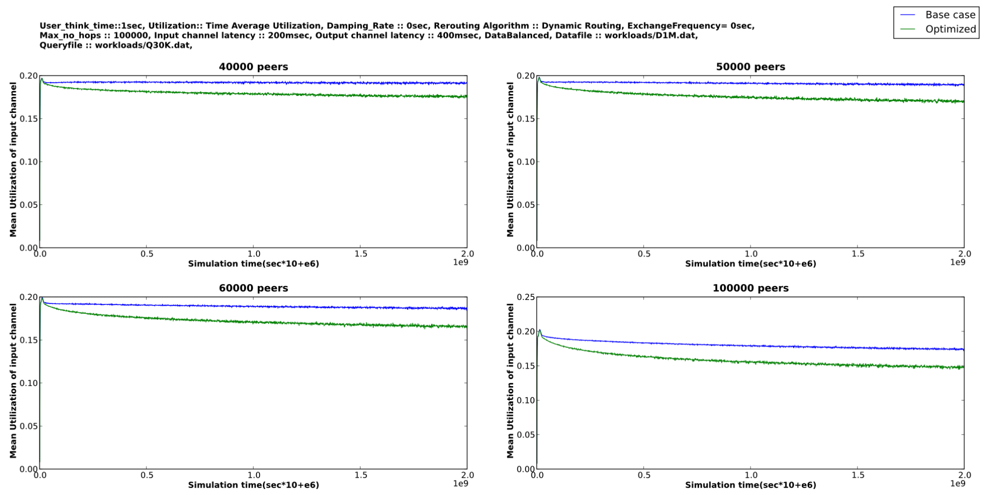


Figure 24: Mean Utilization of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing. All the update algorithms are activated except from polling and exchange.

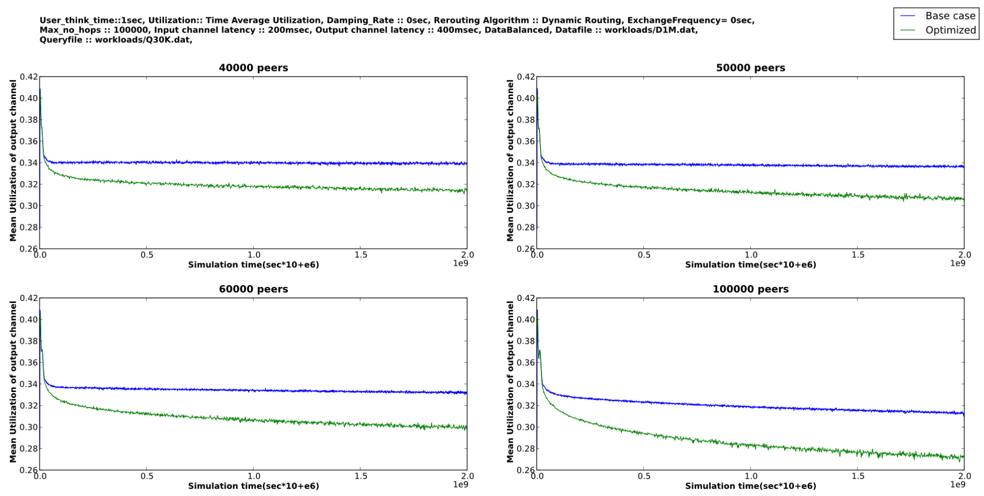


Figure 25: Mean Utilization of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing. All the update algorithms are activated except from polling and exchange.

These figures can lead us in the following conclusions:

- The mean utilization of both input and output channel is smaller for our protocol. Finally the reduction ends up to 10 to 12 percent for both channels.
- The reduction of the mean utilization for both channels increases as the percentage of overloaded peers decreases during the simulation time.

In the figures 27, 28 and 29 below we present the same metrics for the Best Neighbor routing algorithm.

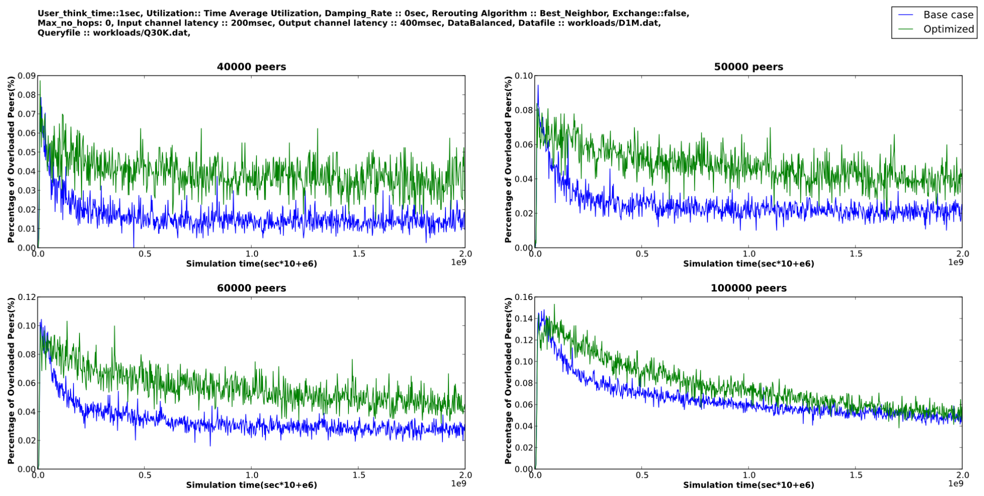


Figure 26: Number of overloaded peers for the traditional P-Grid protocol and our protocol with the Best Neighbor algorithm. All the update algorithms are activated except from polling and exchange.

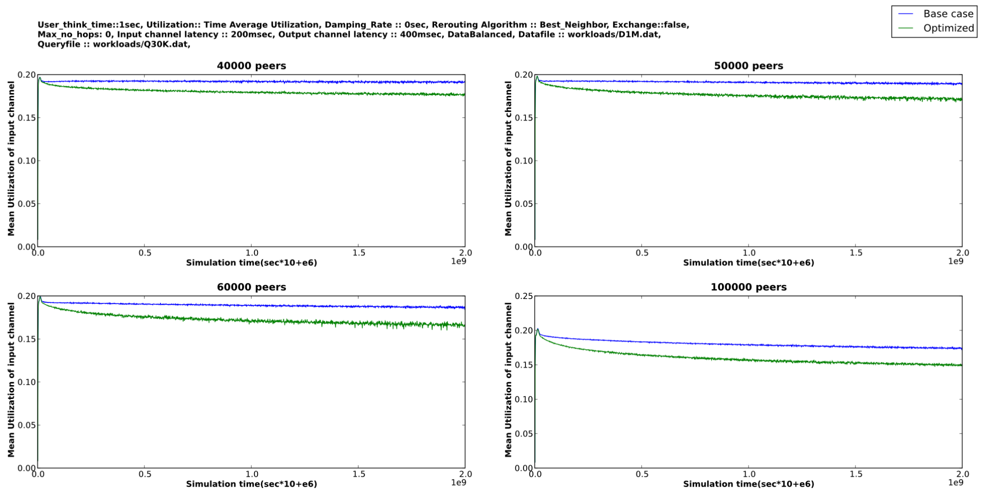


Figure 27: Mean Utilization of the input channel for the traditional P-Grid protocol and our protocol with the Best Neighbor algorithm. All the update algorithms are activated except from polling and exchange.

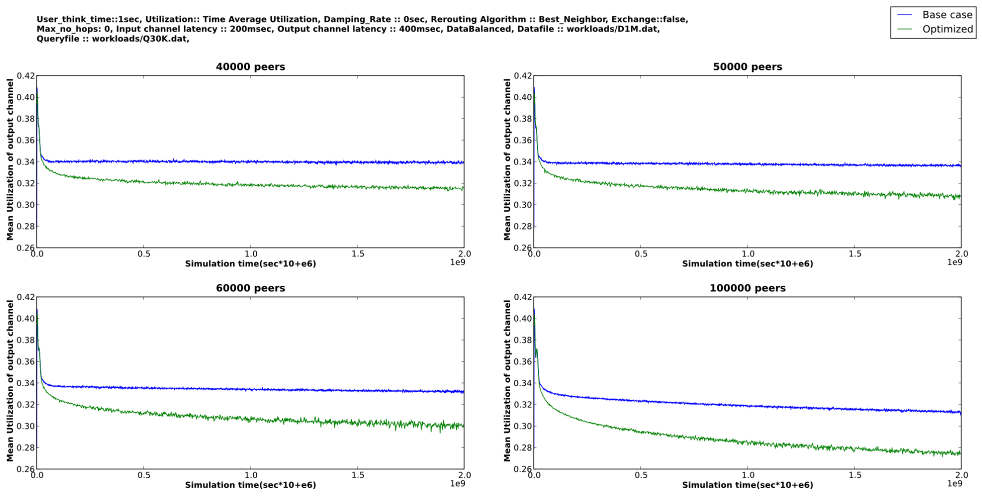


Figure 28: Mean Utilization of the output channel for the traditional P-Grid protocol and our protocol with the Best Neighbor algorithm. All the update algorithms are activated except from polling and exchange.

From the above figures we observe the following:

- The Best Neighbor algorithm converges slower than the Dynamic Routing algorithm as far as it concerns the percentage of overloaded peers.
- Despite the slower converge of the Best Neighbor algorithm the utilization of input and out channel tend to reduce gradually, a reduction which ends up to 10 percent for both channels.
- The additional messages which are created by the Dynamic Routing Algorithm contribute significantly in a faster alleviation of the overloaded peers of the network.

Here we need to mention that the Dynamic Routing with Restricted number of hops Algorithm has similar behavior with the Dynamic Routing Algorithm.

In order to study how the rerouting algorithms impact to the network's response time we need to examine some more metrics. These metrics are the mean process response time, the mean input and output channel response time and the throughput of the network. Figures 30 and 31 show these metrics for the Dynamic Routing algorithm in comparison with the base case.

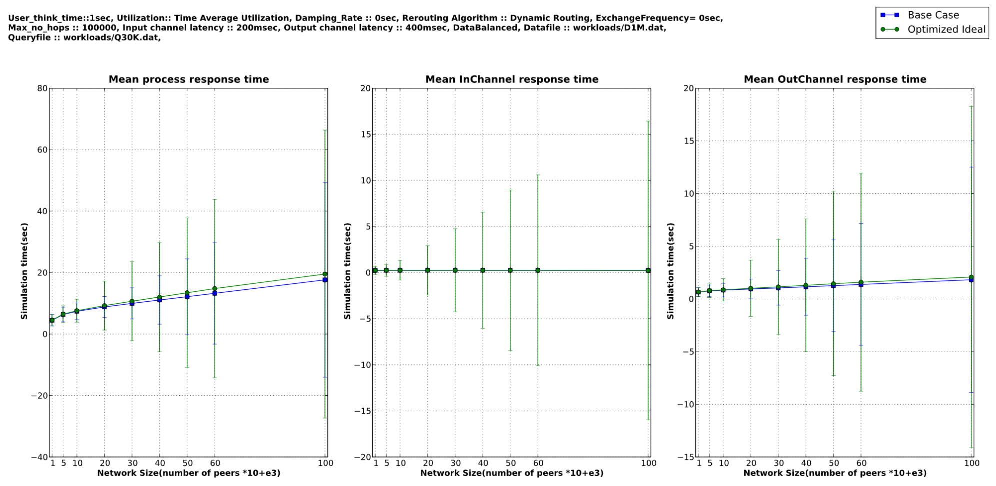

User_think_time::1sec, Utilization:: Time Average Utilization, Damping_Rate :: 0sec, Rerouting Algorithm :: Dynamic Routing, ExchangeFrequency= 0sec,
Max_no_hops :: 100000, Input channel latency :: 200msec, Output channel latency :: 400msec, DataBalanced, Datafile :: workloads/D1M.dat,
Queryfile :: workloads/Q30K.dat,

Figure 29: Process, input and output channel's response time for the traditional P-Grid protocol and our protocol with the Dynamic Routing . All the update algorithms are activated except from polling and exchange.

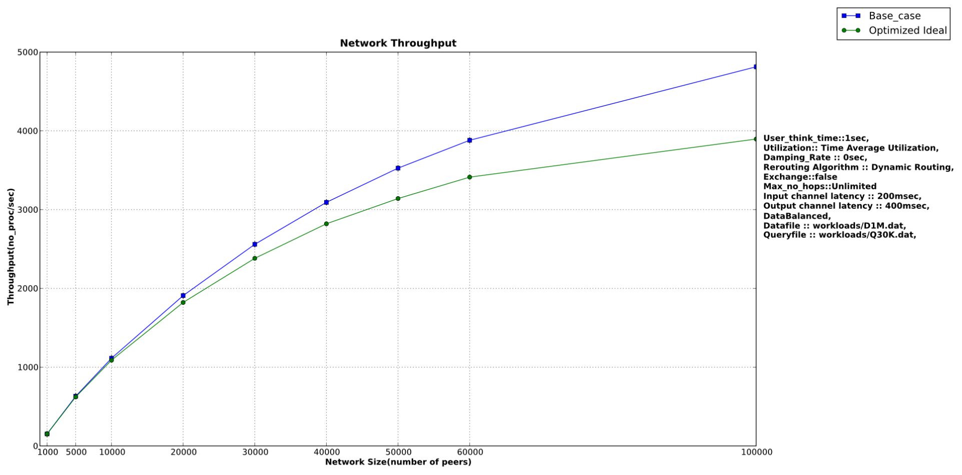
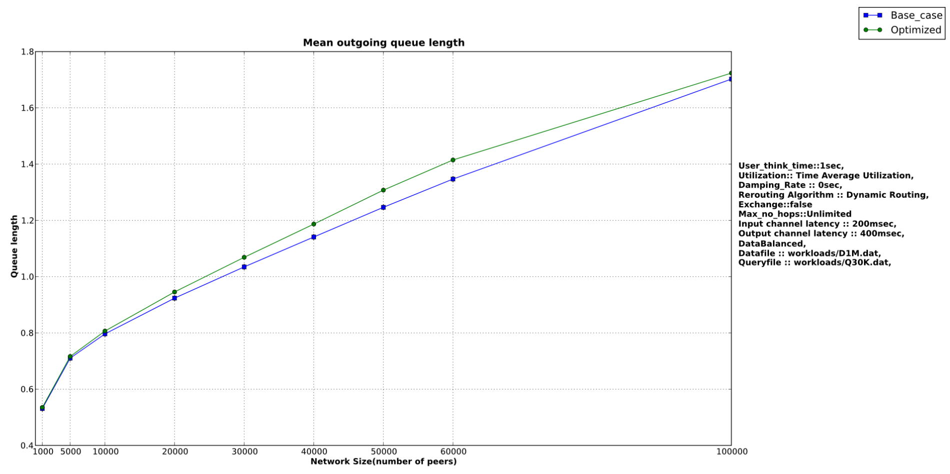

User_think_time::1sec,
Utilization:: Time Average Utilization,
Damping_Rate :: 0sec,
Rerouting Algorithm :: Dynamic Routing,
Exchange::false
Max_no_hops::Unlimited
Input channel latency :: 200msec,
Output channel latency :: 400msec,
DataBalanced,
Datafile :: workloads/D1M.dat,
Queryfile :: workloads/Q30K.dat,

Figure 30: Network Throughput for the traditional P-Grid protocol and our protocol with the Dynamic Routing . All the update algorithms are activated except from polling and exchange.

