ENSC835: Communication Networks

Examination of Routing Algorithms in Distributed Hash Tables (DHTs) for Peer-to-Peer (P2P) Networks

Spring 2008

Final Project

Kevin Thomas

http://www.sfu.ca/kta18/ENSC835Project.html

kta18.at.sfu.ca

Table of Contents

Abstract

Peer-to-Peer (P2P) networks must make use of sometimes novel approaches in order to locate nodes (peers) that hold the information being searched for. Structured vs. unstructured P2P networks are compared. The concept of a Distributed Hash Table (DHTs) is explored as a solution, particularly the Chord DHT implementation. The routing capabilities and performance of P2P networks using the Chord DHT protocol is examined using the OverSim simulation tool. The results of sixteen different simulations under four groups are examined using three different parameters:

- Pattern of growth within a group under exponential growth of the number of peers
- Effect of exponential growth on the percentage of delivered messages
- Effect of exponential growth on the average number of hops needed to find a recipient

Conclusions of network behavior in all cases are drawn. The challenges faced while attempting to implement the project are discussed and the lessons learned listed. The direction of future work on the subject is touched upon. Finally, two examples of real-world applications of DHTs are given.

INTRODUCTION

What are Peer-to-Peer (P2P) Networks?

A Peer-to-Peer (P2P) network or system is “a self-organizing system of equal, autonomous entities (peers) which aims for the shared usage of distributed resources in a networked environment avoiding central services” [1].

In contrast, a network that is built on the centralized (client-server) paradigm concentrates authority regarding system parameters within entities (servers), which are given charge of many other dependent entities (clients). The 'system parameters' referred to here may cover such aspects as:

- Provision of services (file serving, database searches, audio/video conferencing, etc.)
- Management of resources (bandwidth, storage space, processing power, etc.)
- Co-ordination and control of operations

However, networks built using such paradigms possess inherent problems with regards to scalability, security, reliability, flexibility and quality of service. Servers have a limit to the number of clients whose requests can be handled, before performance in terms of response time and data transfer speeds begins to degrade due to increasing load. In addition, they represent vulnerabilities in the network in terms of bottlenecks for traffic and network failure. When a server is compromised due to too much load or failure, the clients dependent on it for resources are cut off from the rest of the network, denied access to resources and the global objective of the network as a system is also impacted. Management and modification of server functionality is also difficult and expensive without extensive changes to the system as a whole.

P2P networks, by the very nature of the paradigm that governs their fundamental architecture present an alternative to the problems that plague the client-server methodology of system design and implementation. Full distribution of responsibility for service provision, resource management and co-ordination among peers means that each new peer that joins the network contributes resources and power to the network, instead of becoming a burden that must be serviced. Also, departure or failure of a peer does not impact the network as severely as failure of a server entity, as there is no single point of compromise. Peers can co-ordinate among themselves to work around failures. The P2P system that

thus emerges from the interacting collective of peer nodes is also more adaptable to change. The fundamental differences between a client-server and peer-to-peer network is shown in Figure 1.

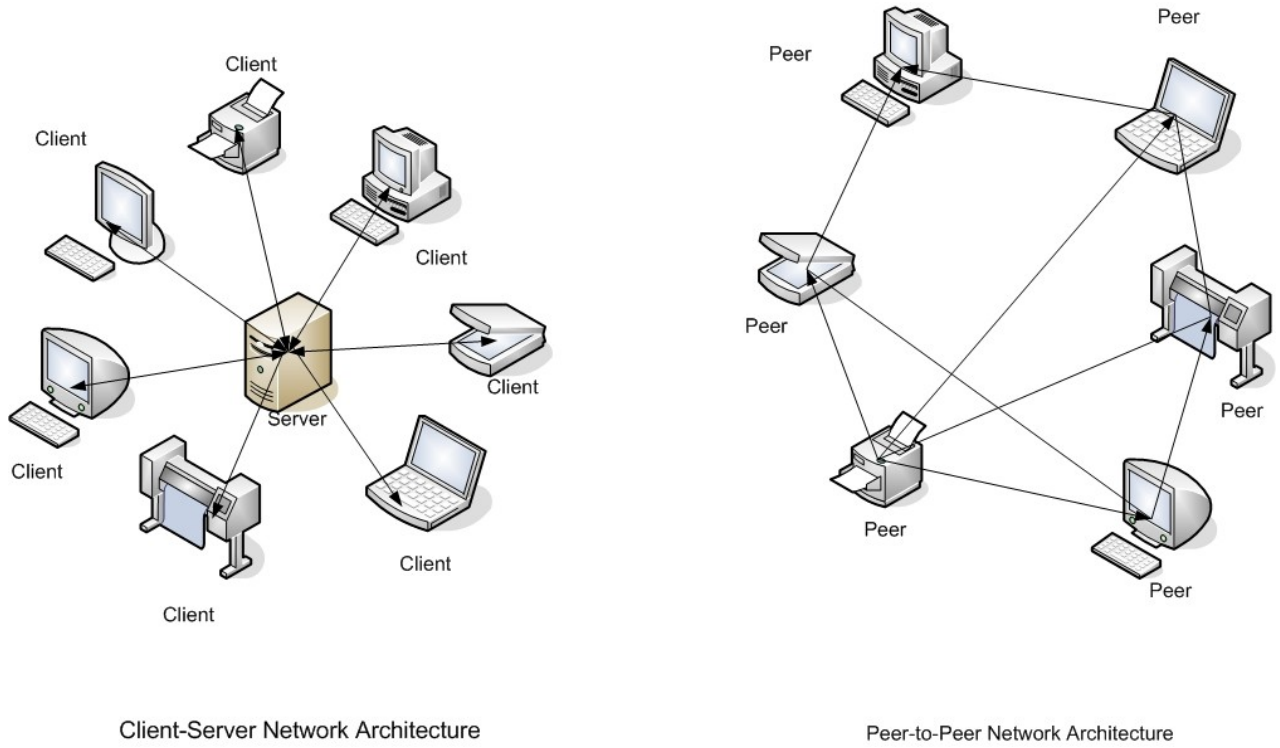


Figure 1. Client-Server vs. Peer-to-Peer Network Topology

Structured vs. Unstructured P2P Networks

Despite inherent strengths that enable P2P networks to overcome the pitfalls of the client-server methodology, P2P networks face unique challenges of their own. One such challenge is the ability to locate other peers efficiently. The goal is to be able to achieve the efficiency and guaranteed results provided by a server while being fully distributed at the same time. Meeting this challenge is particularly crucial in massively distributed P2P networks, containing thousands or even millions of nodes.

The Gnutella protocol was an initial attempt at achieving a fully distributed P2P system for file sharing. When Gnutella is implemented in a network of peers, the resulting routing mechanism that is used in order to find other peers that have a piece of data and retrieve it is described as 'network flooding'. In short, when a peer sends out a query regarding another peer or piece of data, it sends it to all peers within its neighboring peers. The end result is a flood of messages throughout the network in

the hope of finally reaching the intended recipient and getting a reply to the initial query. Even though the network also implements a TTL (Time To Live) restricting the number of hops a message is allowed to traverse, the amount of traffic generated by such a routing methodology is immense. As a result, network performance becomes very poor as the number or peers in the network grow ever larger. P2P networks implementing this manner of routing methodology are known as 'unstructured' P2P networks.

Finding a solution to this problem requires one to answer the question: how can some form of order be brought to a P2P network that will allow efficient lookup of peers and the data that they possess, without bringing in any manner of centralization into the system? The idea of using distributed hashing to create Distributed Hash Tables (DHTs) in P2P networks has emerged over the past several years as a viable solution to this problem. In order to understand what a DHT is and how it operates, the concept of hash functions must be understood first.

The Concept of Hash Functions

A hash function [2] is an algorithm or mathematical function that operates on some form of data, and produces an integer as its output. The result of the hashing process is known as a hash value. A typical use of hash functions is to quicken the lookup of data in tables, such as searching databases and files. Hash functions that work effectively for P2P satisfy some basic properties, such as:

- **Determinism**:

  A given piece of data, when input must hash to the same output value each time.
- **Uniformity**:

  For a given input range, hash functions have a range over which output hash values are produced. The mapping should produce hash values spread out as evenly as possible over the output range.
- **Continuity:**

  Two items of input data that differ by only a little should be mapped to equal or nearly equal output hash values.

A hypothetical hash function operating on a peer's IP address to produce an output hash value is shown below.

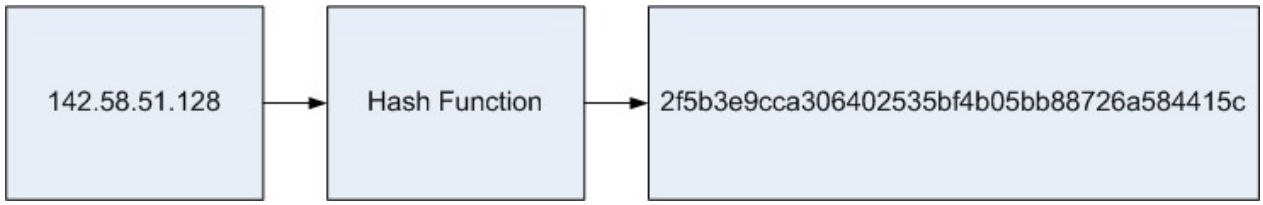


Figure 2. A hypothetical hash function operating on an IP address to provide an output hash value.

Centralized vs. Distributed Hash Tables

The hash function described in the previous section can be used in the creation of a data structure known as a 'hash table'. A hash table is used to link a 'key' field with an associated 'value', forming what is known as a 'key-value' pair, denoted by *(key, value*). A common example may be a person's name (key) and telephone number (value). However, instead of using the data in the key field as it is, it is operated on by a hash function, producing an output hash value. This hash value becomes the key, and is stored along with the associated value in the hash table. In order to retrieve the key's associated value, the key in it's original format (e. g., the person's name) is hashed and the hash value looked up in the hash table. When located, the associated value is retrieved. Thus, the final purpose of the hash table is to support a 'lookup' operation where, given the key (the person's name), it finds its associated value (telephone number). In a communication network, the purpose in this case may be to find a node using some uniquely distinguishing property such as its IP address (the key), in order to retrieve an item of data associated with that node (the value).

In a network built on client-server principles, the hash table would be stored on a central server, and a client desiring to locate another client responsible for some data would request the server to lookup that node's key and retrieve its associated value. However, such a methodology cannot work in a P2P network, as the very nature of the network architecture precludes the implementation of any centralized routing principles to find and retrieve data. The alternative is to distribute the hash table over all the participating peer, by partitioning it and giving each peer responsibility for the partition assigned to it. Along with partitioning and distributing the hash table, there must also be a algorithm to efficiently route queries from peers attempting to find other peers responsible for a particular partition, in order to retrieve the key(s) stored on that peer's partition. Thus, the principle of partitioning and distributing the hash table over the participating peer nodes, along with a routing algorithm to efficiently route requests

to using a peer's key to retrieve its associated values forms the heart of the Distributed Hash Table (DHT).

Different perspectives on implementing the principles of DHT Algorithm have resulted in various types of DHT implementaions over the years: Chord, Pastry, CAN (Content Adressable Network), Symphony, Viceroy and Kademlia are among the best known ones. This project focuses only on the Chord DHT algorithm.

The Chord DHT Algorithm

The Chord DHT algorithm was first published by Stoica et al. [3] in 2001. Chord uses *l*-bit identifiers as its keys, which are integers that span the continuous closed set of $[0, 2^l - 1]$. The Chord algorithm maps the peers in the underlying topology of the network into a virtual, one-dimensional, circular 'identifier space' where the identifies are the *l*-bit keys of the peers. The nature of the mapping is modulo 2^l, i. e. the next identifier after $2^l - 1$begins again at 0. The implementation of the Chord algorithm used for simulation in this project makes use of the US Secure Hash Algorithm 1 (SHA1) [4].

In the Chord implementation, each peer node and the individual data items on it have individual identifiers produced by the hash function. A peer's identifier is known as it's 'ID' and a data item's identifier is known as a 'key'. A peer is responsible for hosting a particular data item's *(key, value)* pair *(k,v)* if the value of its ID is greater than or equal to *k* (i. e. ID $\geq k$). This peer will be known as that particular key's 'successor'. If the Chord circle is pictured as having the keys arranged on it in increasing order as we move around it clockwise, a given peer will be responsible for hosting the keys that precede it in value. This is illustrated in the figure below. In this scenario, 5-bit identifiers are used, i. e. from 0 to 15 for peers and data items. Assuming 8 peers and 5 data items exist, they would be mapped to the Chord identifier space as shown below.