We observe that in our case we have a negligible increase of the mean process duration and also of the output channel response time while the response time of the input channel remains the same. This means that the rerouting deteriorates scarcely the network's response time and that is the reason why we observe a small reduction in the network's throughput. Also we need to notice that the results for the Dynamic Routing with restricted number of hops and the Best Neighbor Routing are equivalent to the presented case, so the increase in the network's response time can be partially attributed to the additional rerouting messages which increase the process response time. We might expect that the reduction in the mean utilization of the input and the output channel would lead in a equivalent reduction in their response time. Here we need to notice that the amplitude of this reduction could only lead in negligible reduction in the channel's response time, we need to achieve much greater reduction in order to have obvious results in the channels' response time. Also for the output channel we observed that despite the reduction of the mean utilization, we have a small increase in the channel's response time. In order to understand this behavior we need to observe figure 32.



User_think_time::1sec,
Utilization:: Time Average Utilization,
Damping_Rate :: 0sec,
Rerouting Algorithm :: Dynamic Routing,
Exchange::false
Max_no_hops::Unlimited
Input channel latency :: 200msec,
Output channel latency :: 400msec,
DataBalanced,
Datafile :: workloads/D1M.dat,
Queryfile :: workloads/Q30K.dat,

Figure 31: Mean output channel's queue length for the traditional P-Grid protocol and our protocol with the Dynamic Routing. All the update algorithms are activated except from polling and exchange.

In figure 32 we see the mean length of the output channel for the base and the optimized case. We observe that the mean length of the output queue is negligibly but steadily increased, this means that each message needs to wait for greater time in the output channel in order to be served. This fact leads in the increase of the output channel's response time. Our algorithms try to alleviate the load in the overloaded peers so the additional load is distributed to underloaded peers. The previously underloaded peers remain underloaded but their total load is increased due to the additional messages that they accept because of the rerouting. We should also explain how a reduction in the utilization of a channel can lead in the increase of its queue length and moreover to the deterioration of its response time.

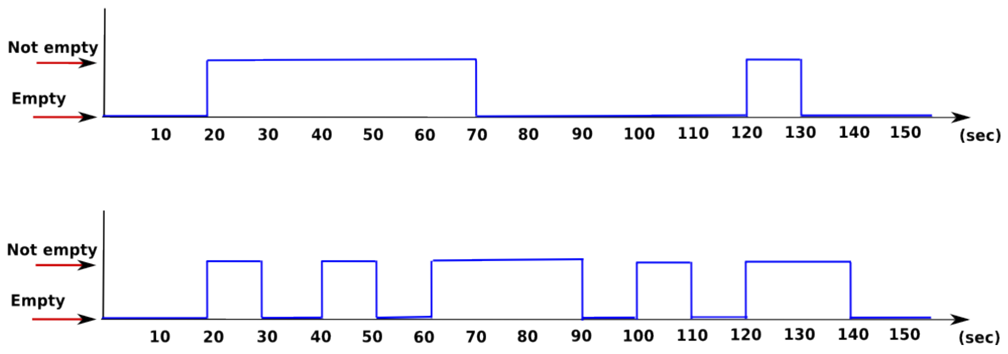


Figure 32: Utilization of two different example channels.

If we observe figure 33 we see the utilization of two different channels respectively. We see that for the first channel we have

$$U = \frac{BusyTime}{ObservationTime} = \frac{60sec}{150sec} = 0.4$$

and for the second channel we have

$$U = \frac{BusyTime}{ObservationTime} = \frac{80sec}{150sec} = 0.53$$

Also we observe that the first channel is busy for greater amount of continuous time than the second but despite that the first channel has smaller utilization. This continuous busy time of the queue leads in the increase of the queue length and moreover in the increase of the channel's response time without any increase in the channel's utilization. So we conclude that a reduction in the channel's utilization does not necessarily leads in an response time reduction. The response time has to do mostly with the arrival rate of the messages.

Now we will present the results for the second way of the utilization calculation, the time average with damping, for the same with the above workload in order to compare the two calculation methods. In the following figures 34, 35, 36, 37 and 38 we see the percentage of the overloaded peers, the mean utilization of input and output channel, the response time of the system and the network's throughput for the traditional P-Grid and our protocol.

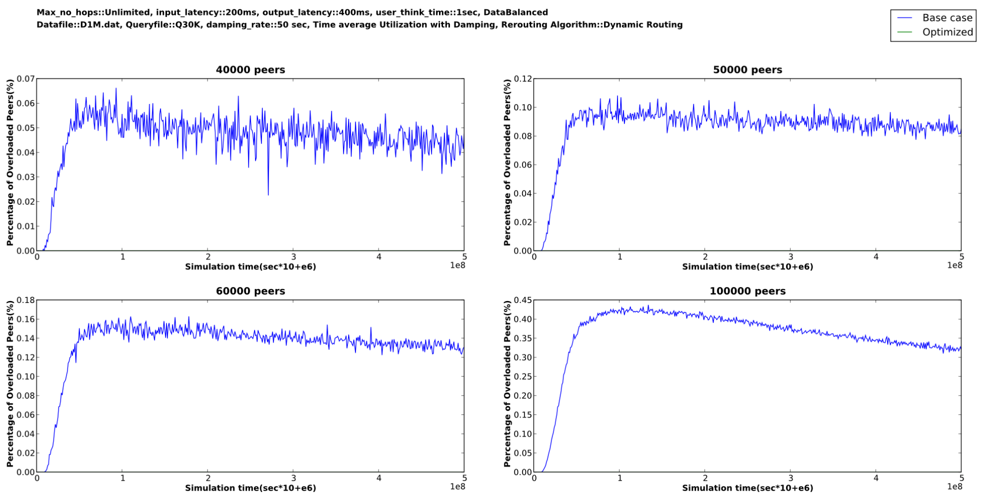


Figure 33: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization. All the update algorithms are activated except from polling and exchange. Damping rate 50 s.

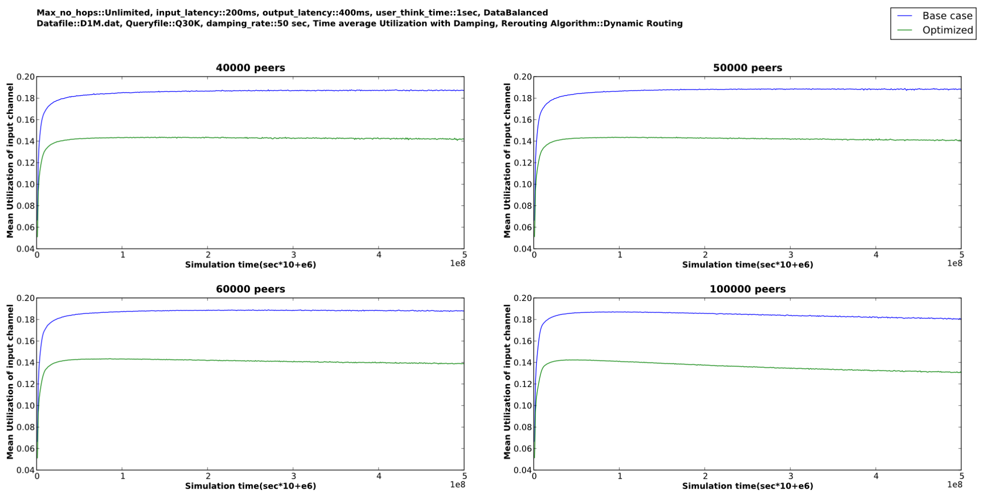


Figure 34: Mean utilization with damping of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing. All the update algorithms are activated except from polling and exchange. Damping rate 50 s.

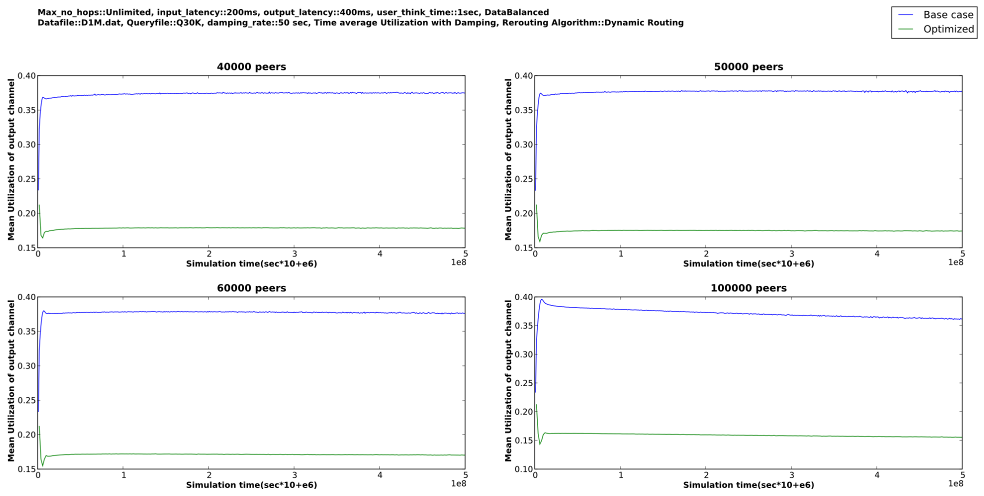


Figure 35: Mean utilization with damping of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing. All the update algorithms are activated except from polling and exchange. Damping rate 50 s.

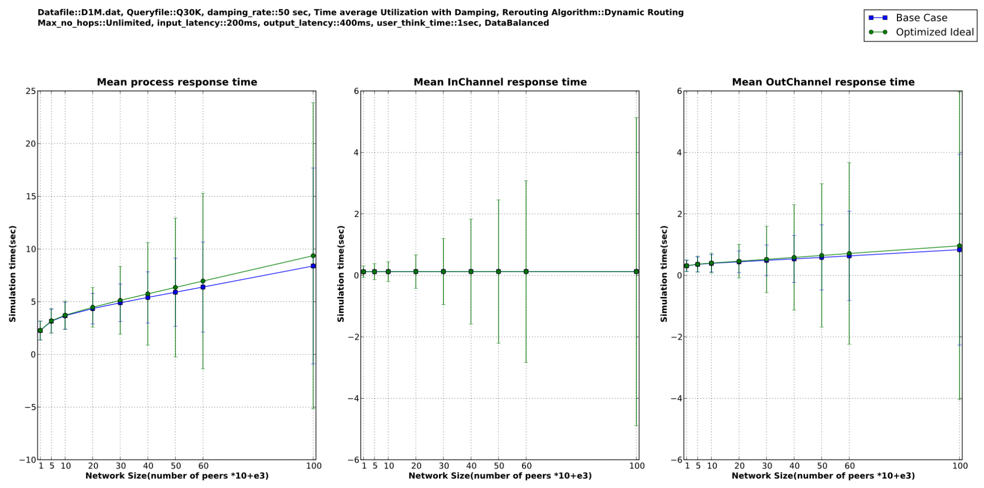


Figure 36: Process, input and output channel's response time for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization. All the update algorithms are activated except from polling and exchange. Damping rate 50 s.

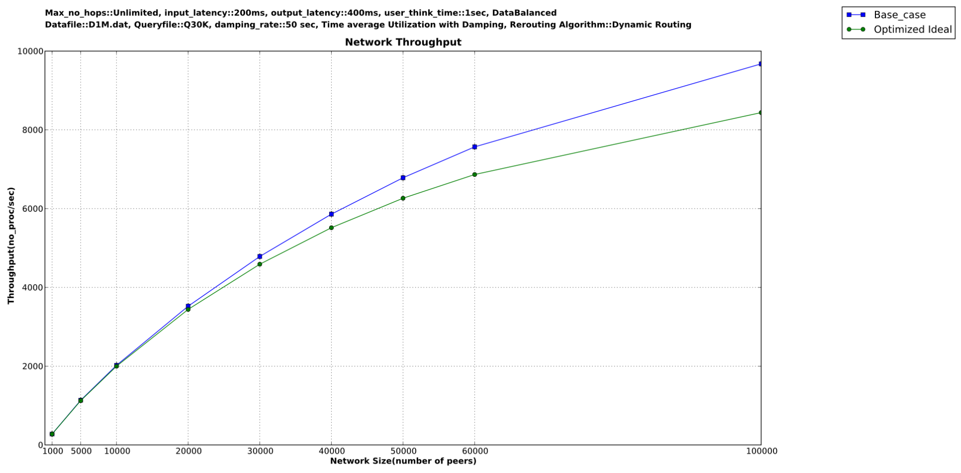


Figure 37: Network Throughput for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization. All the update algorithms are activated except from polling and exchange. Damping rate 50 s.

From the above figures we can conclude:

- The calculation of utilization with damping which assigns greater weights in the most recent utilization values shows us that the utilization is reduced significantly as the simulation time elapses. This means that with the time average method we have relatively high values of utilization at the start of the simulation which need time in order to depreciate. So we need to observe many small values of utilization in order to eliminate the effect of the first observed high utilization values and that is the reason why with the time average method the system needs more time in order to perform better than the base case.

- **The utilization of the input channel is reduced 20 to 25 percent** while **the utilization of the output channel is reduced 50 to 60 percent**.

- **The number of overloaded peers tends to zero**, fact which indicates a reduction of 100 percent for the specific workload where we have a small number of overloaded peers in the base case.

- The reduction of the above metrics tends to stabilize from the beginning of the simulation for the utilization with damping while in the time average utilization needs greater time in order to converge.

- The second method has better behavior as far as it concerns the percentage of overloaded peers but despite that, we observe a small deterioration in the network's response time, fact which leads in an equivalent reduction in the network's throughput.

- We observe zero number of overloaded peers, fact which shows us that the Dynamic routing algorithm was not triggered and the optimization at the system's performance is attributed only to the Best Neighbor rerouting algorithm which does not inquires additional rerouting messages. By this observation we can conclude that in our case where we have 2 dimensional Routing Tables there always is an underloaded neighbor at which we can forward the message regularly when the network has a medium total load. This indicates that only the existence of a variable which holds information of a peer's load state, without the additional rerouting messages, can easily lead in the creation of alternative routing paths which do not burden the already overloaded peers of the network. Here we need to mention that in the base case we have also 2 dimensional Routing Tables where the selection of the destination peers is random among the corresponding level registrations.

Also we should mention that we executed experiments for the time average utilization with damping, with damping equal to 5 seconds and 100 seconds respectively and we got similar results with the presented where we have damping equal to 50 seconds.

In order to evaluate the performance of the other two rerouting algorithms we need to examine them in heavier workloads. In the figures 39, 40, 41, 42, 43, 44, 45 and 46 we present the percentage of overloaded peers and the mean utilization of input and output channel for a heavier workload with damping rate 50s.

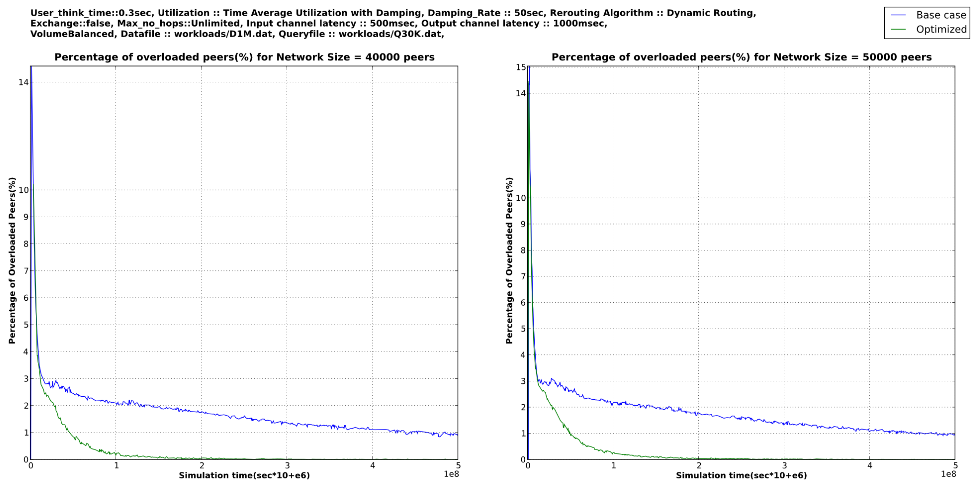


Figure 38: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange. The damping rate is 50s.

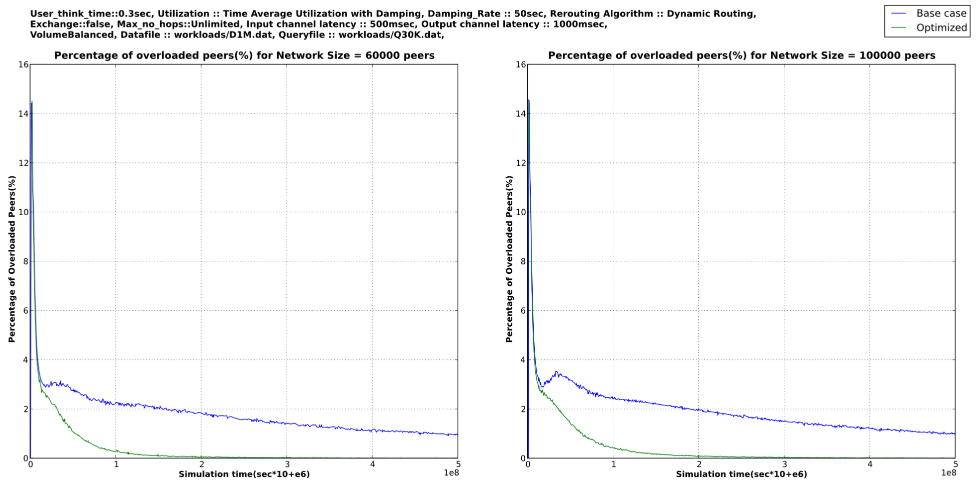


Figure 39: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange. The damping rate is 50s.

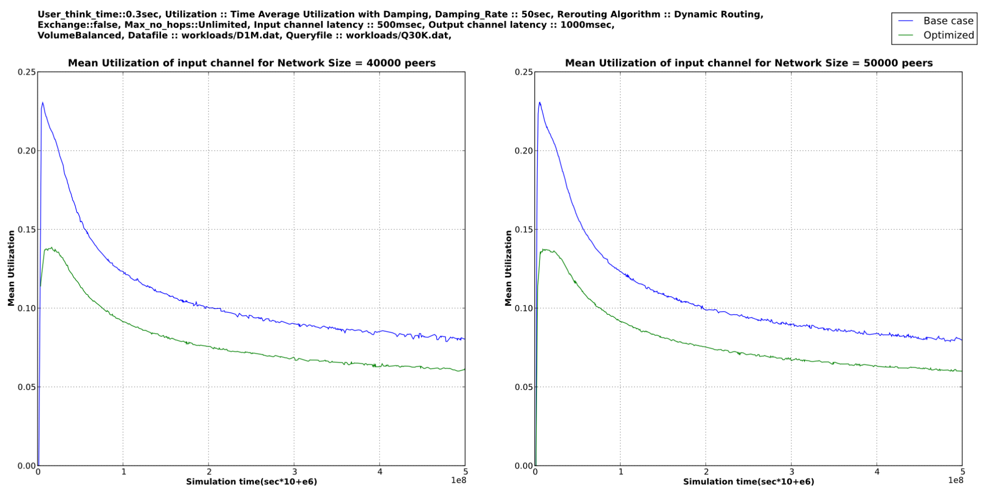


Figure 40: Mean utilization with damping 50s of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange.

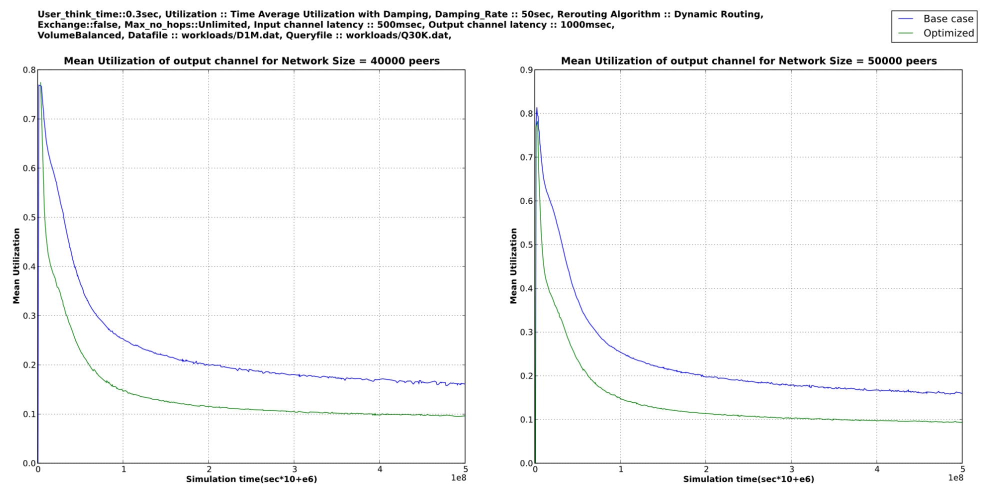


Figure 41: Mean utilization with damping 50s of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange.

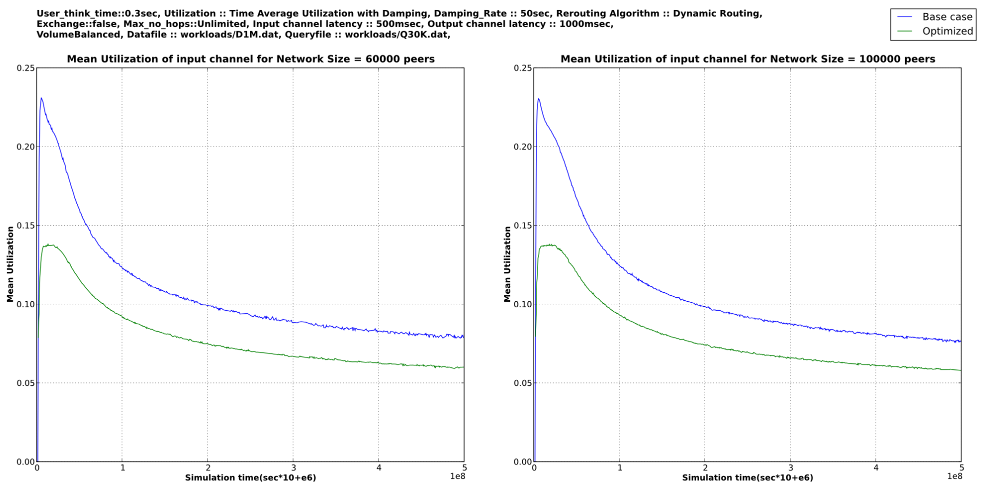


Figure 42: Mean utilization with damping 50s of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange.

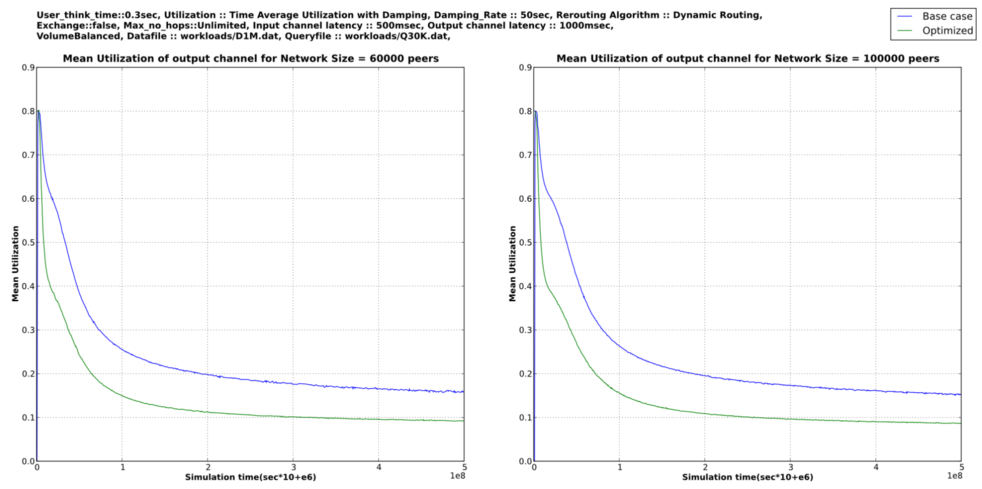


Figure 43: Mean utilization with damping 50s of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange.

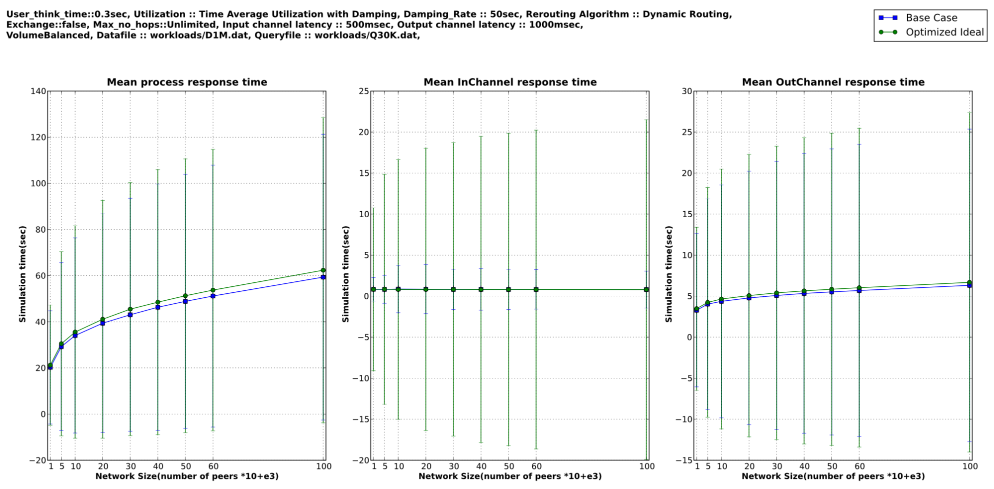

User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing, Exchange::false, Max_no_hops::Unlimited, Input channel latency :: 500msec, Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q30K.dat,

Figure 44: Process, input and output channel's response time for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization. All the update algorithms are activated except from polling and exchange. The damping rate is 50s.

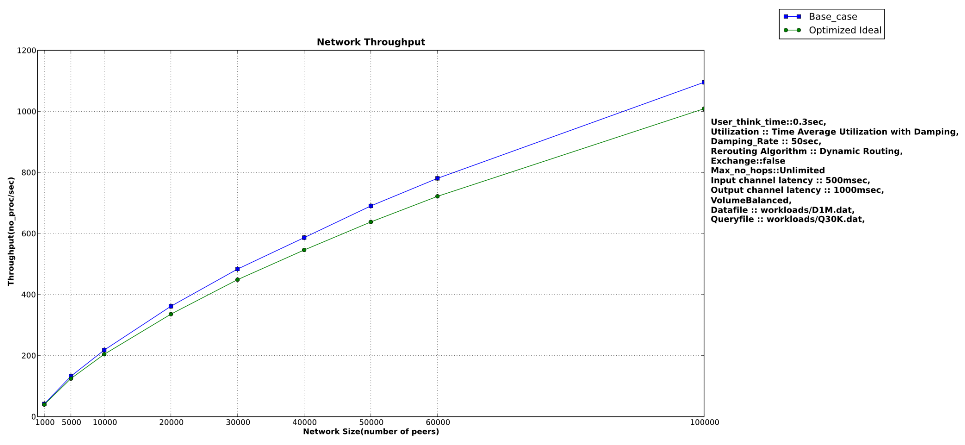

User_think_time::0.3sec,
Utilization :: Time Average Utilization with Damping,
Damping_Rate :: 50sec,
Rerouting Algorithm :: Dynamic Routing,
Exchange::false
Max_no_hops::Unlimited
Input channel latency :: 500msec,
Output channel latency :: 1000msec,
VolumeBalanced,
Datafile :: workloads/D1M.dat,
Queryfile :: workloads/Q30K.dat,

Figure 45: Network Throughput for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization. All the update algorithms are activated except from polling and exchange.

Looking the above figures we cocclude the following:

- Our protocol is by far better than the base case. The number of overloaded peers is significantly reduced **more than 90 percent**.
- Our protocol handles the network load better and manages to reduce the percentage of overloaded peers to almost 0 percent, while the same time the base case varies between 1 and 2 percent. So we can say that our protocol performs

the same well and for quite heavy workloads as far as it concerns the percentage of overloaded peers.