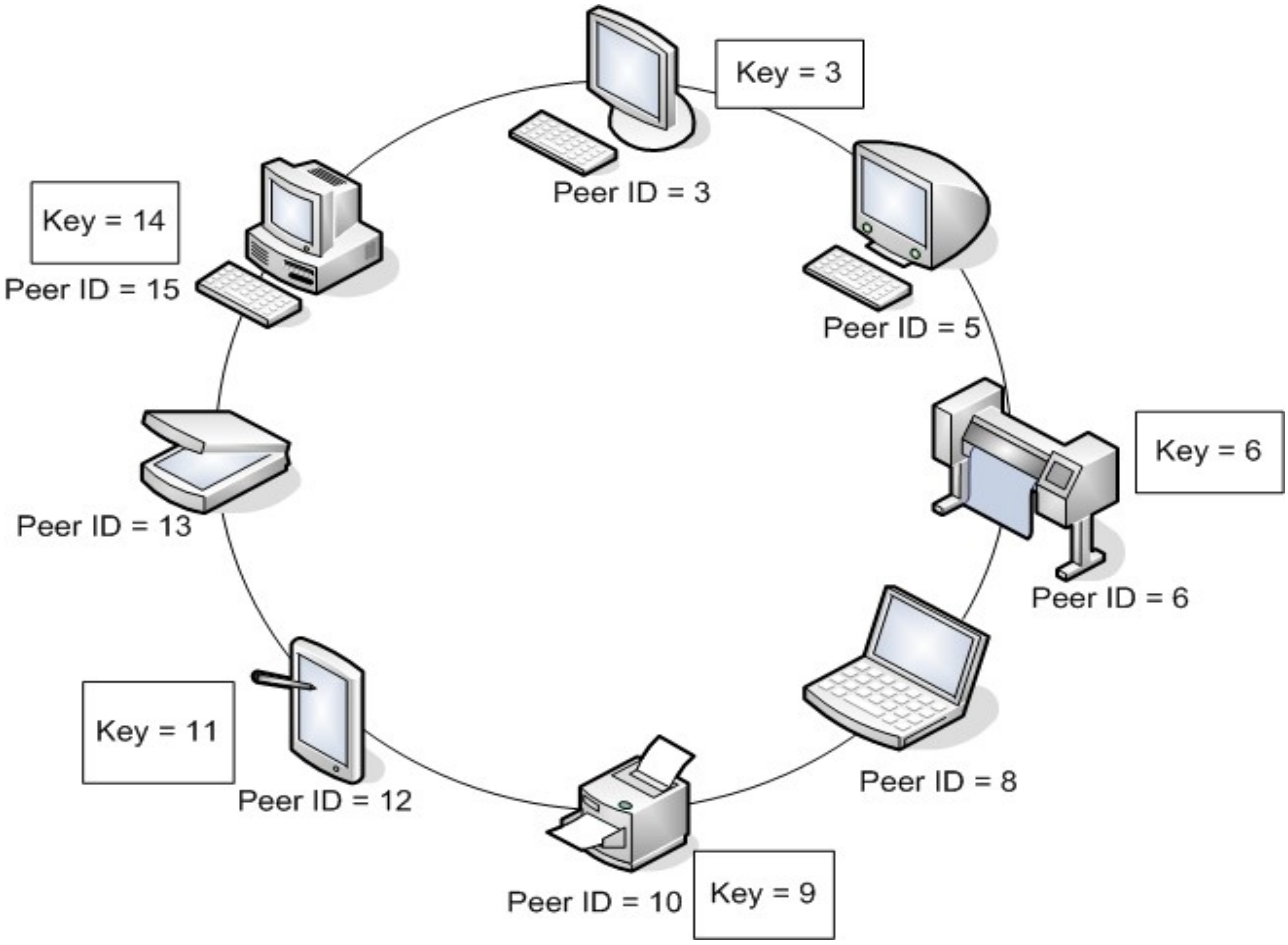


Figure 3. Peers and data items mapped to Chord identifier space by ID and key value.

Thus, it can be understood that the Chord identifier space/circle is formed by the collective of the peers, their ID's and the uniquely associated *(key, value)* pairs of the data items that they host. With the identifier space in place, the issue of efficient routing of messages related to lookup operations becomes essential, in order to achieve good network performance. The most efficient method of routing within the identifier space requires each peer to maintain some amount of per-node state in the form of a 'finger table' (essentially, a routing table) and a pointer to its predecessor within the identifier space.

The purpose of the finger table is for a peer to maintain a record of pointers to other peers in the identifier space. For a given *l*-bit identifier key used to construct the identifiers, a peer must maintain a maximum of *l* entries in its finger table. Also, for a given peer *p* in the Chord identifier circle, the i^{th} entry in its finger table holds the pointer to a neighbor whose ID follows after its own by at least $2^i - 1$. The identifier of the neighbor is therefore essentially given by the relation $ID_{neighbor} = ID_{peer} + (2^i - 1)$,

where $1 \leq i \leq l$. This is illustrated in the table below, taking the peer in figure 3 with ID = 5.

Finger Table of Peer with ID = 5 from Figure 3.		
Finger Table Entry No.	Neighbour	Successor ID
0	$ID_{peer} + 1$	6
1	$ID_{peer} + 2$	8
2	$ID_{peer} + 4$	12
3	$ID_{peer} + 8$	-
4	$ID_{peer} + 16$	-

The implication of implementing a routing methodology constructed in this manner is that each peer possesses more pointers to neighbors that are situated 'close by' in its region of the identifier space, and fewer pointers to neighbors that are 'far away'. Also, irrespective of how large the P2P network grows in terms of the number of peers, each peer will have to maintain a finger table of l entries at the most. Given that SHA-1 uses 160-bit identifiers, this is still a reasonable size of finger table to maintain, for a Chord implementation that uses SHA-1 to generate the necessary identifiers. Apart from the finger table, each peer also maintains a pointer to its immediate predecessor in the identifier circle. This is used in a protocol to maintain accuracy of per-node state information within a dynamic P2P network environment and will be explained later.

With the above infrastructure in place, message routing within the Chord identifier circle proceeds as follows:

- A peer with identifier $ID_{thisPeer}$ receives a query message destined for another recipient ($ID_{receipient}$).
- It examines its finger table and locates the identifier of a neighbor whose ID makes it the closest predecessor to the recipient on the identifier circle, i. e. $ID_{neighbor} < ID_{recipient.}$
- It forwards the message to that neighbor with identifier $ID_{neighbor}$.
- This repeats until the message reaches a node which determines from its finger table that $ID_{recipient}$ is either its immediate successor or lies between its own ID and the ID of its immediate successor.
- The node reports its immediate successor as the query's result, which is sent back to the original querying node.

Apart from a means of efficiently routing messages within the identifier space, DHT algorithms also need to provide a means for a P2P network to allow peers to join and leave the network, as well as dealing with events that could be detrimental to network performance, such as node failures. Chord makes provisions for these capabilities in the following manner.

A node that wishes to join the P2P network must first determine its own ID ($ID_{joiningPeer}$). One common means of doing this is to hash its own IP or MAC address. In addition, it must have the identifier of some existing peer within the P2P network. This peer is known as the 'bootstrap node', that helps introduce the joining node to the network. The joining node queries the bootstrap node for its own ID; the response of this query will be the identifier of the peer that should become the joining node's immediate successor. The next step is to notify its successor of its presence using its ID; this causes the successor to update its state information to register the joining node as its predecessor on the identifier circle. The final step is for the joining node to fill up its finger table by repeatedly querying the bootstrap node for the identifiers of peers that are $ID_{joiningPeer} + (2^i - 1)$ (i. e. 1, 2, 4, 8, 16, 32,...) positions away on the identifier space, where *i* is the entry in the routing table.

The environment of the P2P network may be either static or dynamic. In a static network, peers do not leave the system. A dynamic network is one in which nodes may join and leave the network at different times. Most real-world P2P networks are extremely dynamic in nature, as node arrival and departure may be very frequent. This process of nodes joining and leaving the P2P network is known as 'churn'. The updating of pointers to successors and predecessors is important, as a peer in the identifier space must have accurate information about the peers in its known neighborhood. If a new peer has placed itself in the space between it and its currently known successor, it must become aware of that fact, otherwise messages meant for the new successor will not reach it. It must also be made aware if an existing successor leaves the identifier space. In addition, it must also be aware of changes in the presence or absence of a predecessor peer. Finally, detection of node failures must also be addressed by fixing and updating information in the finger table. Chord has built-in protocols to deal with such events.

Chord uses a 'stabilization protocol' [1] in order to deal with such dynamic events in its identifier space. This is the purpose of the predecessor pointer mentioned earlier. Every peer runs a function

called 'stabilize()' periodically. This function sends a query message to its immediate known successor, requesting it to reply with the identifier of its currently known predecessor. If the reply to the query matches the peer's own identifier, it means that the information that this peer and its successor have about their immediate neighbors is correct. However, if the identifier in the reply lies between the inquiring peer's own ID and its known successor's ID, then this means that a new node with the identifier in the query has joined in the space between the two. The inquiring node therefore must adjust the information in its successor pointer to point to the identifier of the new known peer and inform the new peer to update its own predecessor pointer accordingly. Thus, the stabilization protocol lays the responsibility of a predecessor node to check on its successors and correct discrepancies in pointers. When such a correction occurs, the new successor becomes responsible for all the *(key, value)* pairs that were handled by the predecessor node; those keys are copied across from the predecessor peer, and the predecessor removes them from its records.

Similarly, Chord also has a 'fix fingers' protocol [1] to deal with inconsistencies in routing information in a peer node's finger table. A peer node checks if a timeout has occurred on any expected replies regarding queries that have been sent out. If a timeout is detected, it infers that the entry regarding the recipient peer in its finger table is incorrect, and removes that entry from the finger table. It then chooses the identifier from the preceding entry in the finger table and sends the query to that peer.

Having said this, it should also be noted that Chord requires peers to notify their predecessors and successors before voluntarily leaving the network. This is necessary so that the *(key,* value) pairs that the peer was responsible for are not lost; they are instead transferred to its immediate successor so that the data or information is preserved within the network for other peers to access.

Related Work

- Stoica et al. [3] laid the groundwork with their published paper detailing the concept and operation of the Chord DHT algorithm.
- Manku [5] conducted a detailed theoretical analysis of various routing implementations in DHTs, providing mathematical insight into the nature of routing used Chord.
- Li, et al. [] conducted simulation experiments examining the behavior of various DHT

algorithms under churn.

- Yang [] built a simulation of networks implementing two DHT algorithms, Chord and Pastry in the OPNET simulation tool environment.
- Steinmetz and Wehrle [1] provided a comprehensive yet simple and well-written description of the concept and implementation of various DHT algorithms in their book “Peer-to-Peer Systems and Applications”.
- Baumgart, Heep and Krause [8] built the open-source P2P overlay simulation tool and components necessary to conduct simulations of DHT algorithms and published a paper describing its architecture.

Project Scope

The aim of this project is to examine the behavior of a Chord DHT algorithm implemented in a P2P overlay network with varied parameters in the algorithm and P2P network. Sixteen simulations were planned under four major groups, as follows:

1. **Chord Recursive DHT Algorithm in a simple network.**
   Four simulation runs involving 16, 32, 64 and 128 peers in the P2P overlay network, each conducted for a simulation time of 1 hour (3600 seconds).

2. **Chord Iterative DHT Algorithm in a simple network.**
   Four simulation runs involving 16, 32, 64 and 128 peers in the P2P overlay network, each conducted for a simulation time of 1 hour (3600 seconds).

3. **Chord Iterative DHT Algorithm in an IPv4-based network.**
   Four simulation runs involving 16, 32, 64 and 128 peers in the P2P overlay network, each conducted for a simulation time of 1 hour (3600 seconds).

4. **Chord Recursive DHT Algorithm in a simple network with faster stabilization time.**
   Four simulation runs involving 16, 32, 64 and 128 peers in the P2P overlay network, each conducted for a simulation time of 1 hour (3600 seconds).

The purpose of groups 1 and 2 was to examine and compare the behavior of two possible types of routing methodology in the Chord DHT: iterative routing vs. recursive routing. They also looked into the behavior of the DHT algorithm with exponentially increasing numbers of nodes. Group 3 aimed to examine the behavior within a network using the Internet Protocol, version 4. The last group was conducted to observe and compare Chord's behavior with that of group 1 when the stabilization time was cut in half.

Project Implementation Details

Simulation Tools

Three software components were needed for the working of simulation environment during the course of this project:

- **OMNeT++ Discrete Event Simulator**

  OMNeT++ [9] is a discrete event simulation tool built by Andras Varga at the Technical University of Budapest, Hungary. It contains a network editor (GNED) that allows for easy construction of various network topologies on a workspace, a graphical simulation execution environment (Tkenv) that allows for configuration and observation of a simulation run, event-by-event (if desired) and two statistical recording and analysis tools (plove and scalars) for visualizing and analyzing statistics generated during the course of the simulation run.

- **INET Framework for OMNeT++**

  The INET Framework [10] is an open-source set of models built in OMNeT++ and meant for the simulation of various network protocols and topologies, such as wired, wireless and ad-hoc networks. It incorporates various protocol suites, such as TCP-IP, PPP, IEEE 802.11, Ethernet, IPv4, IPv6, OSPF, etc. that can be used in combination with models of network components such as hosts, routers and buses among others to build models of networks and test them.