- At the beginning we have high mean utilizations in both channels, proportional to the great number of overloaded peers for the corresponding simulation time. Then the means starts to reduce gradually and if we observe the last seconds of the simulation time we will see that we reach a stable reduction of 50 percent for the mean utilization of the input and the output channel.

- Also we observe a small increase in the output channel's response time which with the additional rerouting messages leads to a small increase in the process response time. The increase of the process response time leads in a small reduction in the network's throughput as it was expected.

- So we can say that in general terms our protocol performs the same well and for much heavier than the previous workloads.

We should mention that we tested many heavy workloads with damping rates 5s, 50s and 100s for all of the Rerouting algorithms. In all the cases we got almost the same good results and we conclude that all the rerouting algorithms have equivalent performance for heavier workloads.

We would also present some interesting results which emerge from specific workloads. In figures 47,48,49,50,51 and 52 we see our protocol with and without the exchange method in comparison with the traditional P-Grid protocol. We observe that the results remain the same either with or without exchange. This means that the exchange update algorithm does not improves further our protocol. Lastly we should mention that the exchange algorithm is the only update algorithm which can change a registration in a peer's Routing Table, so offers updating in the registered neighbors except from updating in the neighbors' load factors. This unique characteristic of the exchange algorithm must be studied further in order to see the impact in the network's behavior.

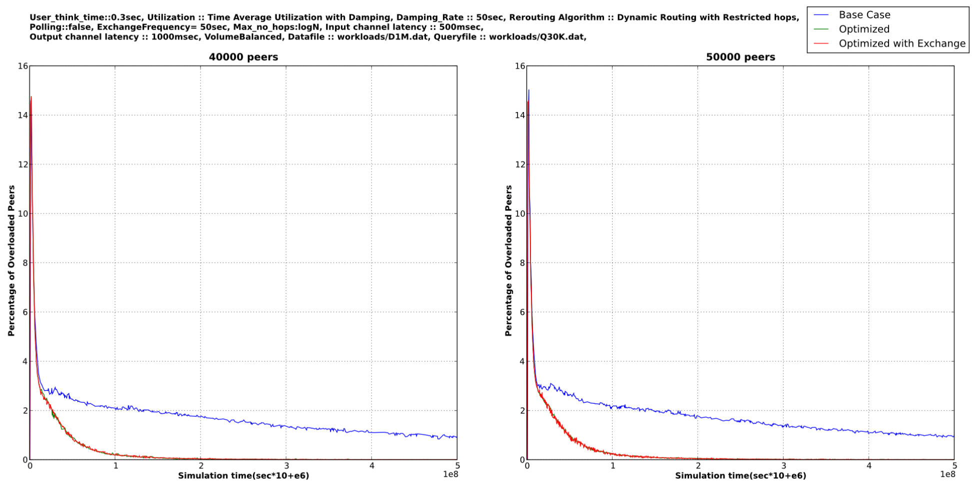


Figure 46: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with exchange, for network sizes 40000 and 50000 peers. Exchange frequency 50s.

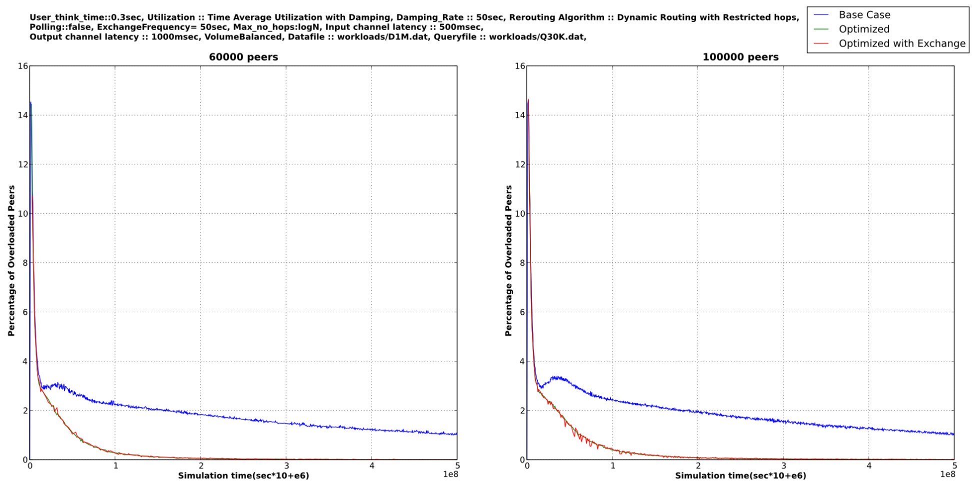


Figure 47: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with exchange, for network sizes 60000 and 100000 peers. Exchange frequency 50s.

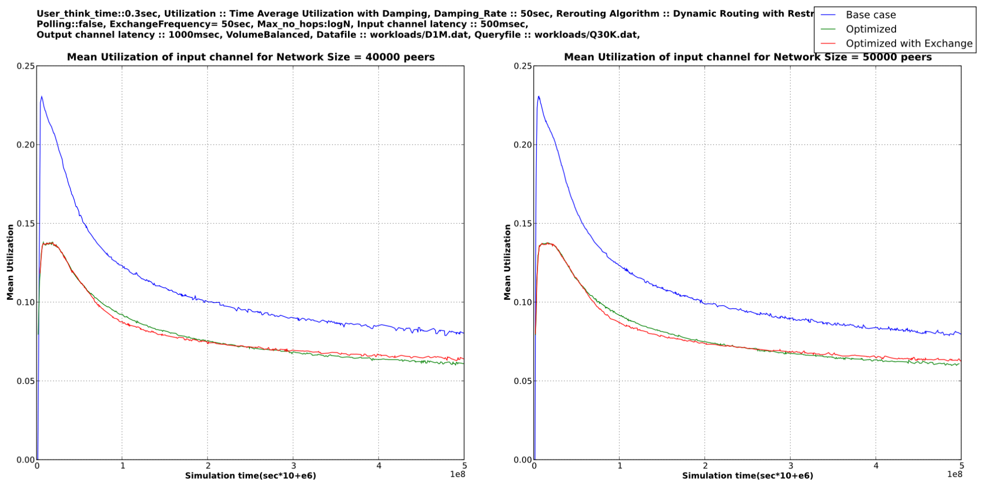


Figure 48: Mean Utilization of the input channel for the traditional P-Grid protocol and our protocol with exchange, for network sizes 40000 and 50000 peers. Exchange frequency 50s.

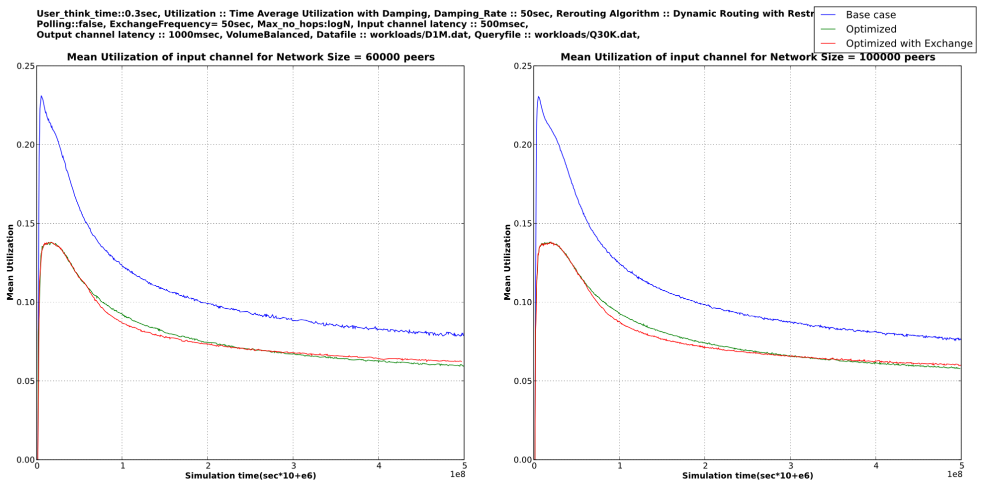


Figure 49: Mean Utilization of the input channel for the traditional P-Grid protocol and our protocol with exchange, for network sizes 60000 and 100000 peers. Exchange frequency 50s.

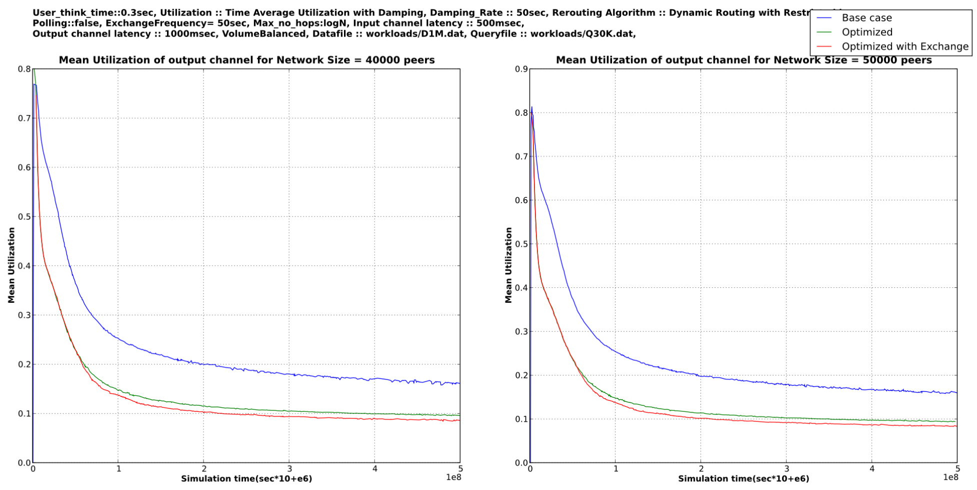

User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing with Restr
Polling::false, ExchangeFrequency= 50sec, Max_no_hops:logN, Input channel latency :: 500msec,
Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q30K.dat,

Figure 50: Mean Utilization of the output channel for the traditional P-Grid protocol and our protocol with exchange, for network sizes 40000 and 50000 peers. Exchange frequency 50s.

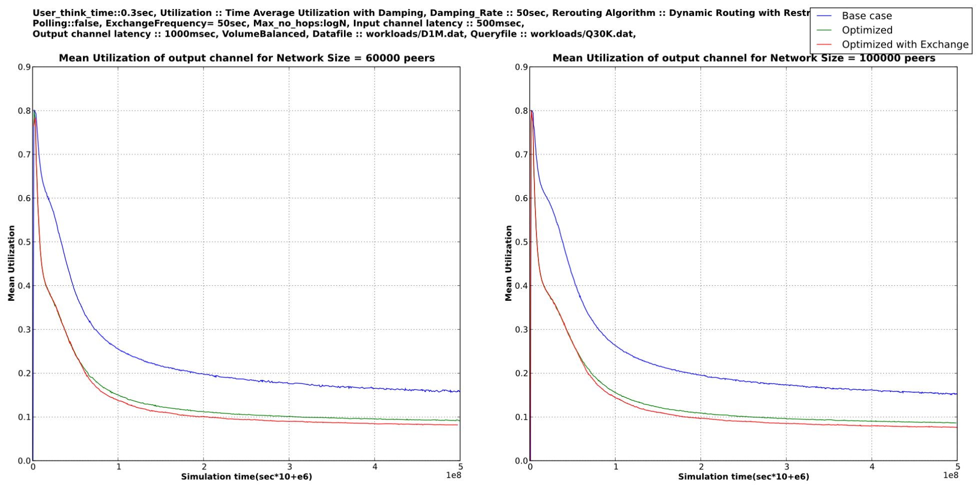

User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing with Restr
Polling::false, ExchangeFrequency= 50sec, Max_no_hops:logN, Input channel latency :: 500msec,
Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q30K.dat,

Figure 51: Mean Utilization of the output channel for the traditional P-Grid protocol and our protocol with exchange, for network sizes 60000 and 100000 peers. Exchange frequency 50s.

If we see the figure 53 and 54 we observe that the mean process response time and the mean output channel response time increase and as a result the maximum throughput of the network decreases. If we see figure 55 we conclude that the exchange method creates extremely high number of additional messages about 1/13 of the total messages. This amount of messages does not increase the load of a peer because the exchange takes place between two peers if both are underloaded, but increases the peers' output channel response time. Moreover the process duration is increased because the exchanges are executed as part of the already created processes.

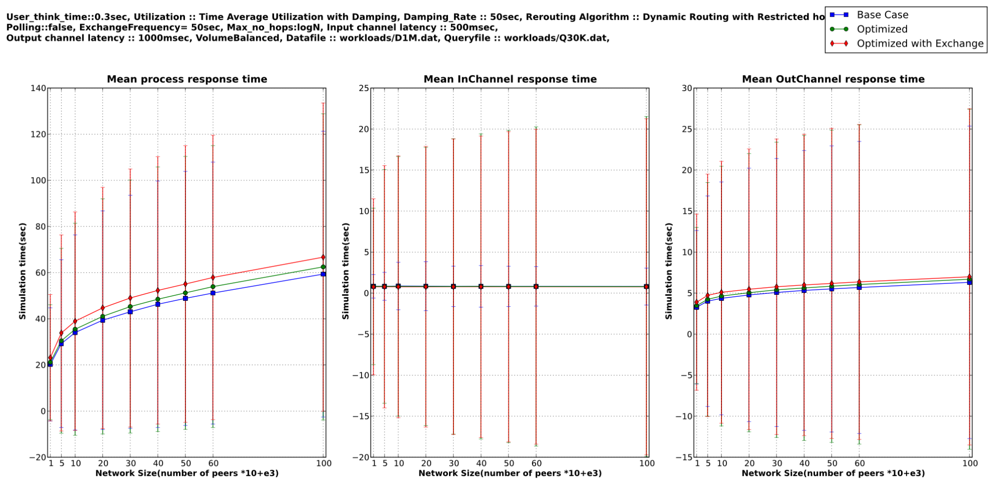


Figure 52: Process, input and output channel's response time for the traditional P-Grid protocol and our protocol with the exchange update algorithm. Exchange frequency 50s.

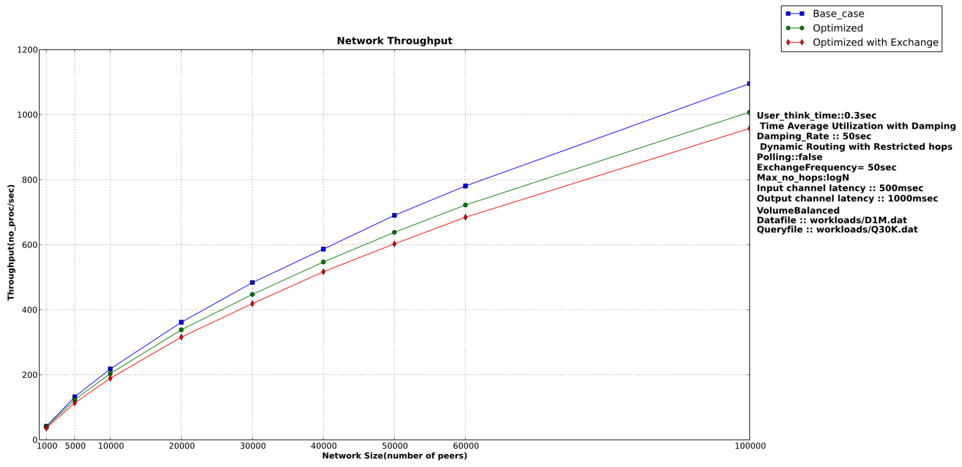


Figure 53: Network Throughput for the traditional P-Grid protocol and our protocol with the exchange update algorithm. Exchange frequency 50s.

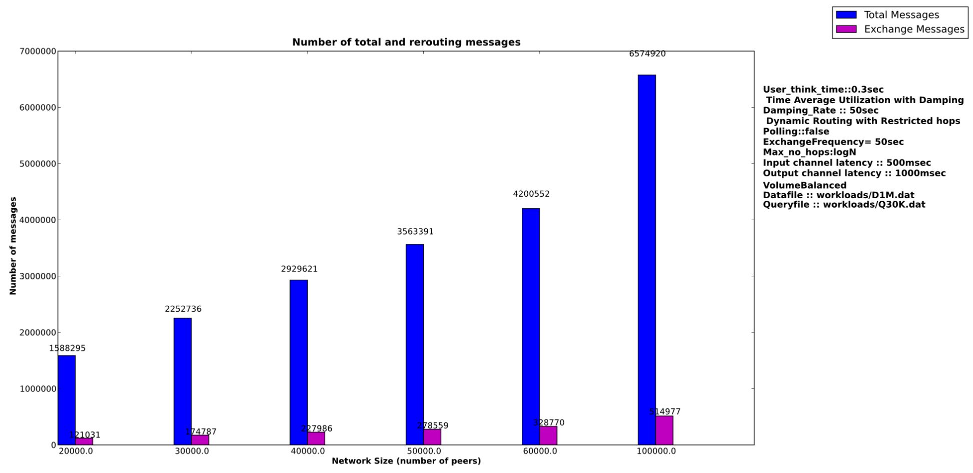


Figure 54: Number of exchange messages in comparison with the total created messages. Exchange frequency 50s.

In figures 56,57,58,59,60,61 we see the network's performance with the polling update algorithm. Despite the fact that the percentage of overloaded peers remains in the same level we have a negligible but peculiar increase in the mean utilization of the input channel, after the time that the polling starts (polling frequency = 50s), while the mean utilization of the output channel remains almost the same.

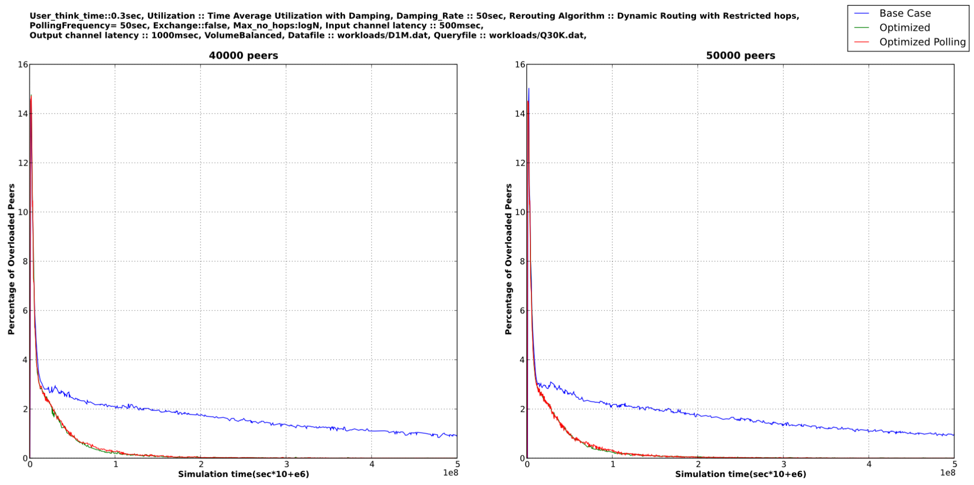


Figure 55: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with polling, for network sizes 40000 and 50000 peers. polling frequency 50s.

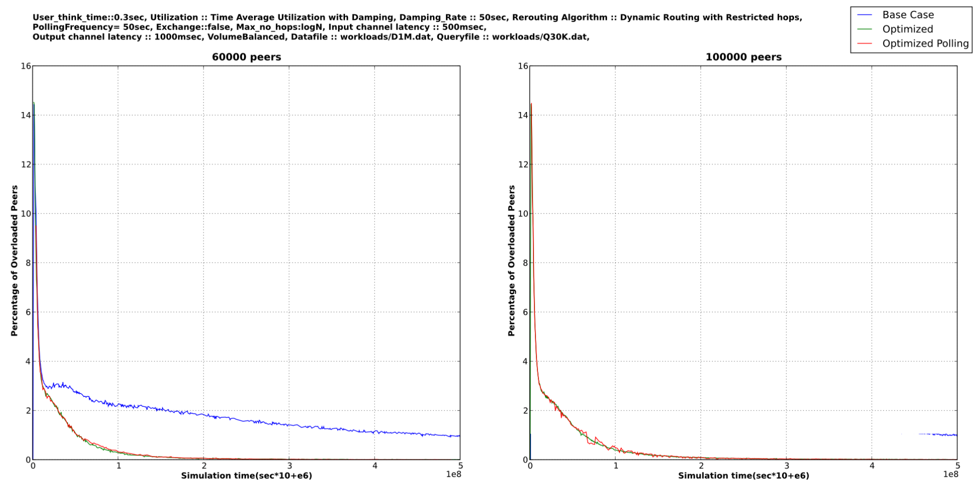


Figure 56: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with polling, for network sizes 60000 and 100000 peers. polling frequency 50s.

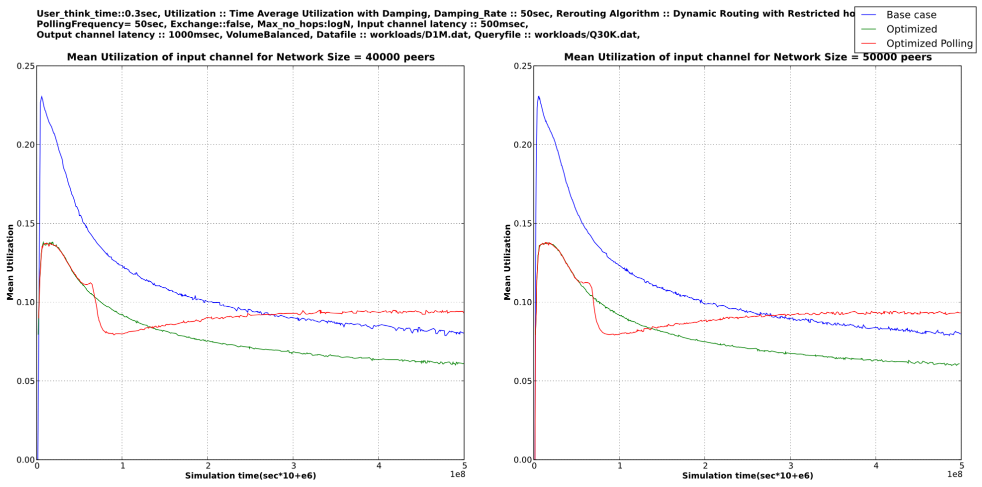


Figure 57: Mean Utilization of the input channel for the traditional P-Grid protocol and our protocol with polling, for network sizes 40000 and 50000 peers. polling frequency 50s.

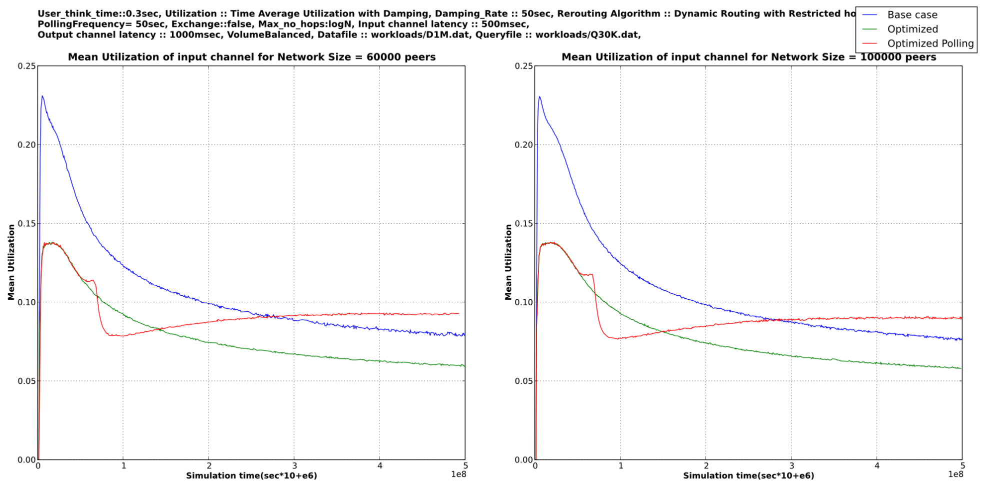


Figure 58: Mean Utilization of the input channel for the traditional P-Grid protocol and our protocol with polling, for network sizes 60000 and 100000 peers. polling frequency 50s.

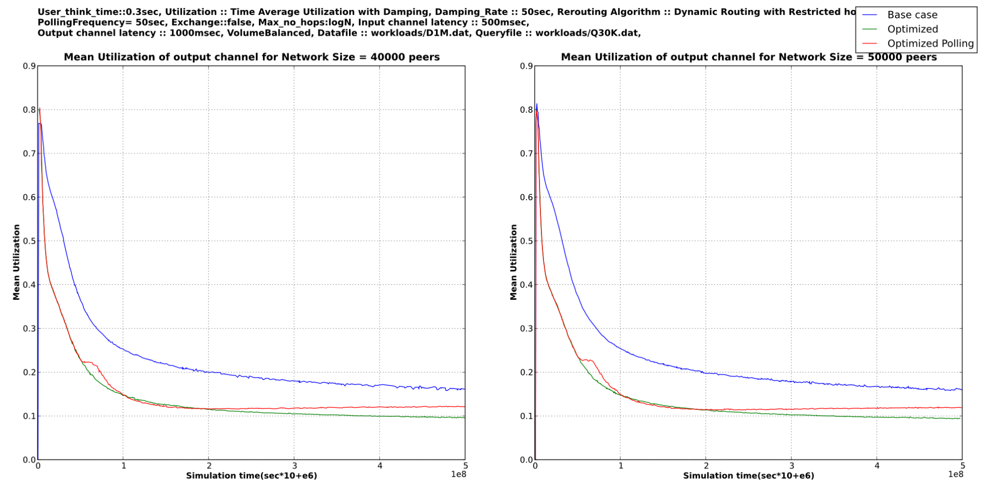


Figure 59: Mean Utilization of the output channel for the traditional P-Grid protocol and our protocol with polling, for network sizes 40000 and 50000 peers. polling frequency 50s.

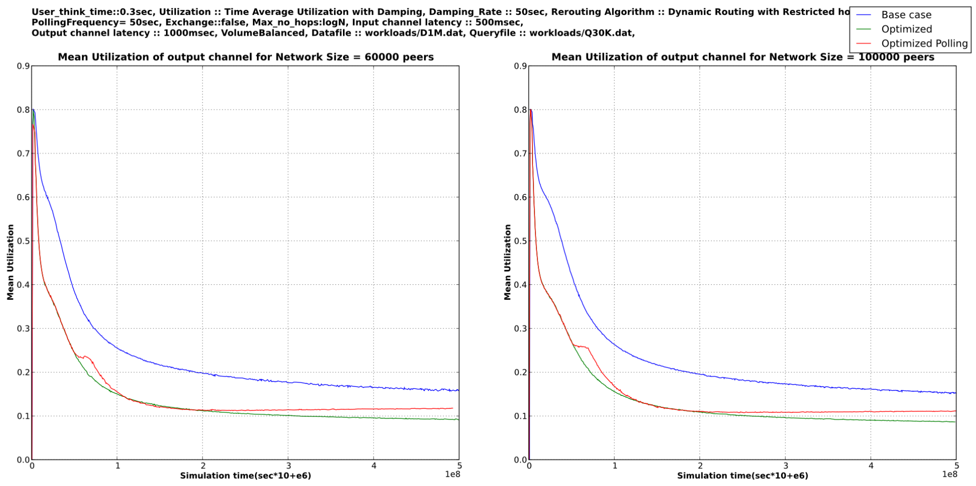


Figure 60: Mean Utilization of the output channel for the traditional P-Grid protocol and our protocol with polling, for network sizes 60000 and 100000 peers. polling frequency 50s.

Also in figures 62 and 63 that we see a great increase in the network's response time which leads in an equivalent reduction in the network's throughput.As we see in figure 64 this increase is attributed to extremely high number of polling messages,the polling messages are about the half of the total messages. So we should examine the algorithm's efficiency in order to see if the trade off in the network's response time worths.