- **OverSim-20070926: The P2P Overlay Simulation Framework for OMNeT++**

  OverSim[11] is developed under the scope of the ScaleNet [12] project at the Institute of Telematics, Universität Karlsruhe, Germany. OverSim allows the simulation of P2P overlay networks, using OMNeT++ and the INET Framework. Of particular interest that made it suitable for this project was the implementation of various DHT algorithms and network models built using the same, one of which was Chord.

All the above simulation tools ran on a workstation with Ubuntu Linux [13] as the operating system.

Simulation Runs and Screen Outputs

As mentioned earlier, 16 simulation runs within 4 groups were initially planned. Screen shots of some of the graphical displays of the network models and their interior components, along with some screen shots taken during simulation runs are shown below.

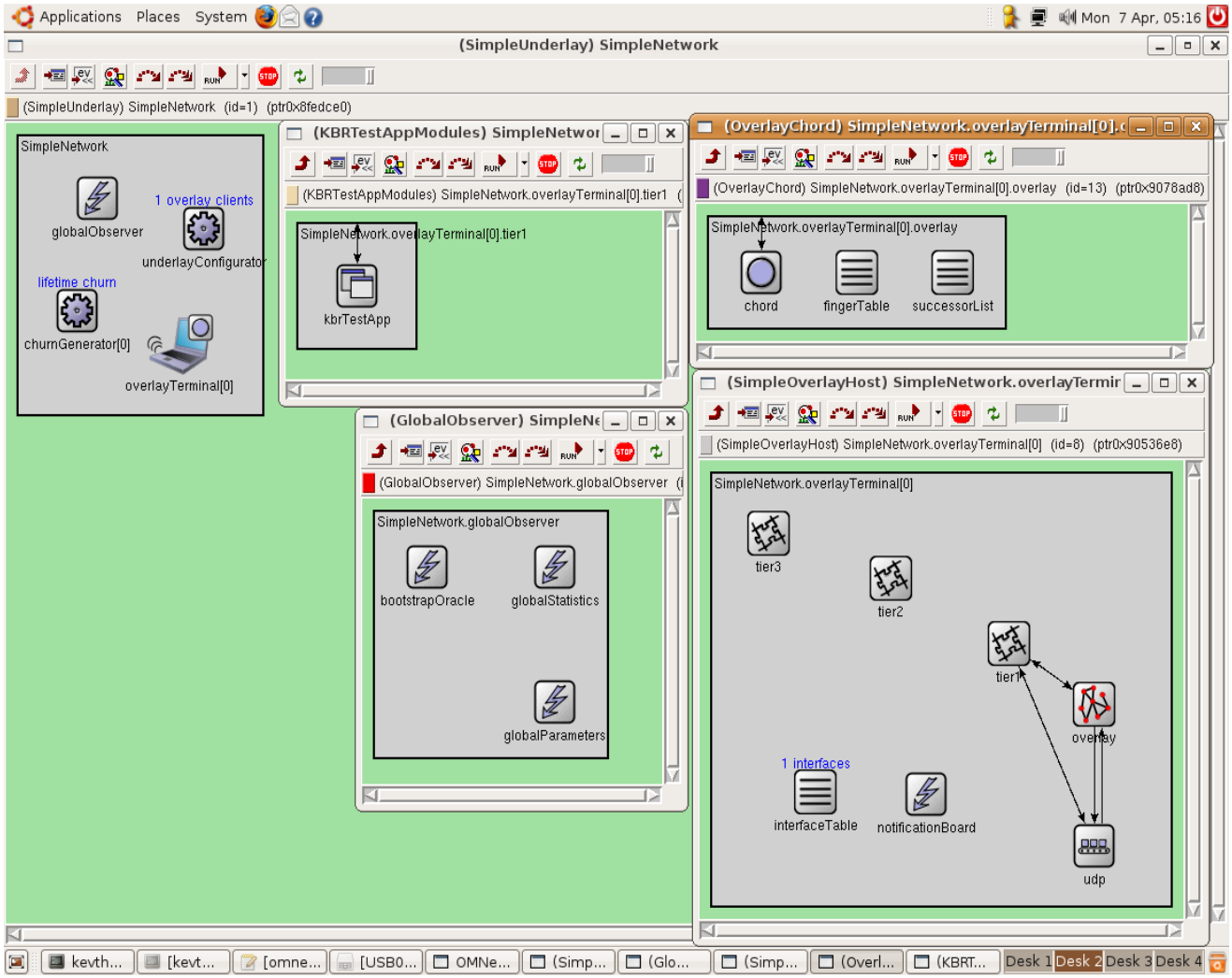


Figure 4. A screen shot of the various components of a Chord simple network simulation model

The above diagram shows a screen shot of the component modules that would constitute a Chord simulation model using a simple point-to-point protocol (PPP) for communication between peers. On the main simulation workspace are a 'Global Observer', 'Underlay Configurator', 'Churn Generator' and 'Overlay Terminal' module. The Global Observer functions as a module to set global parameters of the simulation, provide the identifier of a bootstrap peer to a node joining the network, and collects global

statistics. The Underlay Configurator is used to specify parameters of the underlying network. The Churn Generator is used to specify the type of churn (random churn, no churn, etc.), how often it occurs and other parameters related to nodes joining and leaving the network. Finally, contained within the 'Overlay Terminal' is the model of the peer. This contains a UDP module that generates UDP messages for the peer, an Overlay module that contains the three essential modules of the Chord DHT (Chord algorithm, finger table and successor list) and finally a test application module used to test the routing of messages to peers. The above mentioned modules are common to all the Chord simulations, including the one shown below with Chord running on a P2P network making use of the IPv4 protocol. The additional modules seen in this screen shot are those belonging to the access and backbone routers.

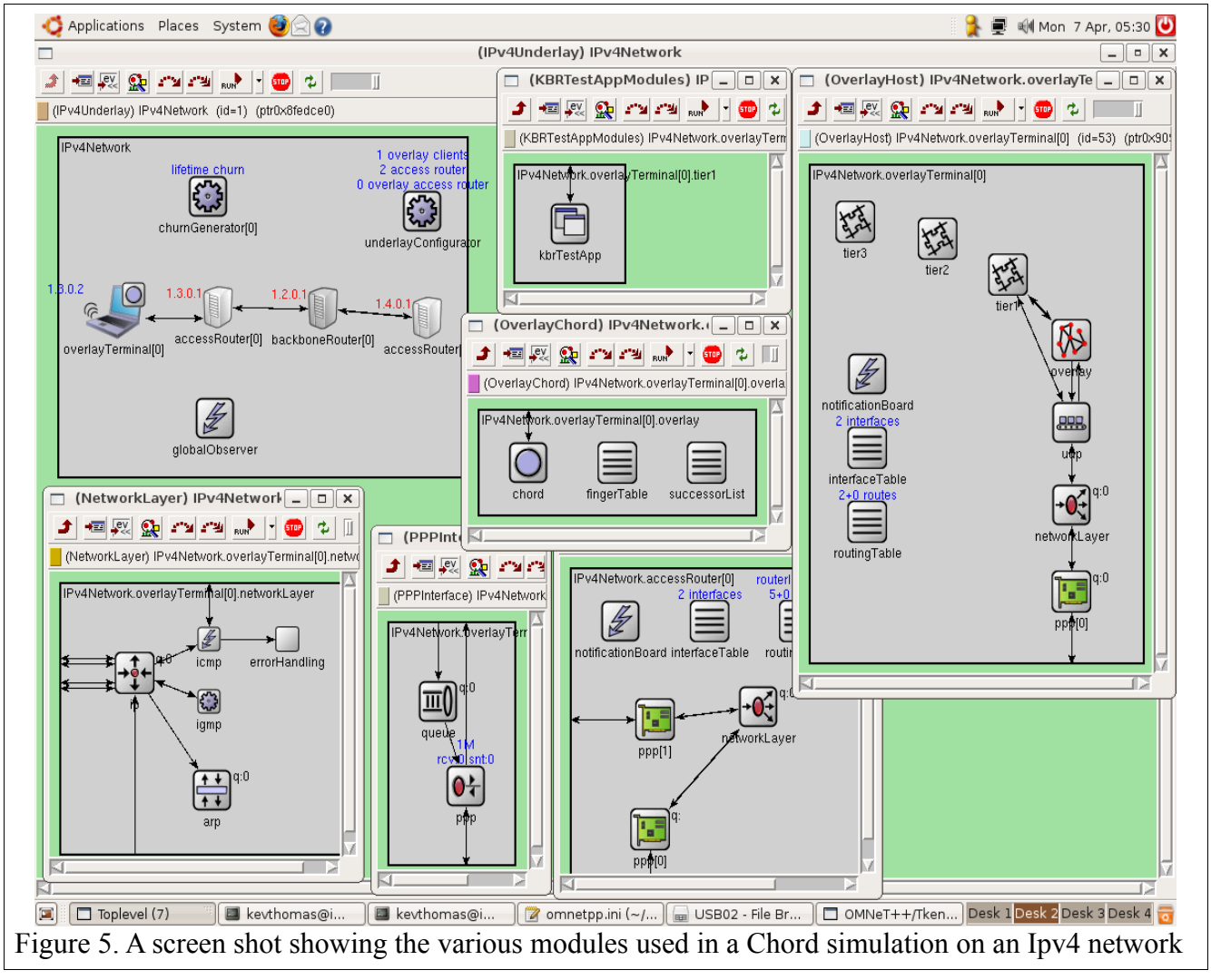


Figure 5. A screen shot showing the various modules used in a Chord simulation on an Ipv4 network

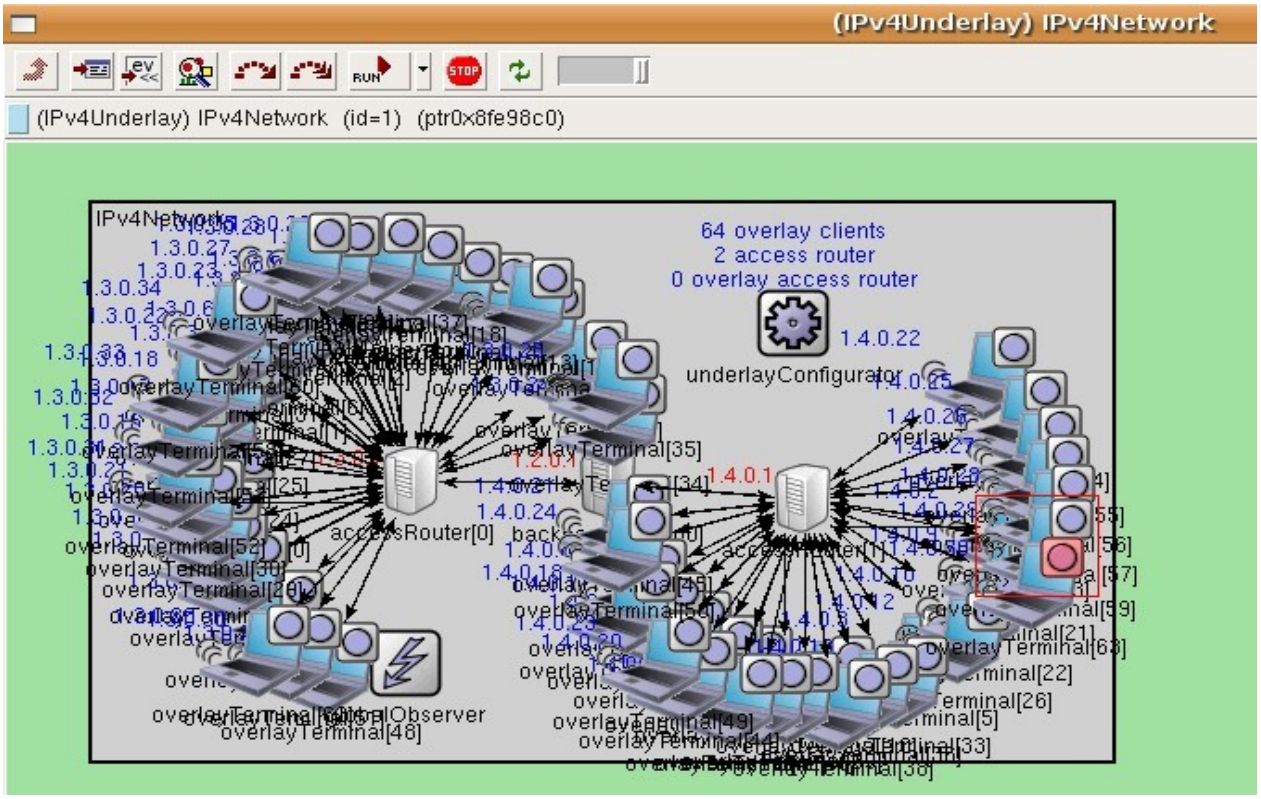


Figure 6. A screen shot taken during simulation of a Chord DHT on an Ipv4 network with 64 peers

The above diagram shows the main simulation workspace of Figure 5 during the third simulation run of group 3 (the Chord Iterative DHT Algorithm in an IPv4-based network) with 64 peers, clustered around 2 access routers.

Discussion and Conclusion

Analysis of Simulation Statistics

The simulation runs conducted generated extensive output traces. An exhaustive analysis of all the statistical data is beyond the scope of this project. Hence, with regard to time-varying data, three categories of graphs common to all the simulations were chosen and plotted using the 'plove' (**plo**t **ve**ctor) plotting tool provided by OMNeT++. These three categories of plots are:

1. **Node Population vs. Time**

   This plot describes the variation of the number of peers present in the Chord identifier space over the course of the 1 hour (3600 seconds) simulation. One's intuition expects to see an increase in the fluctuation of node population as the number of peers is increased exponentially with each simulation run under each of the four groups.

2. **Current Delivery Ratio vs. Time**

   The second category of time-varying graph shows how the percentage of successfully delivered queries to recipient peers (current delivery ratio) varies over the course of the simulation. As with the previous category, one expects to see some discernible pattern in the delivery ratio as the number of peers is grown exponentially.

3. **Global Hop Count vs. Time**

   With the perspective of the entire network as a whole, this plot describes how the average number of hops needed to deliver a message to a recipient (i. e. the 'global' hop count) peer varies during simulation time.

Instead of analyzing each simulation or group individually, the methodology of analysis within this project intends compare groups against each other. The agenda of comparison will be as follows:

- **Group 1 (Chord Recursive) vs. Group 2 (Chord Iterative)**

  As the DHT in both these groups is implemented on a simple network with peers communicating via a point-to-point protocol, this comparison aims to evaluate the behavior of the recursive routing strategy against the iterative routing strategy. In doing so, the objective is to note if the behavior of the current delivery ratio and global hop count are significantly

influenced by the routing strategy used.

- **Group 2 (Chord Iterative) vs. Group 3 (Chord Iterative, on an IPv4 network)**

  In this case, the routing strategy remains the same, but the nature of the communication protocol used by the peers is different. Group three implements an IPv4 network with peers clustered around two access routers connected by a backbone router. The objective in this comparison is to observe how the change in communication protocol affects properties such as the delivery ratio and global hop count.

- **Group 1 (Chord Recursive) vs. Group 4 (Chord Recursive, with faster stabilization time)**

  Once again, both these simulation groups are run with a simple point-to-point network communication protocol. In this case, the objective is not to compare two different routing strategies, but to compare variation of the stabilization and finger table fixing parameters with the recursive routing strategy in place. The reader will recollect that in Group 4, all other parameters are the same as Group 1, except for the time setting of the stabilization and finger table fixing parameters, which are set to half the values of Group 1. Thus, with more frequent updates of predecessor and successor pointers and fixing of routing table entries, the amount of messages are expected to approximately double. It is expected that this will reflect in the form of increased fluctuation in the delivery ratio and global hop count plots. With regard to Group 4, it should be mentioned that the fourth simulation of 128 peers could not be completed due to hardware limitations of the workstation that resulted in inability to successful run the simulation to completion. Thus, wit respect to Group 4, comparison can only be made using the results of runs with 16, 32 and 64 peers.

Trace Comparison of Node Population vs. Time for Group 1 (Chord Recursive on a simple network)

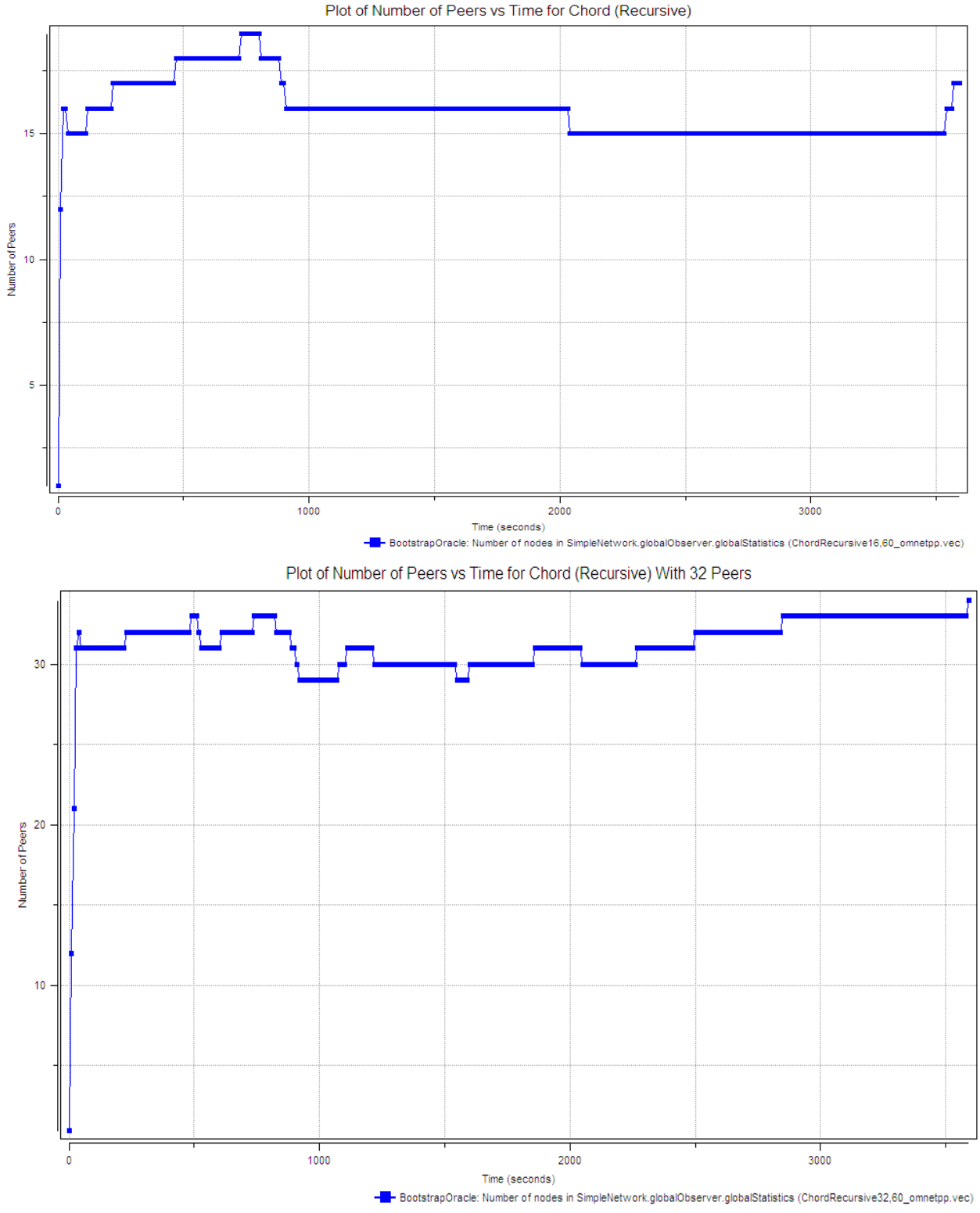


Figure 7(a) Population of Peers vs. Time with 16 and 32 peers respectively for Chord (Recursive)

Observation: Increasing churn is observed as the number of peers doubles; frequent variations in peer population observed in graph 2 with 32 peers.

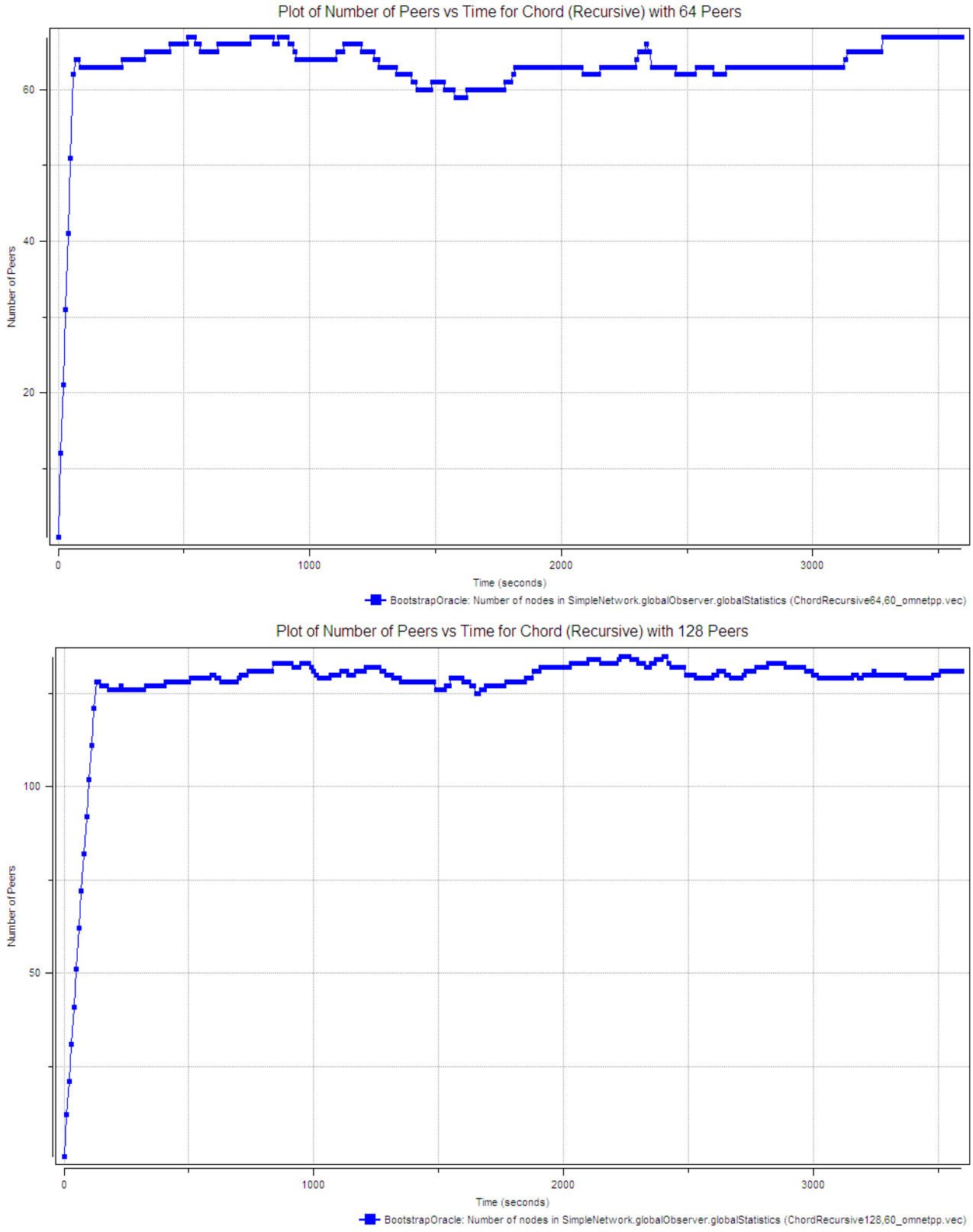


Figure 7(b) Population of Peers vs. Time with 64 and 128 peers respectively for Chord (Recursive)
Observation: Relative to 16 and 32 peers, these show increased fluctuation over shorter time frames with shorter periods of constant number of peers.

Trace Comparison of Node Population vs. Time for Group 2 (Chord Iterative on a simple network)