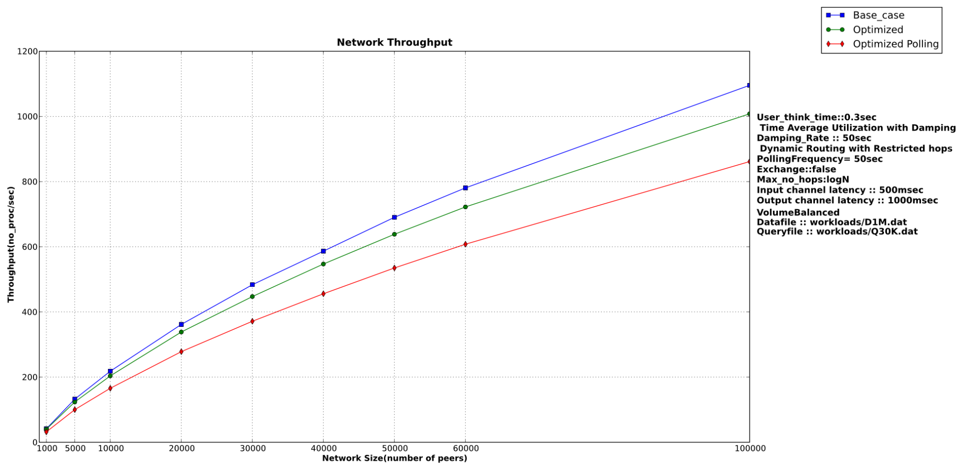
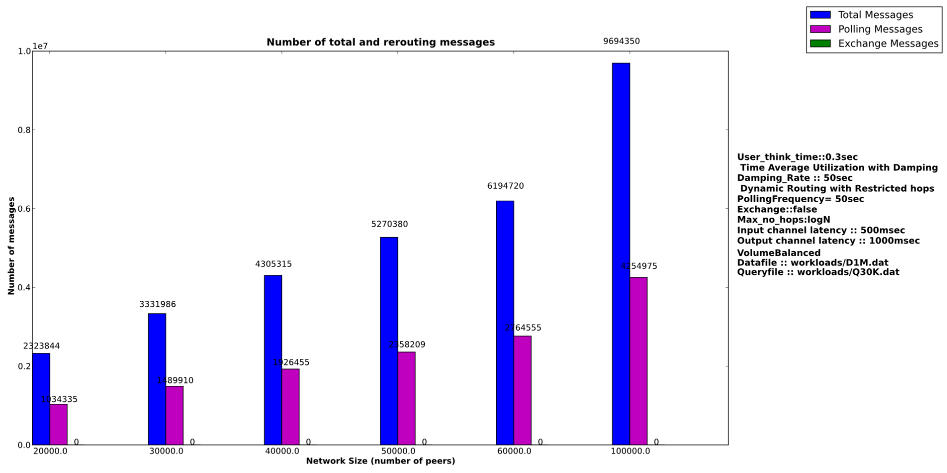


Figure 61: Network Throughput for the traditional P-Grid protocol and our protocol with the polling update algorithm. polling frequency 50s.

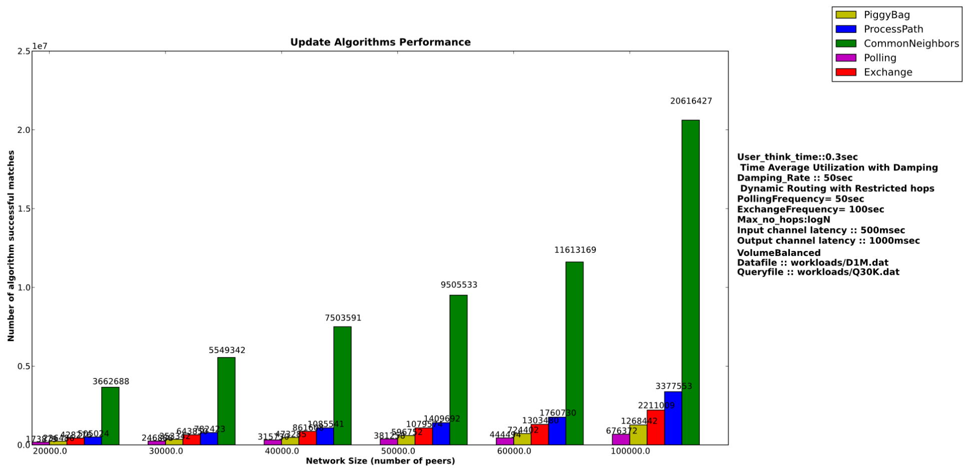


User_think_time::0.3sec
Time Average Utilization with Damping
Damping_Rate :: 50sec
Dynamic Routing with Restricted hops
PollingFrequency= 50sec
Exchange::false
Max_no_hops:logN
Input channel latency :: 500msec
Output channel latency :: 1000msec
VolumeBalanced
Datafile :: workloads/D1M.dat
Queryfile :: workloads/Q30K.dat

Figure 62: Number of polling messages in comparison with the total created messages. polling frequency 50s.

5.3.2 *Update Algorithms Evaluation*

In this section we will observe and study the performance of the proposed update algorithms. In figure 64 and 65 we see the number of updates that each update algorithm performed for workloads with medium and heavy load respectively. The percentage of the updates that each update algorithm performs is almost the same in all the tested workloads, so we present only one in order to compare the algorithms' performance.



User_think_time::0.3sec
Time Average Utilization with Damping
Damping_Rate :: 50sec
Dynamic Routing with Restricted hops
PollingFrequency= 50sec
ExchangeFrequency= 100sec
Max_no_hops:logN
Input channel latency :: 500msec
Output channel latency :: 1000msec
VolumeBalanced
Datafile :: workloads/D1M.dat
Queryfile :: workloads/Q30K.dat

Figure 63: Performance of the update algorithms for a workload with medium load. In the specific workload we have Dynamic Routing with restricted hops, all update algorithms activated with polling and exchange frequency 50 s and 100 s respectively.

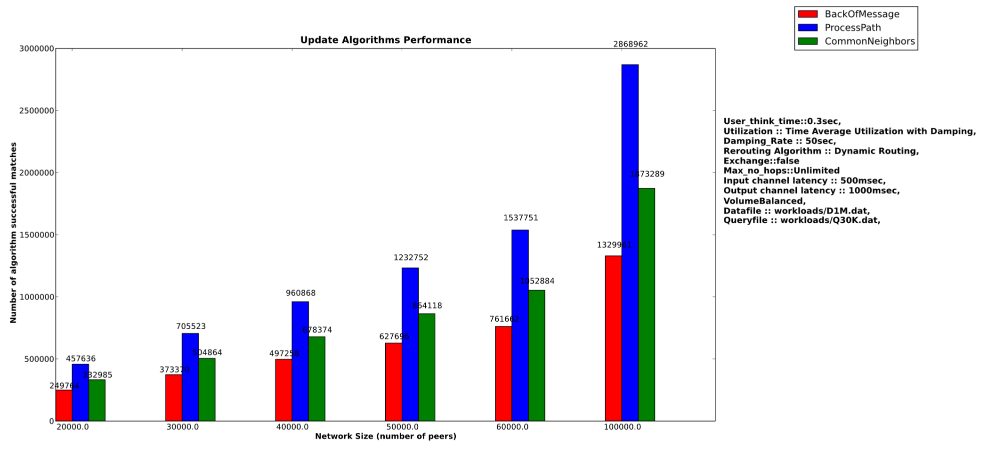


Figure 64: Performance of the update algorithms for a workload with heavy load. Exchange and Polling Periodically are deactivated.

So we conclude the following:

- The Common Neighbor and the Back of routing path algorithms is by far better than the others. Their performance alternates according to each workload, if we have many rerouting messages the routing path increases and as a result the performance of the Back of routing path algorithm improves significantly, otherwise the Common Neighbors algorithm performs better.

- The exchange algorithm comes third. We observe that the number of the updates that it performs is relatively low in comparison with the two first. Although we need to notice that the exchange algorithm except from updating also offers exchanges in the registrations of the peers' Routing Tables, so it has additional functionality. As we observed in the previous section this additional functionality of the exchange algorithm performs relatively well so we can not reject totally this algorithm, we should further examine and observe its unique characteristics.

- The piggy bag and the polling back periodically algorithms come last. Despite the low performance of the piggy bag algorithm we choose to keep it in our protocol because it operates without creating additional messages, so it has no additional cost. Polling periodically algorithm has also mediocre performance but creates additional messages. As we see in previous section the additional messages burden significantly the network's performance. If we want to increase the algorithm's performance we need to increase its frequency but this action would burden the network's performance even more because of the increased quantity of the additional messages, so we should reject polling periodically. Also we need to notice that polling back periodically performs a bit better than the simple polling but creates twofold number of messages, and so we reject it for the same reasons.

5.3.3 *Ideal Case Scenarios*

Another way to study more thoroughly the performance of the update algorithms is to examine the ideal case. As the ideal case we define the case that the update algorithms would perform so good that the information which each peer stores for the load state of every neighbor, would be the same with the real load state of the corresponding neighbor. **So we run some experiments in order to examine the consistency of the stored load information with the real load information**. With other words we tried to examine the performance of the network in the case at which each peer would have information for its neighbors' load state which would be equivalent to their real load state. Of course this was possible to be performed inside the simulation framework. We performed many workloads for the ideal state and in the figures 66, 67, 68, 69, 70, 71, 72 and 73 we illustrate one representative of the network's behavior. In the presented workload we have activated only the update algorithms which do not create additional messages.**We observe that the performance of our protocol is almost the same with the ideal case. The mean utilization of the input and output channel coincide for the two cases and the number of overloaded peers remains the same, the network's response time and the network's throughput are in the same level for our protocol and the ideal case. This means that we have a very good propagation of the load information in the network only with the update algorithms which do not need any additional messages . So the update algorithms Common Neighbors, Back of Routing path and Piggy Bag perform at the optimum level and we can reject the other 2 which deteriorate the network's response time.**

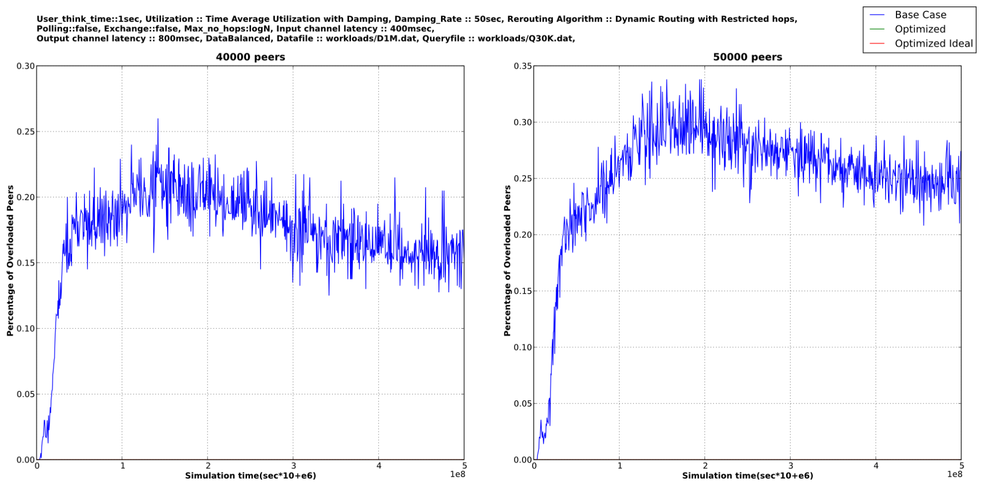


Figure 65: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization and the ideal scenario, for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange. The damping rate is 50s.

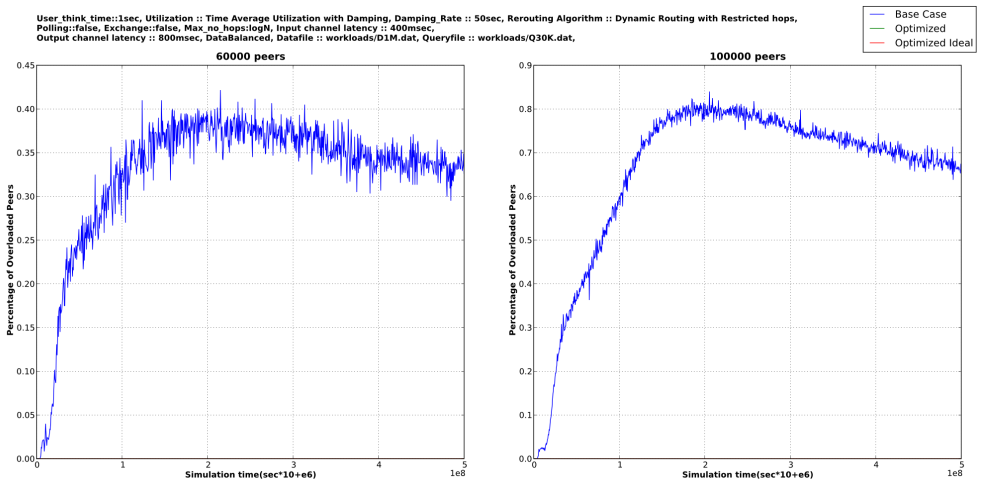


Figure 66: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization and the ideal scenario, for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange. The damping rate is 50s.

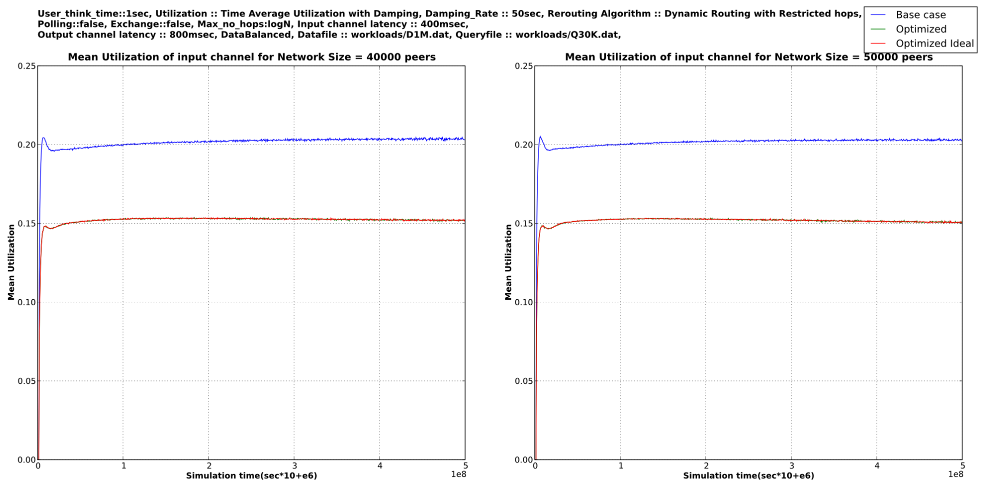


Figure 67: Mean utilization with damping 50s of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing and the ideal scenario, for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange.