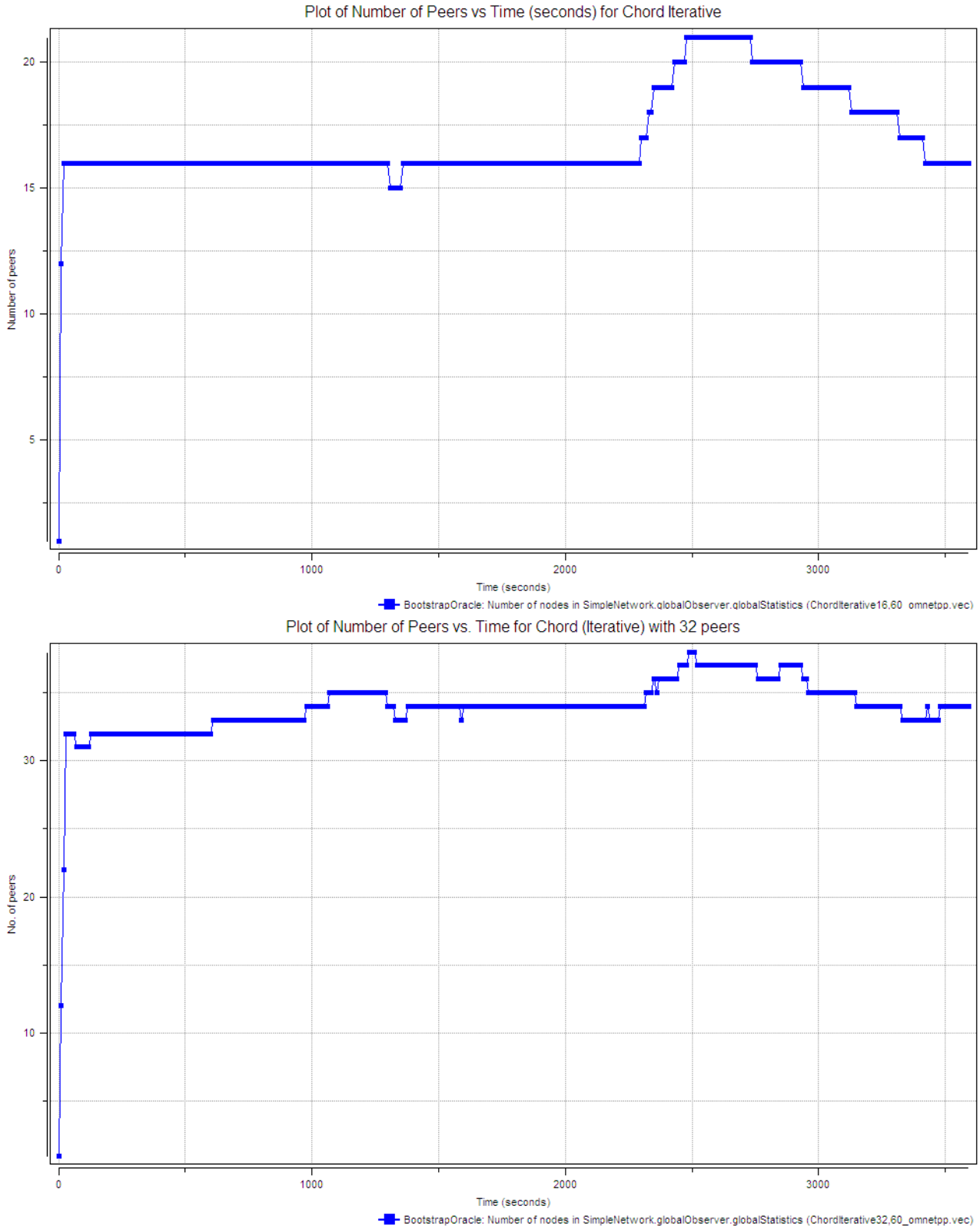
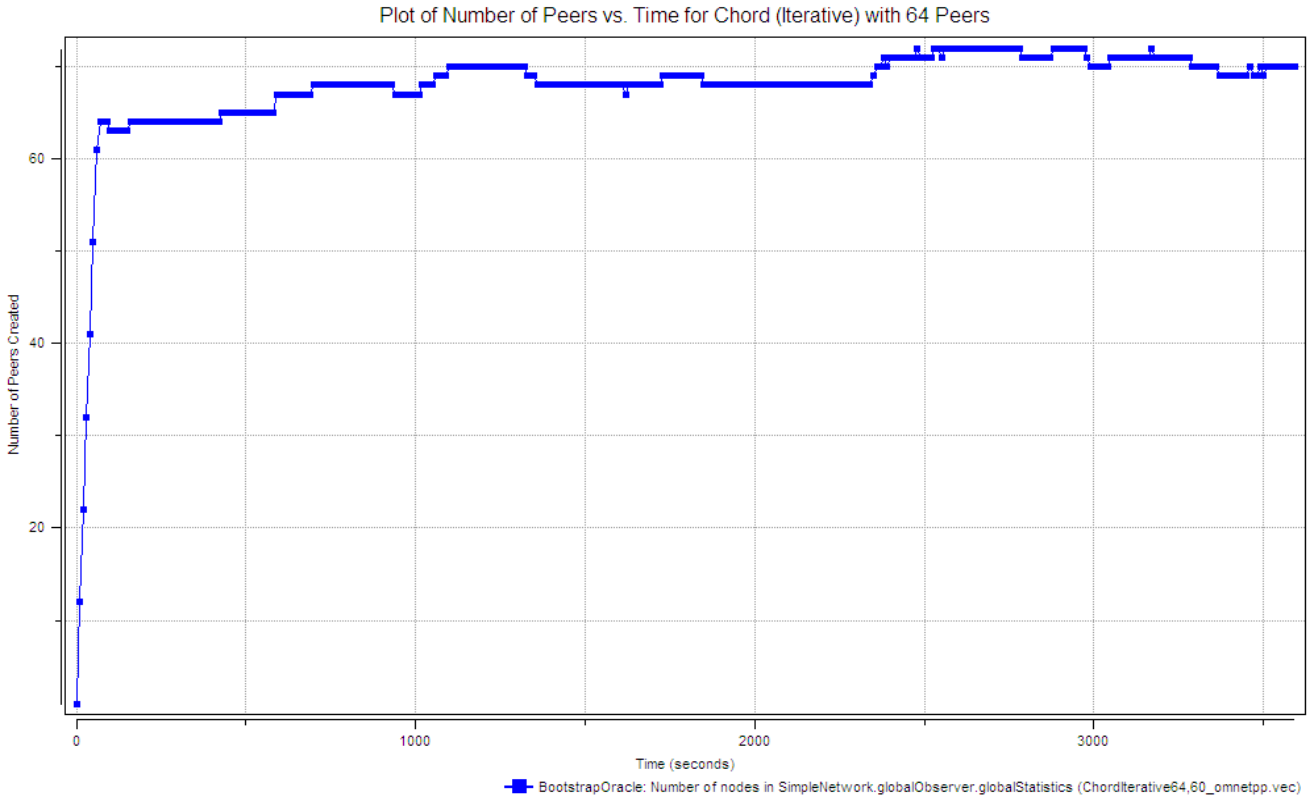


Figure 8(a) Population of Peers vs. Time with 16 and 32 peers respectively for Chord (Iterative)
Observation: Longer periods of stable peer numbers with sporadic churn, but not occurring as often as the corresponding graphs for Chord with Recursive routing in Figure 7(a).



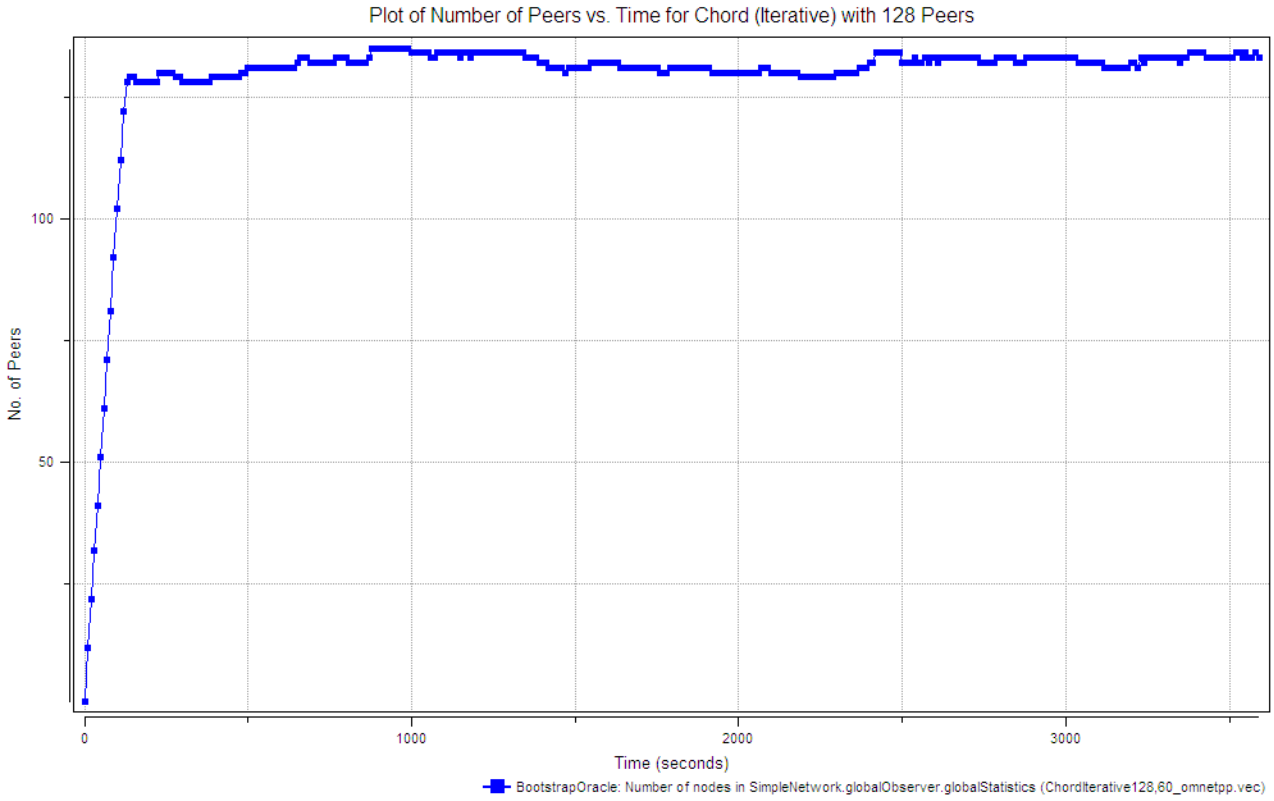


Figure 8(b) Population of Peers vs. Time with 64 and 128 peers respectively for Chord (Iterative)
Observation: Exponential population increase results in a greater rate of churn, but also seems to indicate more convergence as seen in the figure with 128 peer nodes.

Trace Comparison of Node Population vs. Time for Group 3 (Chord Iterative on an IPv4 network)

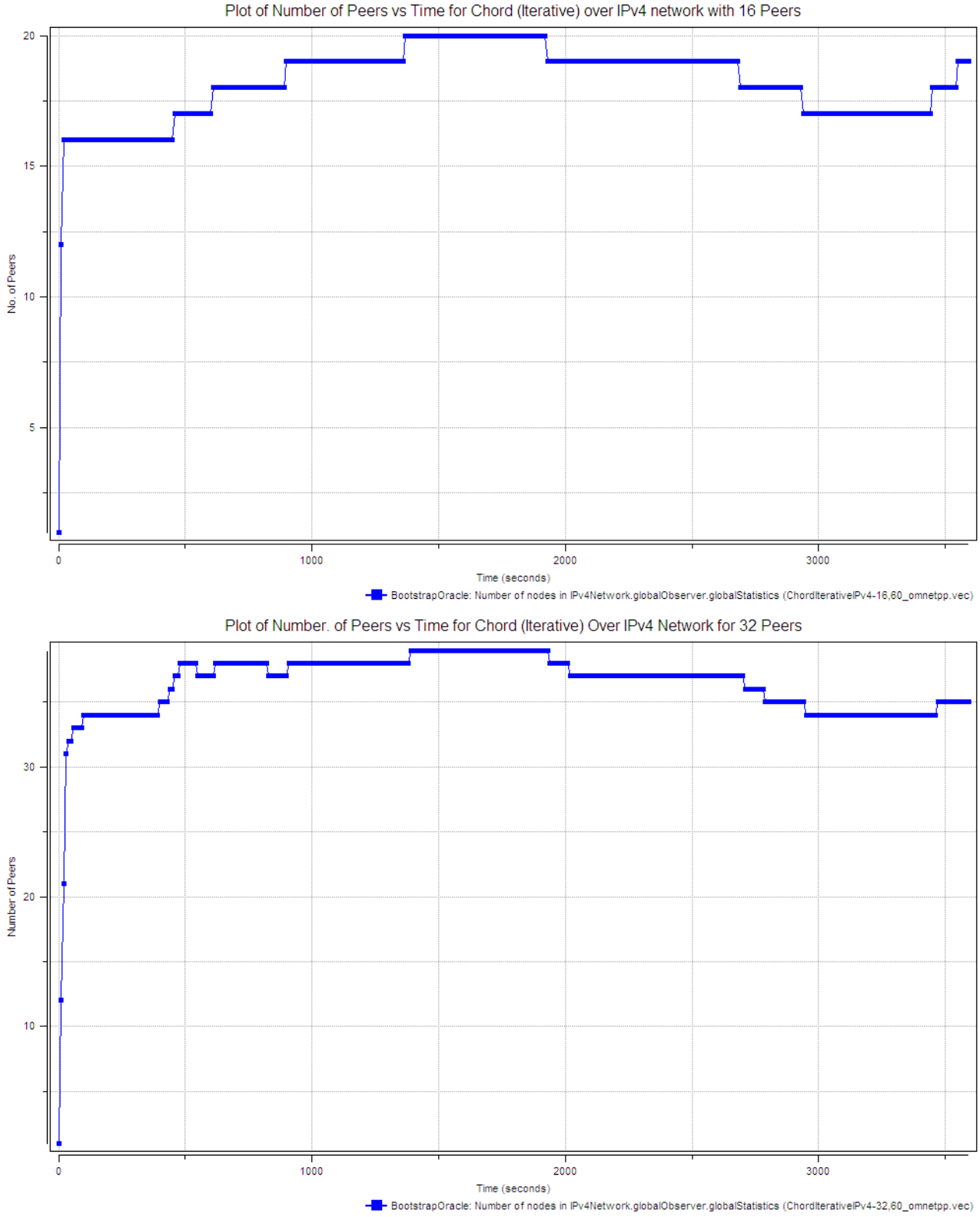


Figure 9(a) Population of Peers vs. Time with 16 and 32 peers respectively for Chord (Iterative, IPv4)
Observation: Large variations from initially set network size with relatively long periods of stability.

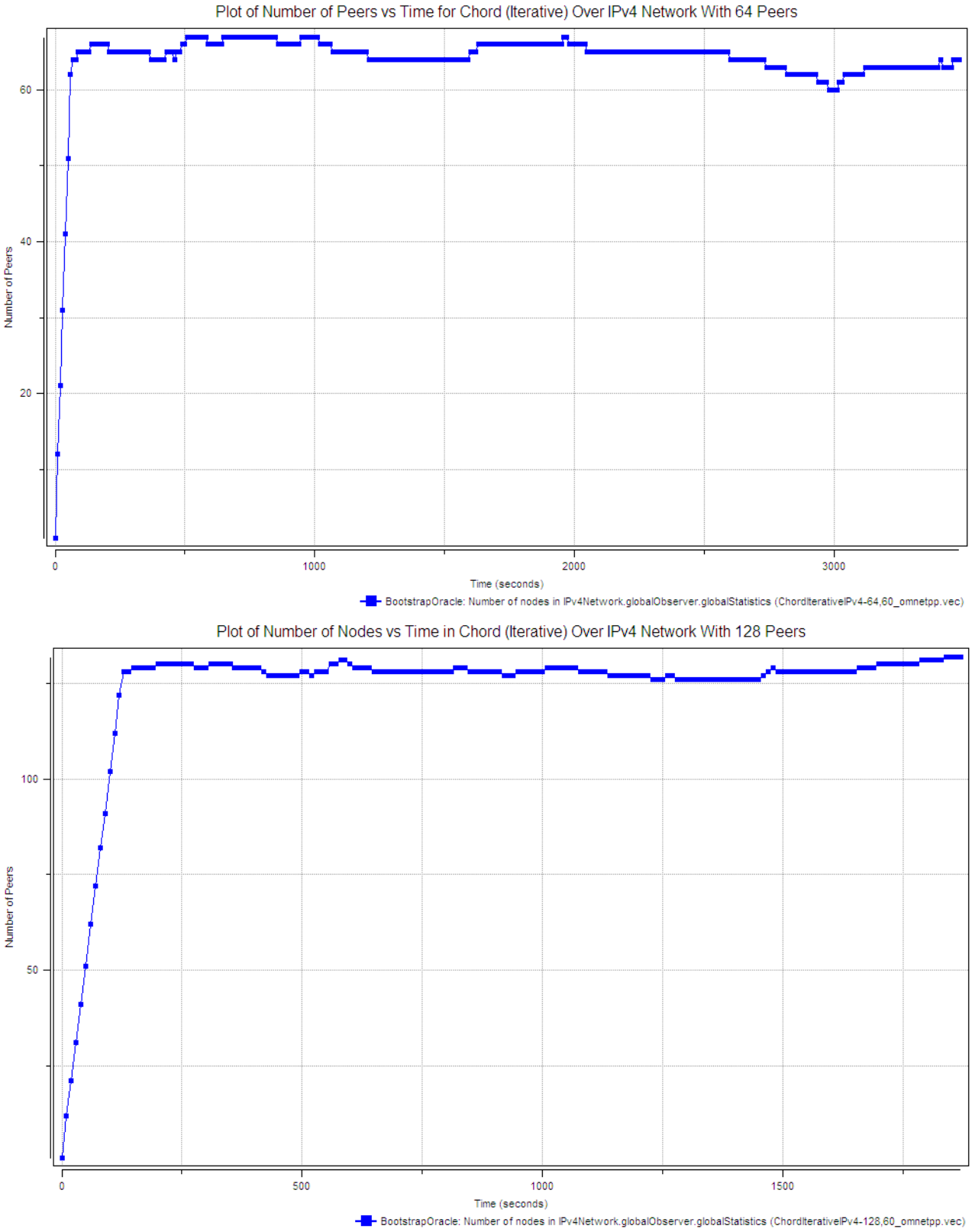
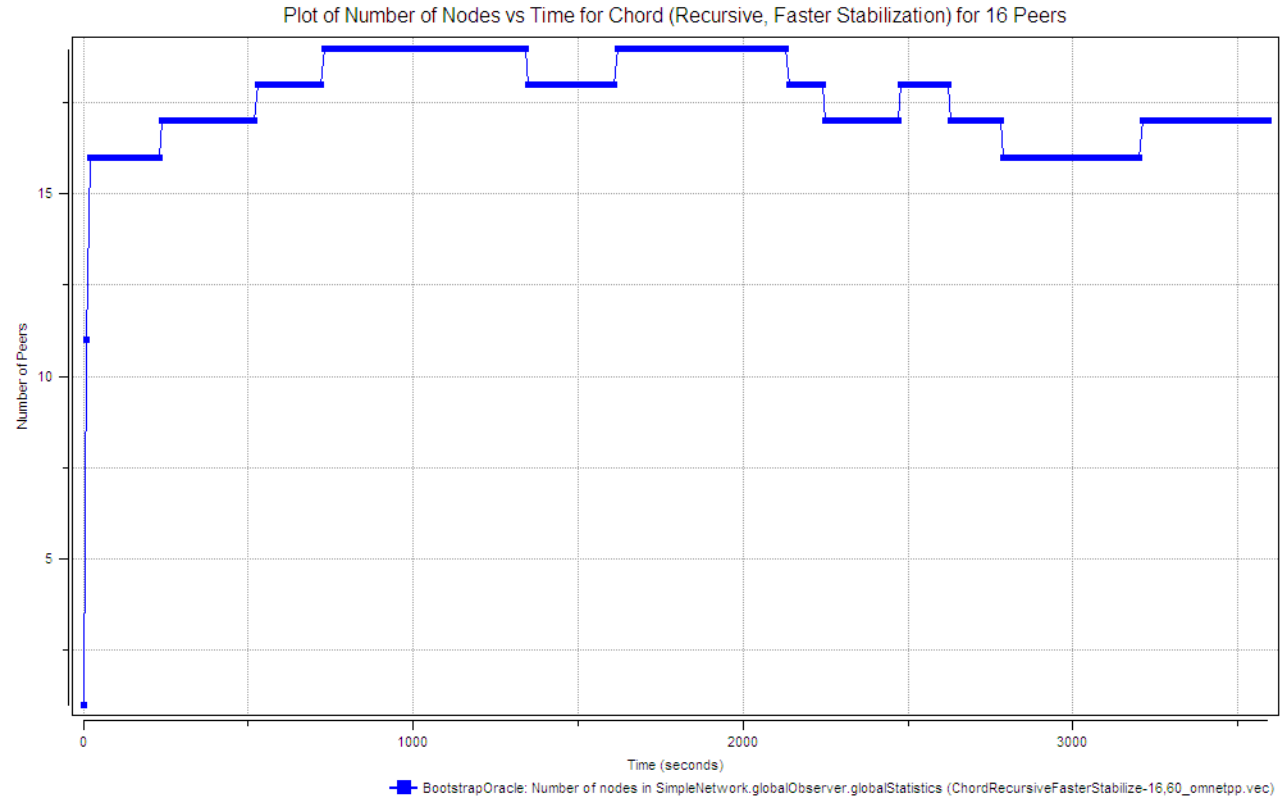


Figure 9(b) Population of Peers vs. Time with 64 and 128 peers respectively for Chord (Iterative, IPv4)
Observation: Less deviation from initial network size, but increasing churn over shorter intervals. Network with 128 peers seems to indicate convergence of churn.

Trace Comparison of Node Population vs. Time for Group 4 (Chord Recursive, faster stabilization)



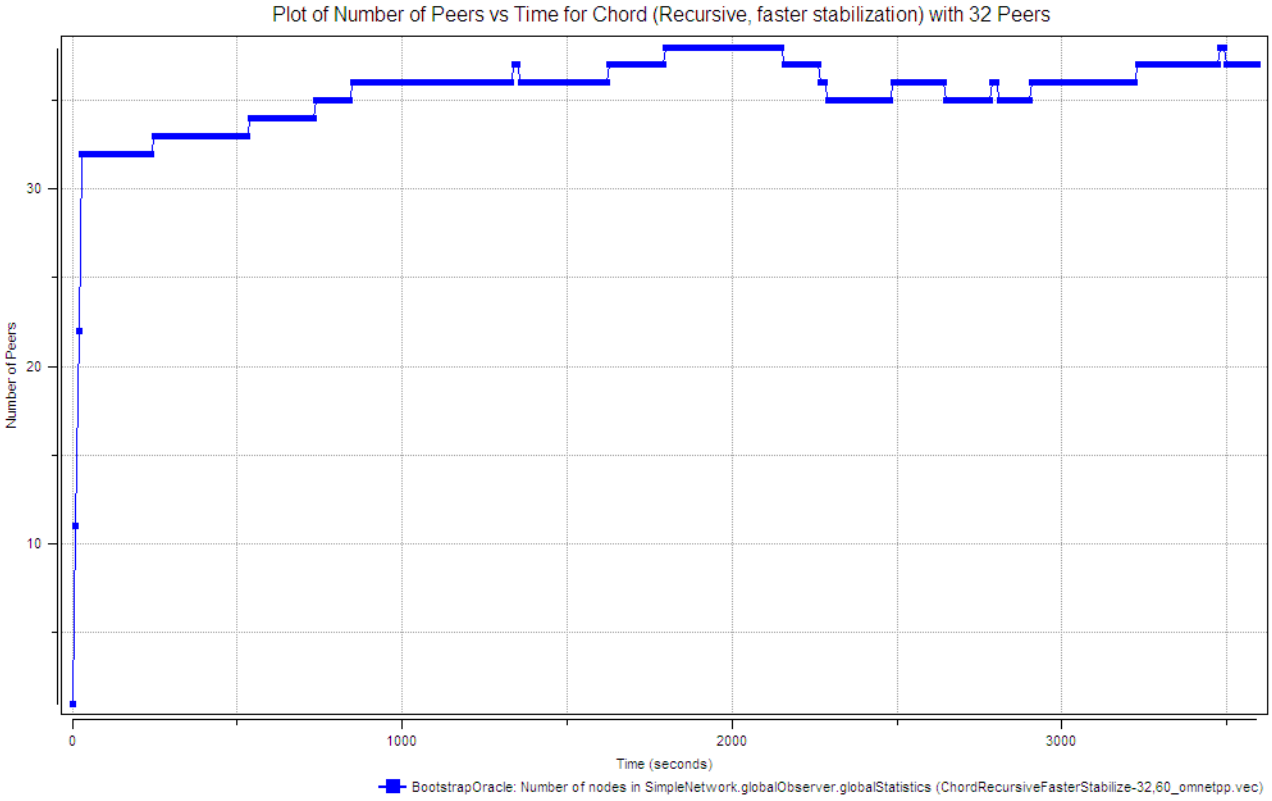


Figure 10(a) Population of Peers vs. Time with 16 and 32 peers respectively for Chord (Recursive, fast stab.)

Observation: Decreased periods of constant size and larger deviations from initial numbers relative to initial Chord Recursive simulation run of group 1

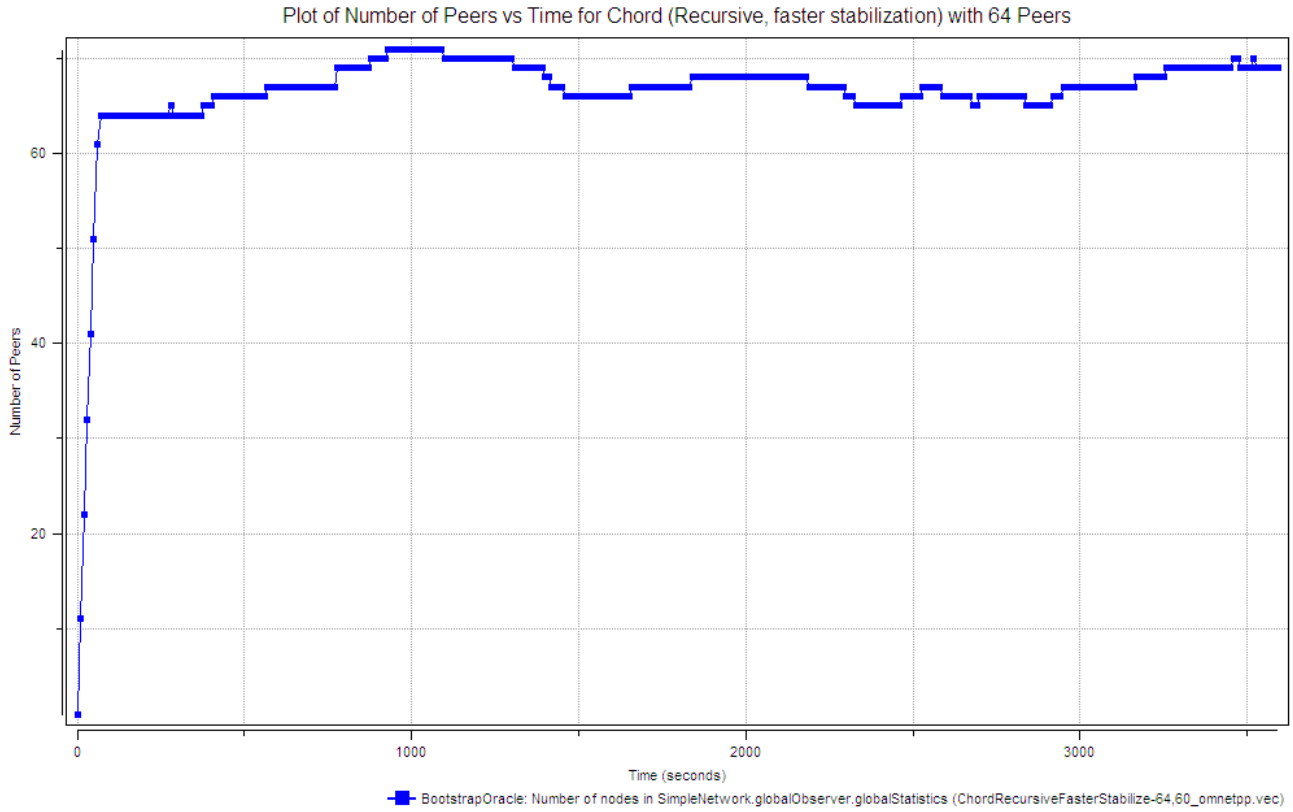


Figure 10(b) Population of Peers vs. Time with 64 and 128 peers respectively for Chord (Recursive, fast stab.)
Observation: With doubling of the peers present in the system, the network seems to be taking longer to move towards convergence, but the regions at which the number of peers remains constant persist for longer periods of time before node arrival or departure moves the network to another state.