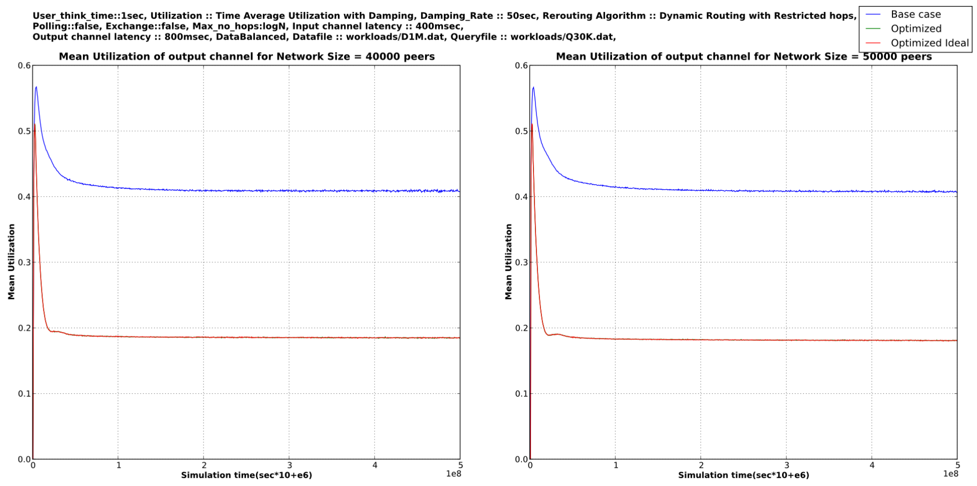


Figure 68: Mean utilization with damping 50s of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing and the ideal scenario, for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange.

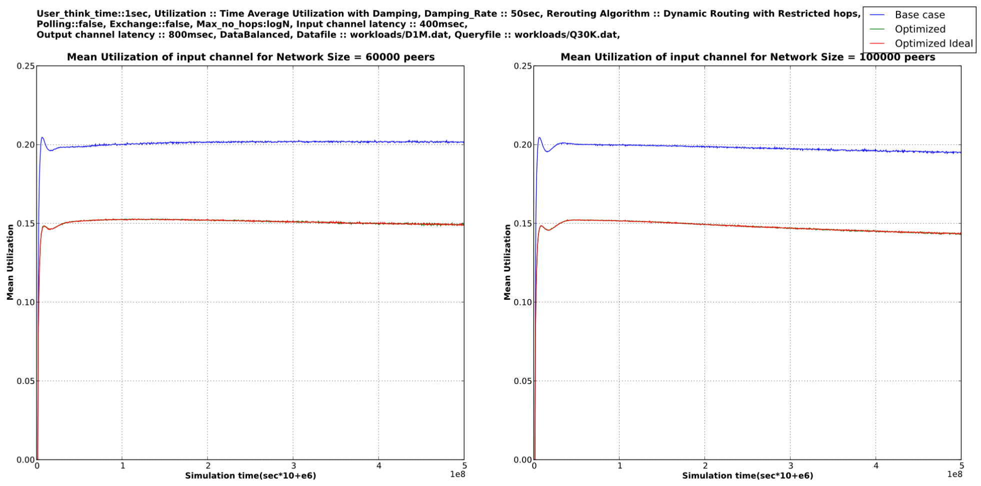


Figure 69: Mean utilization with damping 50s of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing and the ideal scenario, for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange.

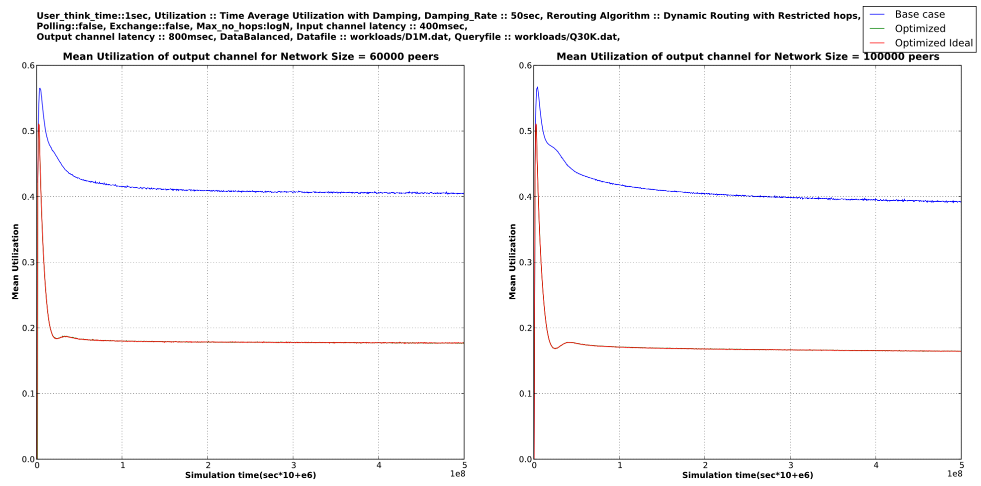


Figure 70: Mean utilization with damping 50s of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing and the ideal scenario, for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange.

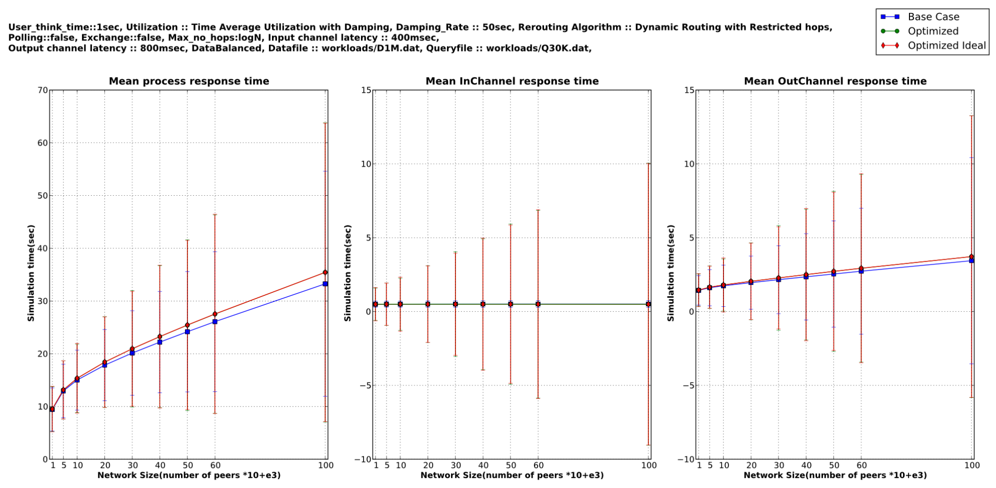

User_think_time::1sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing with Restricted hops, Polling::false, Exchange::false, Max_no_hops:logN, Input channel latency :: 400msec, Output channel latency :: 800msec, DataBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q30K.dat,

Figure 71: Process, input and output channel's response time for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization and the ideal scenario. All the update algorithms are activated except from polling and exchange. The damping rate is 50s.

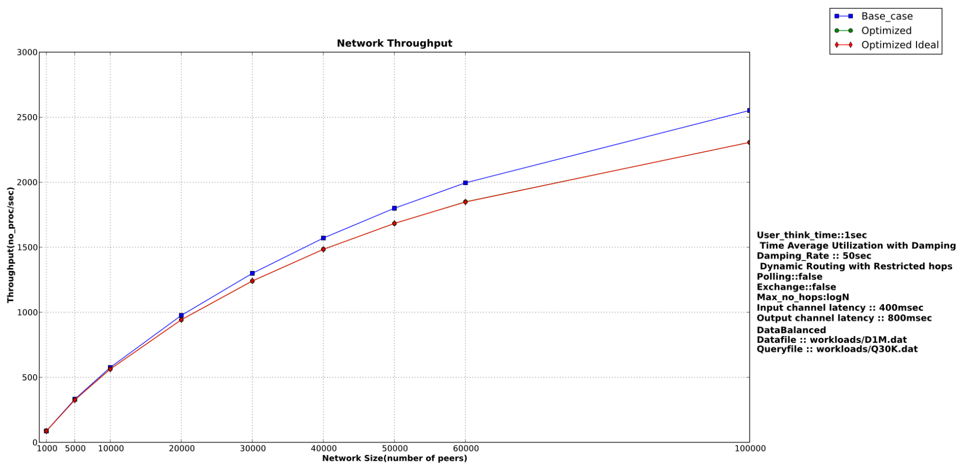
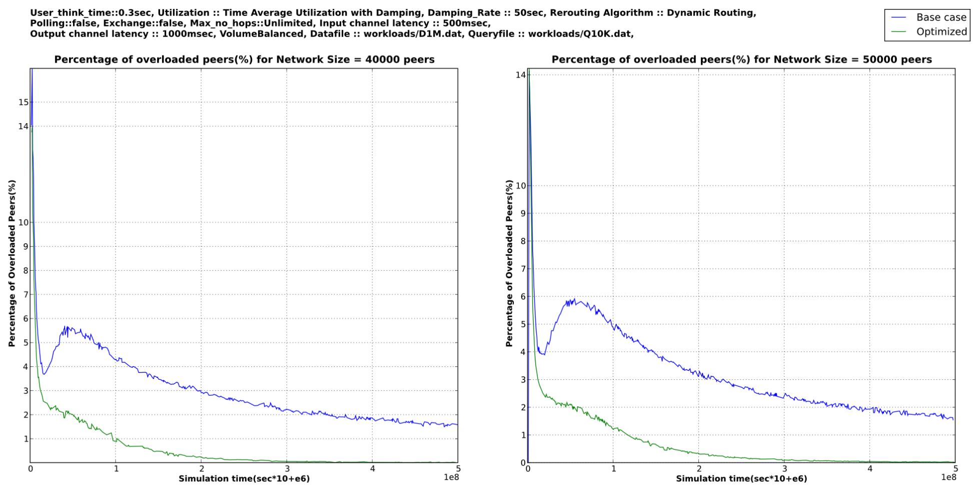

User_think_time::1sec
Time Average Utilization with Damping
Damping_Rate :: 50sec
Dynamic Routing with Restricted hops
Polling::false
Exchange::false
Max_no_hops:logN
Input channel latency :: 400msec
Output channel latency :: 800msec
DataBalanced
Datafile :: workloads/D1M.dat
Queryfile :: workloads/Q30K.dat

Figure 72: Network Throughput for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization and the ideal scenario. All the update algorithms are activated except from polling and exchange.

5.3.4 *Scalability*

In this section we will evaluate the scalability of our protocol, we will estimate its performance by examining all the scalability parameters presented in the workload schema. We performed various simulations under different workloads and we will present some representative results in order to show the protocol's performance for each scalability parameter respectively.

- In the figures of the section 5.3.1 we presented workloads for both data balancing algorithms, we show that our protocol has better performance in comparison with the traditional P-Grid protocol for both Data and Volume Balanced algorithms. The optimization is equivalent for both data balancing algorithms so our protocol is ideal for both cases.
- Except form the presented network sizes we performed experiments for network sizes 1000, 5000, 10000, 20000 and 30000. Our protocol has the same performance and equivalent optimization for all the 9 different sizes and so we conclude that has satisfactory scalability as far as it concerns the network size.
- In the figures below we present a workload with a smaller query file. The workload is the same with the one presented in the figures 39, 40, 41, 42, 43, 44, 45 and 46 but with different queryfile. We observe that the network's behavior remains the same so we achieved good scalability and in the size of queryfile.



User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing,
Polling::false, Exchange::false, Max_no_hops::Unlimited, Input channel latency :: 500msec,
Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q10K.dat,

Figure 73: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange. The damping rate is 50s. Small queryfile.

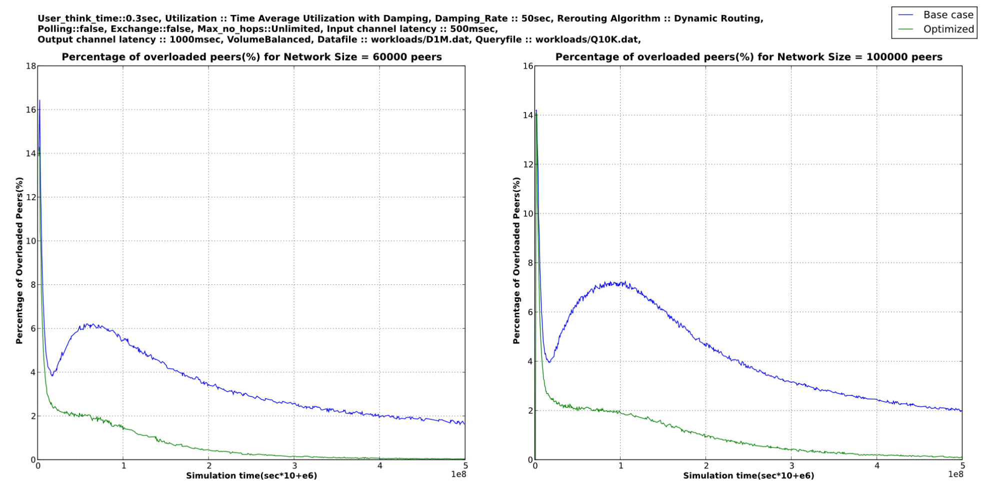

User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing,
Polling::false, Exchange::false, Max_no_hops::Unlimited, Input channel latency :: 500msec,
Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q10K.dat,

Figure 74: Percentage of overloaded peers for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange. The damping rate is 50s. Small queryfile.

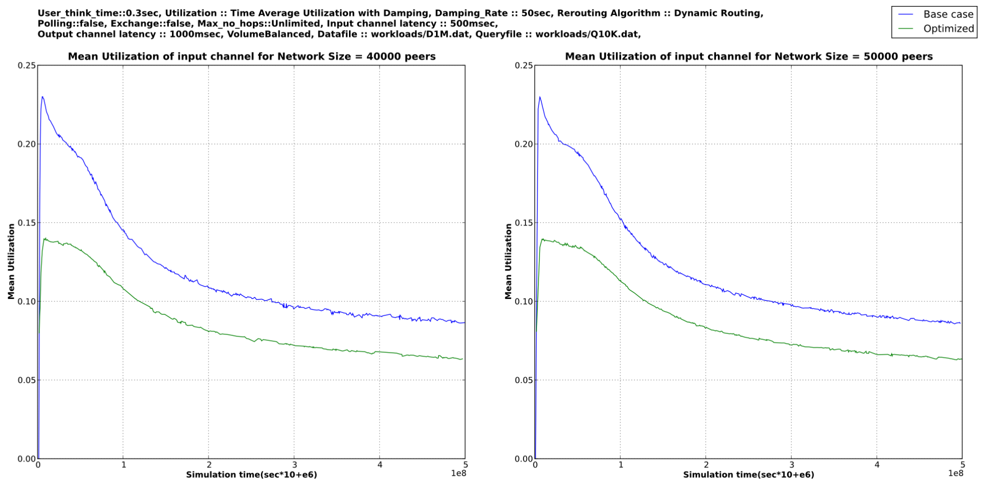

User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing,
Polling::false, Exchange::false, Max_no_hops::Unlimited, Input channel latency :: 500msec,
Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q10K.dat,

Figure 75: Mean utilization with damping 50s of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange. Small queryfile.

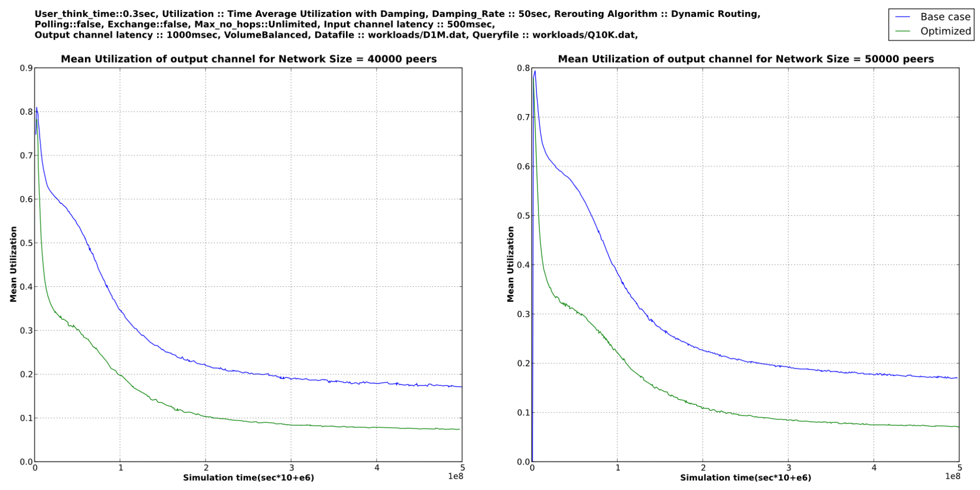

User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing,
Polling::false, Exchange::false, Max_no_hops::Unlimited, Input channel latency :: 500msec,
Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q10K.dat,

Figure 76: Mean utilization with damping 50s of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 40000 and 50000 peers. All the update algorithms are activated except from polling and exchange. Small queryfile.

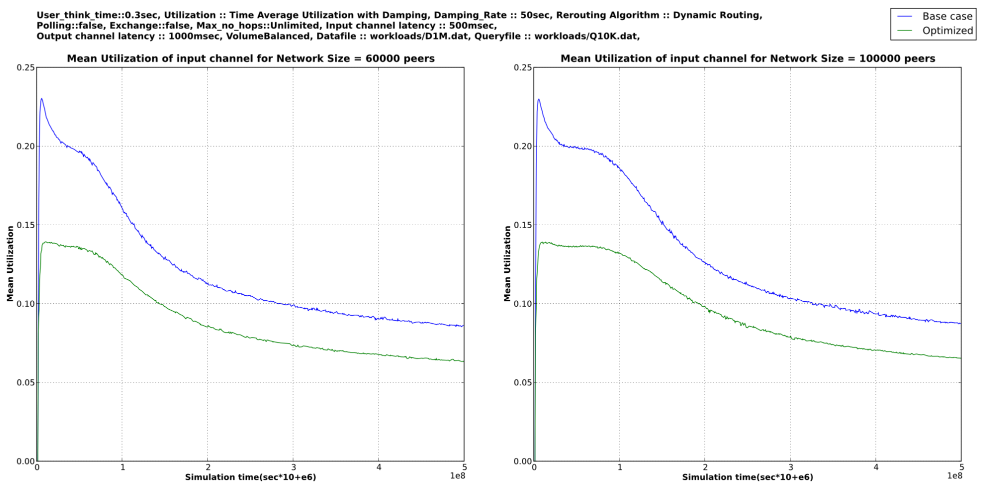

User_think_time::0.3sec, Utilization :: Time Average Utilization with Damping, Damping_Rate :: 50sec, Rerouting Algorithm :: Dynamic Routing,
Polling::false, Exchange::false, Max_no_hops::Unlimited, Input channel latency :: 500msec,
Output channel latency :: 1000msec, VolumeBalanced, Datafile :: workloads/D1M.dat, Queryfile :: workloads/Q10K.dat,

Figure 77: Mean utilization with damping 50s of the input channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange. Small queryfile.

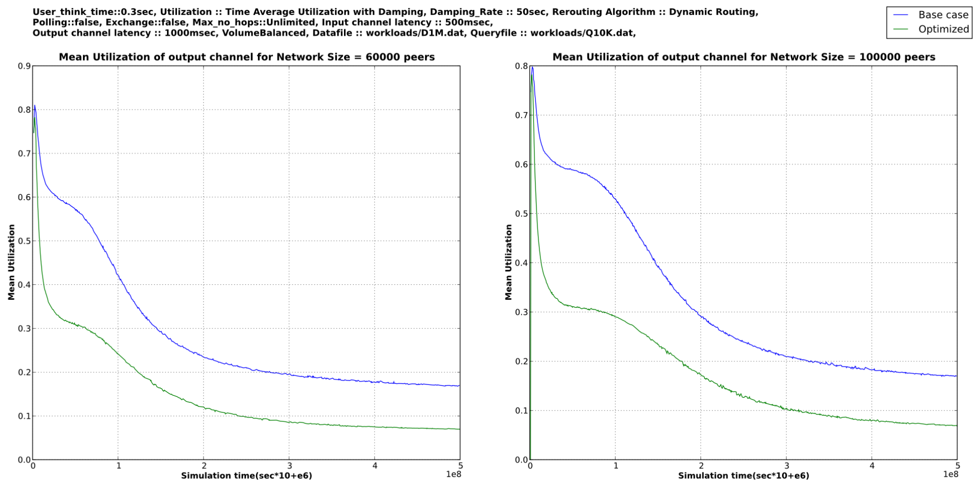


Figure 78: Mean utilization with damping 50s of the output channel for the traditional P-Grid protocol and our protocol with the Dynamic Routing for network sizes 60000 and 100000 peers. All the update algorithms are activated except from polling and exchange. Small queryfile.

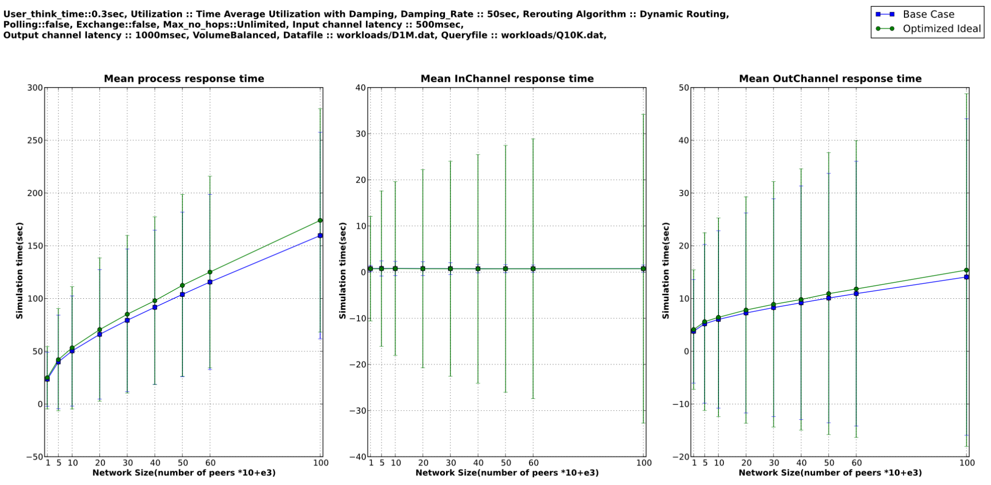


Figure 79: Process, input and output channel's response time for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization. All the update algorithms are activated except from polling and exchange. The damping rate is 50s. Small queryfile.

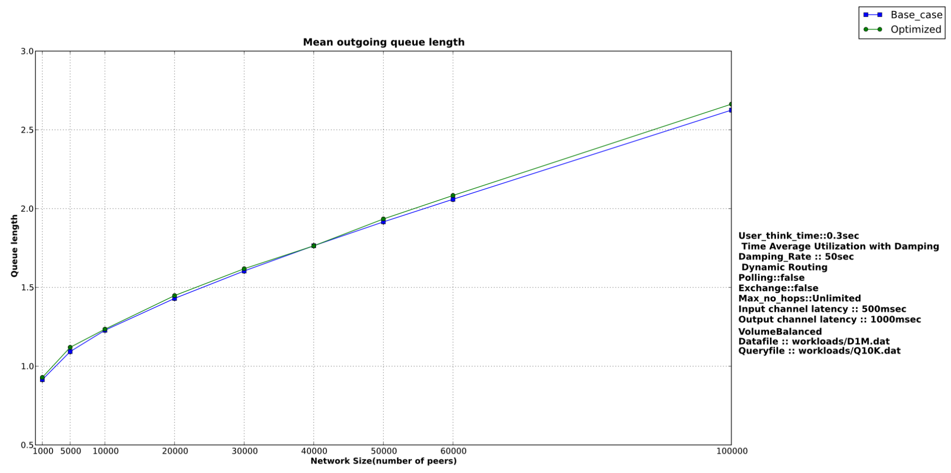


Figure 80: Network Throughput for the traditional P-Grid protocol and our protocol with the Dynamic Routing with damping utilization. All the update algorithms are activated except from polling and exchange. Small queryfile.

6 CONLUSION

As observed from the previous chapter our protocol has better performance in comparison with the traditional P-Grid protocol. Some general purpose observations extracted from our simulation results are the following:

- Time average Utilization with damping which offers us a more reliable view of the network's load for each simulation time, performs better than the time average Utilization. The time average utilization performs satisfactory but converges slower than the first and furthermore does not give us as recent view of the network's load as the time average with Damping. So we propose the time average Utilization with damping in order to estimate the peers' utilization of the input and output channel.

- All the proposed rerouting algorithms have equivalent performance. The Dynamic Routing and the Dynamic Routing with restricted number of hops perform a bit better than the Best Neighbor when the network is extremely loaded. This can lead us to the observation that the most important factor which optimizes the network's performance is firstly the good propagation of the load information an secondly the alternative rerouting paths which requires additional rerouting messages.

- Our protocol reduces significantly the number of overloaded peers and the mean utilization of the peers' input and output channels. These great reductions exceed the range of 50 percent and so the processing ability of the peers increases. Also offer greater consistency and resistance in the network.

- The amplitude of the observed optimization in the number of the overloaded peers and in the utilization of the peers' input and output channel do not improve the network's response time on the contrary deteriorate it a bit.

- The update algorithms operate at an optimum level and as a result we have consistent and cohesive load information stored in the peers' Routing Tables.

- In the presentation of the ideal case we see that the update algorithms which do not need additional messages can give us an optimum performance. The Biggy Bag, the Back of process path and the Common Neighbors algorithms can guarantee good propagation of the load information without the need of additional messages. The only drawback of these efficient update algprithms is a small overhead in the message size which can be technically supported from the realistic peer to peer networks.

- The increased amount of additional messages, even if are sent between underloaded peers, increase the network's response time.
- The exchange algorithm does not improve the utilization of the peers' input and output channel but the increased amount of additional messages that it creates deteriorates the network's response time. Here we have a trade-off, we need to choose between the unique functionality of the specific algorithm and the maintenance of the network's response time.
- The polling periodically and the polling back periodically update algorithms in order to be functional deteriorate significantly the network's performance and so we reject them.
- All the proposed algorithms have a good scaling regarding the network's size, the data balancing algorithms and the Query answer size.
- The additional rerouting messages, which are created by the Dynamic Routing and Dynamic Routing with restricted number of hops algorithms, do not burden the process response time further.

All in all, we accomplish to relief the overloaded peers of the network by transferring a portion of their load in the underloaded peers and so we have a more uniform load distribution regarding the network traffic. Despite the succeeded more uniform load distribution among the network's participants we did not manage do improve the network's response time. Also we maintain the process response time in the same level, with a negligible increase, despite the additional rerouting messages.

In the future, we would like to experiment more with higher dimensionality spaces and give more realistic datasets for such problems. Also we would like to adapt the framework described here to work with dynamic networks, but this also requires work in terms of specifying realistic behavior of peers who join, leave and fail. Also it would require a careful assessment of different metrics to perform the comparison.

Another aspect is to apply this technique of routing path modulation according to the neighbors' load and in other peer to peer protocols. This of course would require a more thorough study of the corresponding overlay network topologies.

Finally we would like to examine the network's behavior with different load factors. We could define different than the proposed metrics by taking into consideration the network's characteristics that we want to improve. For example instead of the utilization of a channel we could measure the response time of a channel or its service ability. Also the estimation of a peer's load factor could emerge by examining combination of multiple parameters.

BIBLIOGRAPHY

[1] Karl Aberer. P-grid: A self-organizing access structure for p2p information systems. In *CoopIS '01: Proceedings of the 9th International Conference on Cooperative Information Systems*, pages 179–194, London, UK, 2001. Springer-Verlag. ISBN 3-540-42524-1.

[2] Karl Aberer, Philippe Cudré-Mauroux, Anwitaman Datta, Zoran Despotovic, Manfred Hauswirth, Magdalena Punceva, and Roman Schmidt. P-grid: a self-organizing structured p2p system. *SIGMOD Rec.*, 32(3):29–33, 2003. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/945721.945729.

[3] Stephanos Androutsellis-theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36:335–371, 2004.

[4] Roman Schmidt Renault John Karl Aberer Anwitaman Datta, Manfred Hauswirth. Range queries in trie-structured overlays. *Fifth IEEE International Conference on Peer-to-Peer Computing, "Use of Computers at the Edge of Networks (P2P, Grid, Clusters)"*, 2(1):41–53, 2005. ISSN 0163-5808.

[5] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proceedings of 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM 2007, Anchorage, AK, USA*, pages 79–84, May 2007. doi: 10.1109/GI.2007.4301435. URL `http://doc.tm.uka.de/2007/OverSim_2007.pdf`.

[6] G. Scott Graham Kenneth C. Sevcik Edward D. Lazowska, John Zahorjan. *Quantitative System Performance:Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.

[7] Manfred Hauswirth EPFL Karl Aberer, Anwitaman Datta, 2003. The Quest for Balancing Peer Load in Structured Peer-to-Peer Systems.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM. ISBN 1-58113-411-8. doi: http://doi.acm.org/10.1145/383059.383072.

[9] Roman Schmidt EPFL Renault John, Karl Aberer, 2004. Range Queries in P-Grid.

[10] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, 2001.

[11] Hans Sagan. *Space-Filling Curves*. Springer-Verlang, .

[12] Hans Sagan. *Space-Filling Curves*. Springer-Verlang, .

[13] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, 2001. ISSN 0146-4833. doi: http://doi.acm.org/10.1145/964723.383071.

[14] Ling Stribling J Rhea S C Joseph A D Kubiatowicz J D Zhao, B Y | Huang. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications.*, 2(1):41–53, 2004. ISSN 0163-5808.