Trace Comparison of Group 1 vs. Group 2: Current Delivery Ratio vs. Time

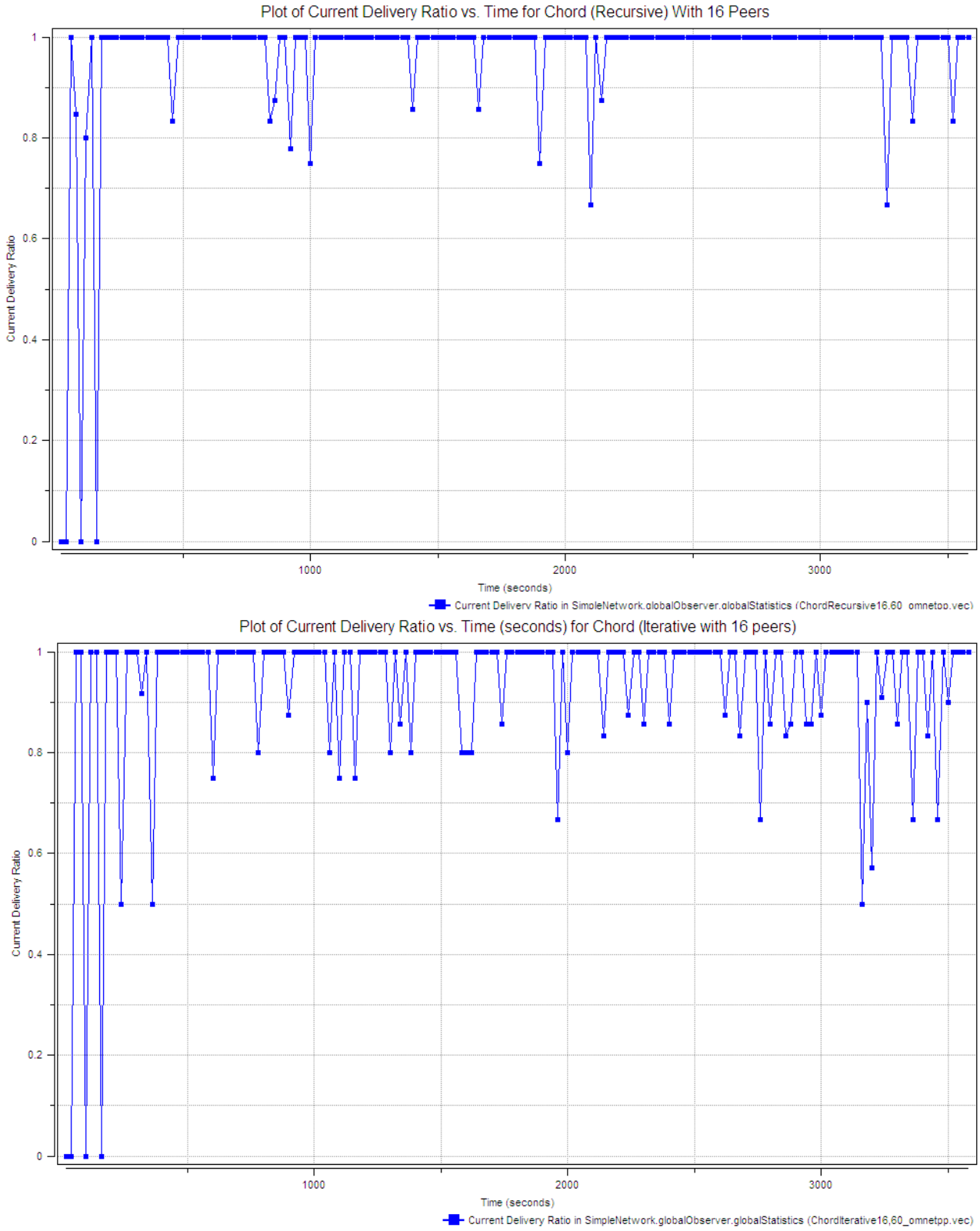


Figure 11(a) Current Delivery Ratio vs. Time for Group 1 (top) and 2 (bottom) with 16 peers

Observation: Larger fluctuations in delivery ratio of iterative routing due to increased traffic; reporting back to inquiring node at each stage of routing procedure is necessary in iterative routing.

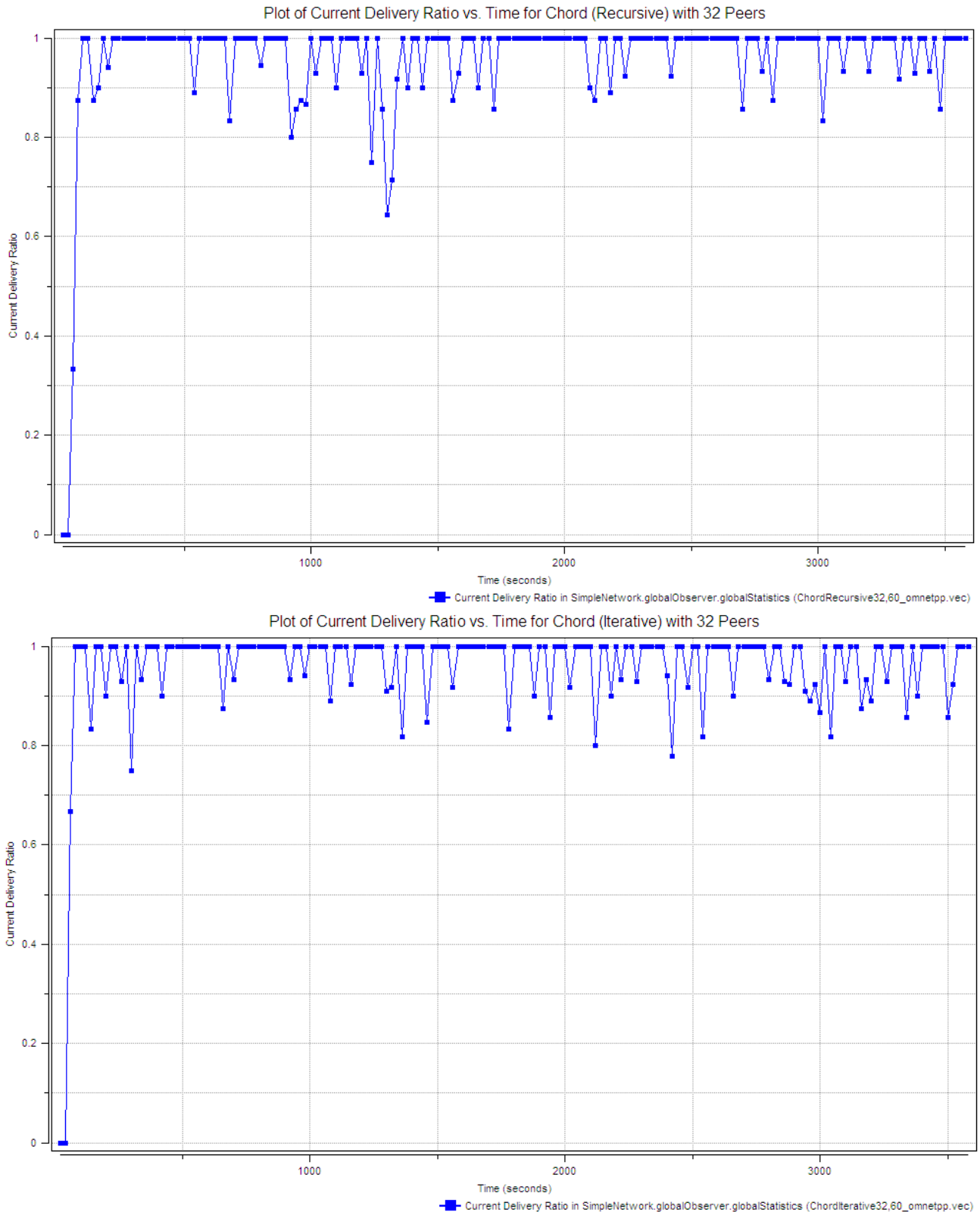


Figure 11(b) Current Delivery Ratio vs. Time for Group 1 (top) and 2 (bottom) with 32 peers

Observation: Again, more frequent fluctuations for iterative routing with exponential growth. However, increased network size provides alternative routing paths, so magnitude of fluctuations are lower than before.

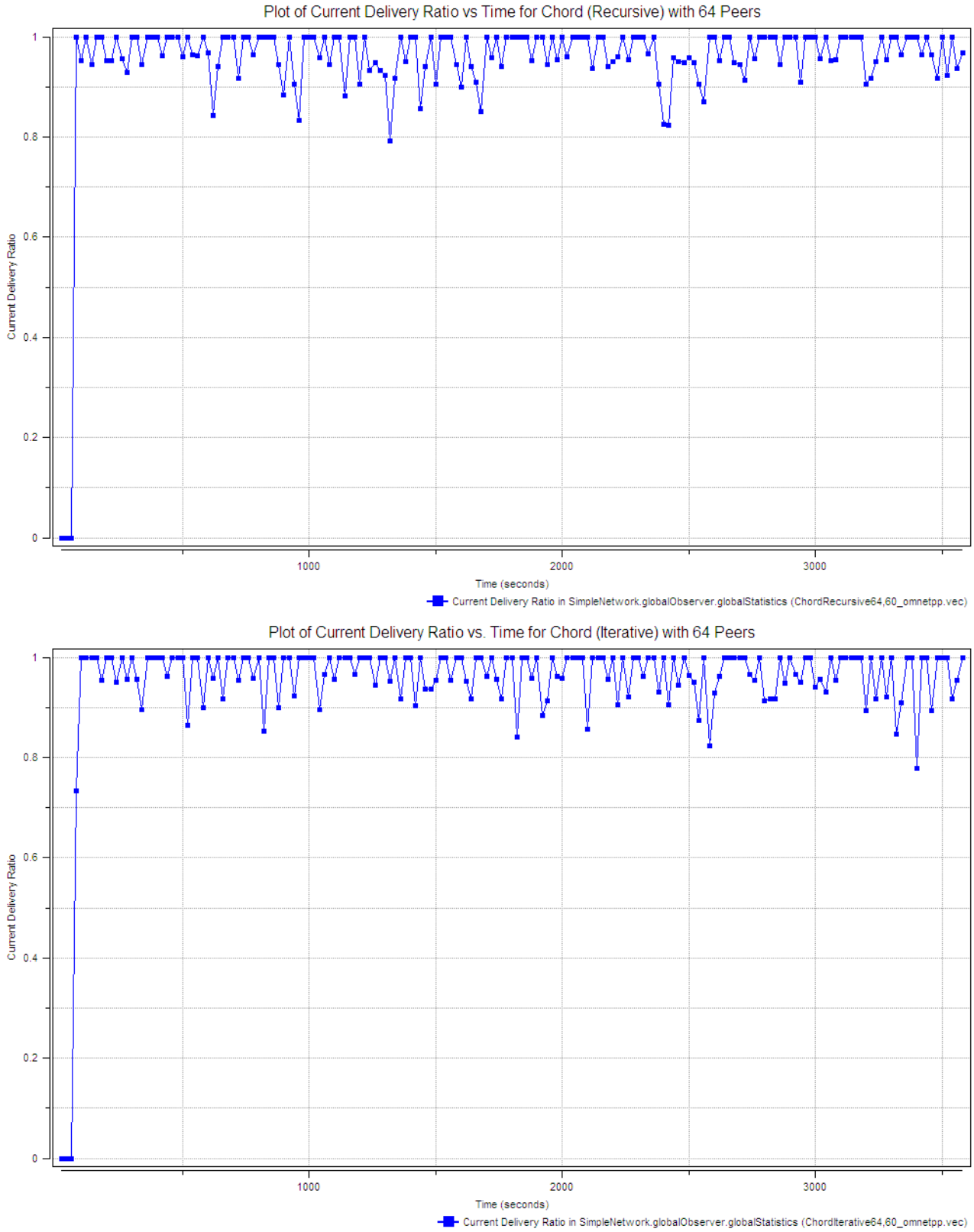


Figure 11(c) Current Delivery Ratio vs. Time for Group 1 (top) and 2 (bottom) with 64 peers

Observation: Again, increased frequency of fluctuation in iterative routing delivery ratio, but magnitude of fluctuation is lower with increased network size providing alternative routing paths, hence successful delivery.

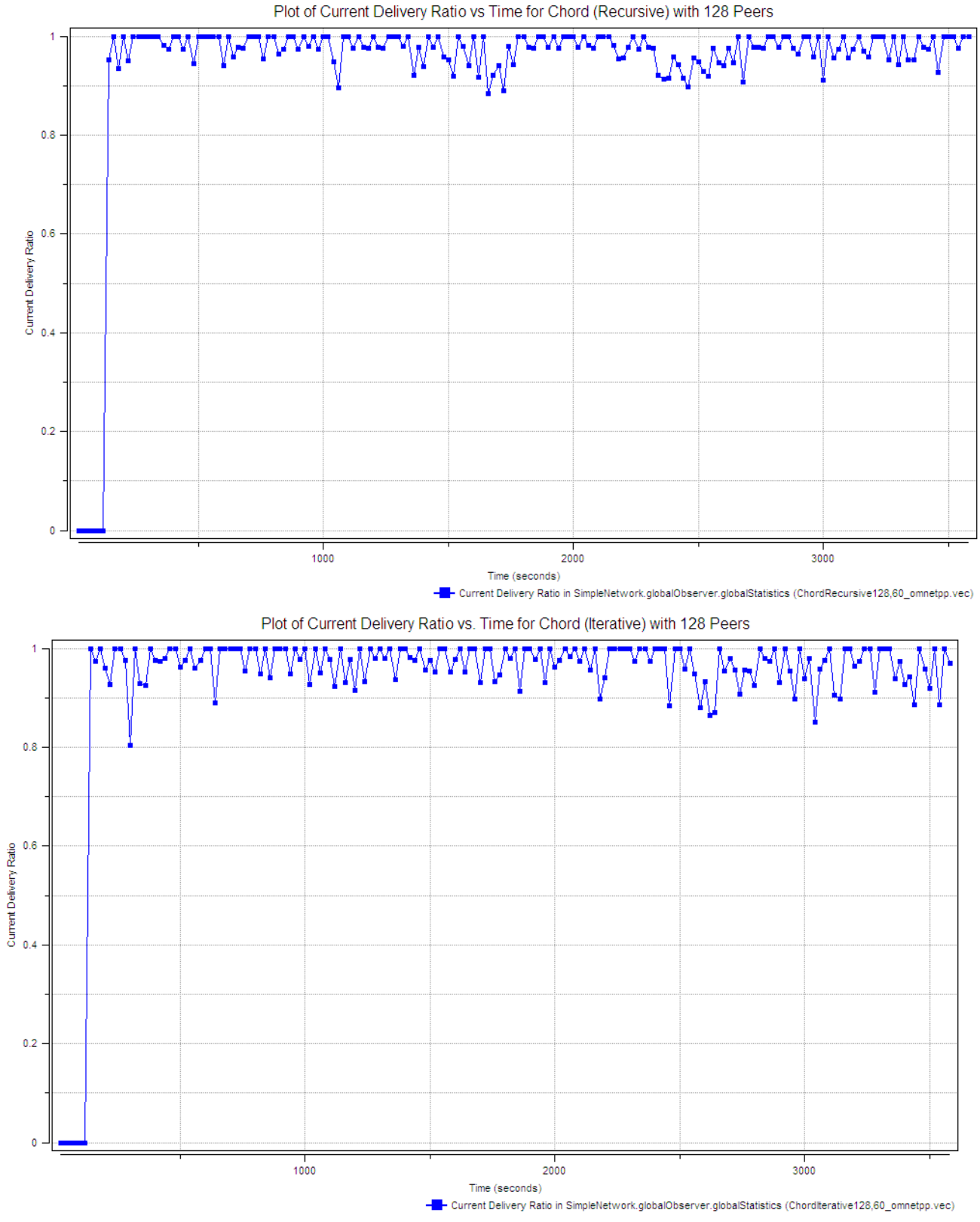


Figure 11(d) Current Delivery Ratio vs. Time for Group 1 (top) and 2 (bottom) with 128 peers
Observation: Frequency of fluctuations even higher due to doubling in network size, but the system seems to be tending toward a level of convergence as the average magnitude of the fluctuations remains approx. same.

Trace Comparison of Group 1 vs. Group 2: Global Hop Count vs. Time

Figure 12(a) Global Hop Count vs. Time for Group 1 (top) and 2 (bottom) with 16 peers

Observation: Wild fluctuations in hop count seen due to churn and consequent delay during stabilization & finger-table fixing procedure. Recursive routing remains bounded upto 3 hops; whereas iterative jumps to 4.

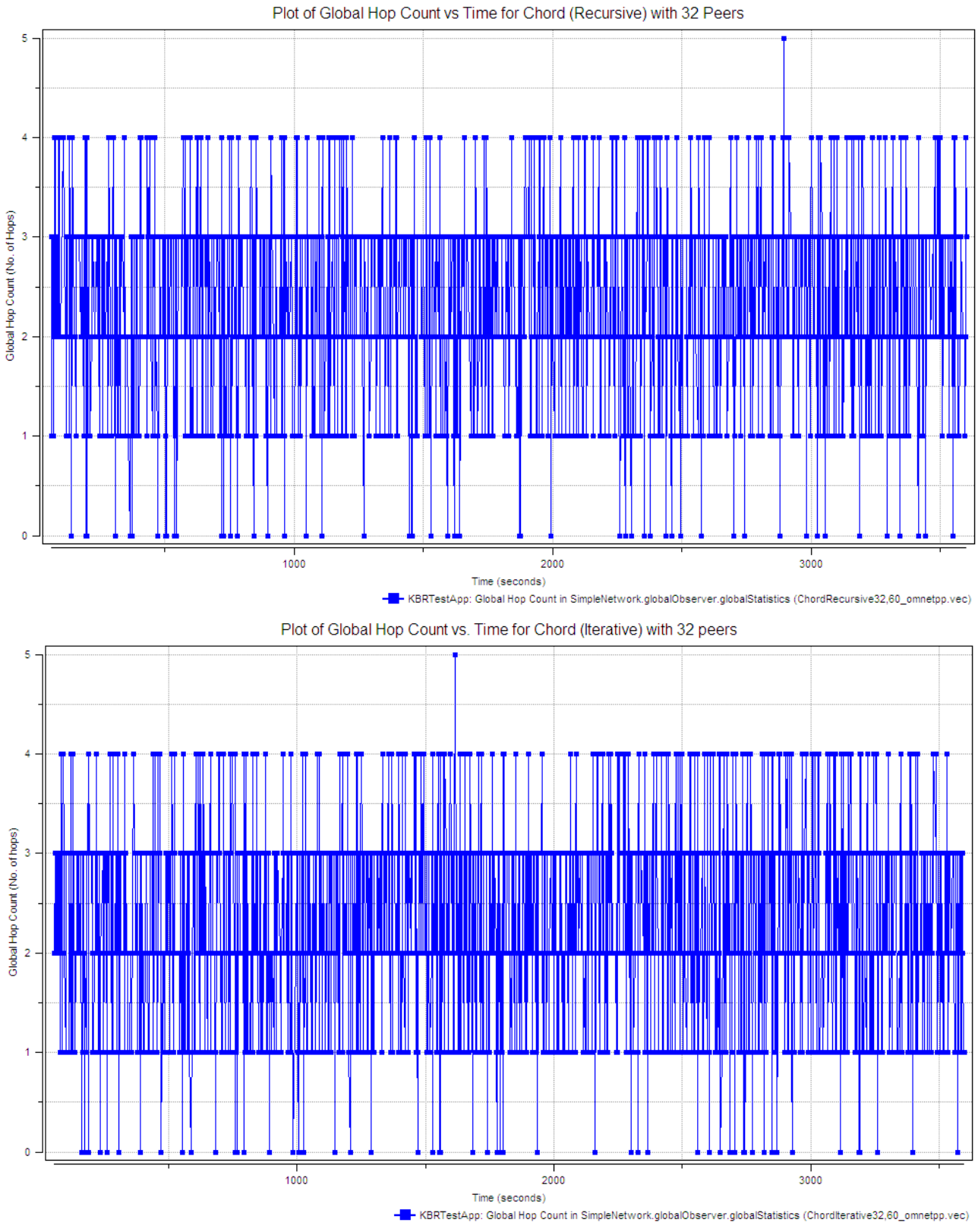


Figure 12(b) Global Hop Count vs. Time for Group 1 (top) and 2 (bottom) with 32 peers

Observation: Doubling of network size results in roughly similar behavior; global average of number of hops jumps to a max. of 5 as more signaling overhead is needed to update identifiers with increasing churn.

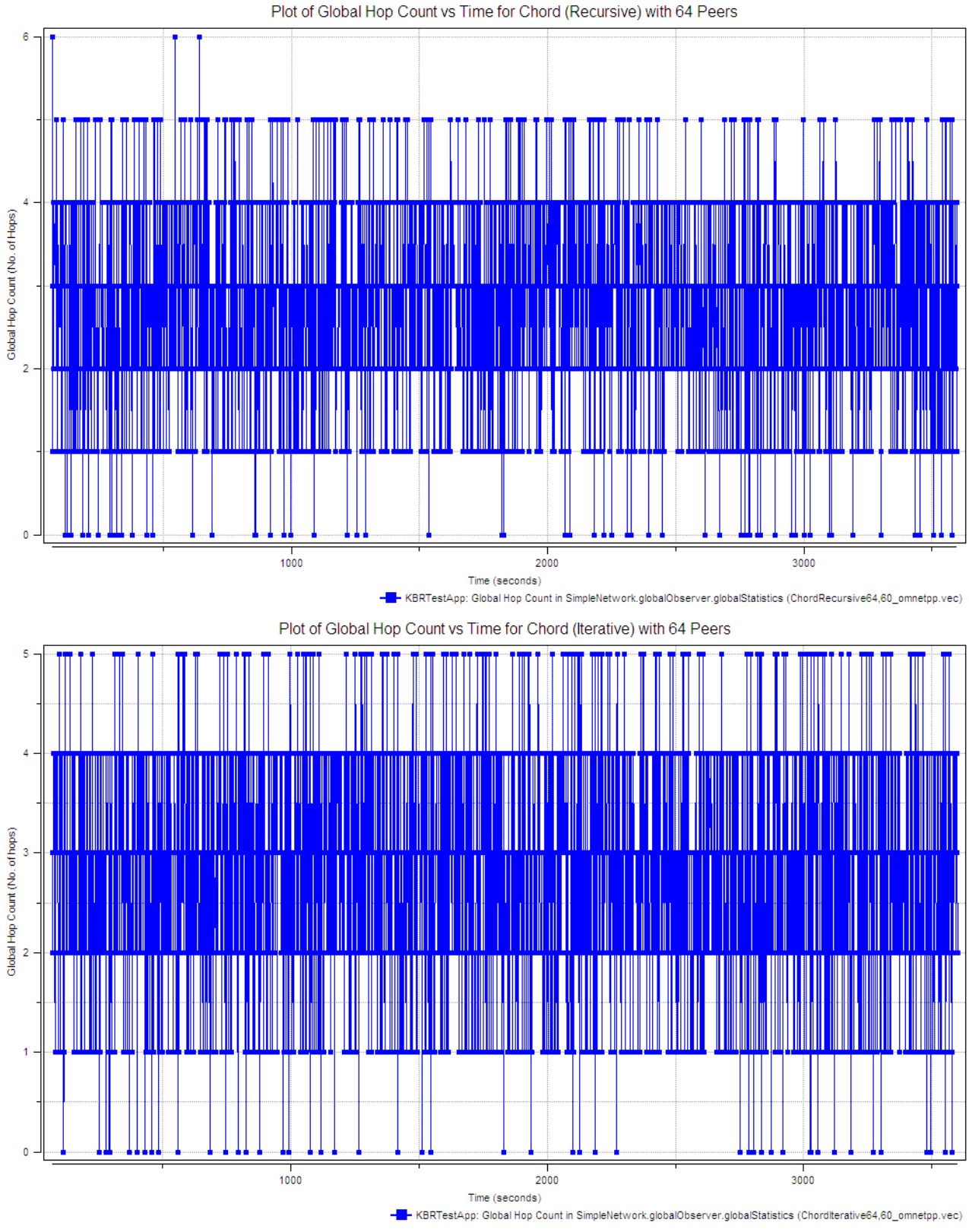


Figure 12(c) Global Hop Count vs. Time for Group 1 (top) and 2 (bottom) with 64 peers

Observation: Exponential growth results in recursive Chord requiring an average of 6 hops occasionally. Iterative remains strongly bounded at 5 hops, as reporting back to inquiring node results in more correct routing information.

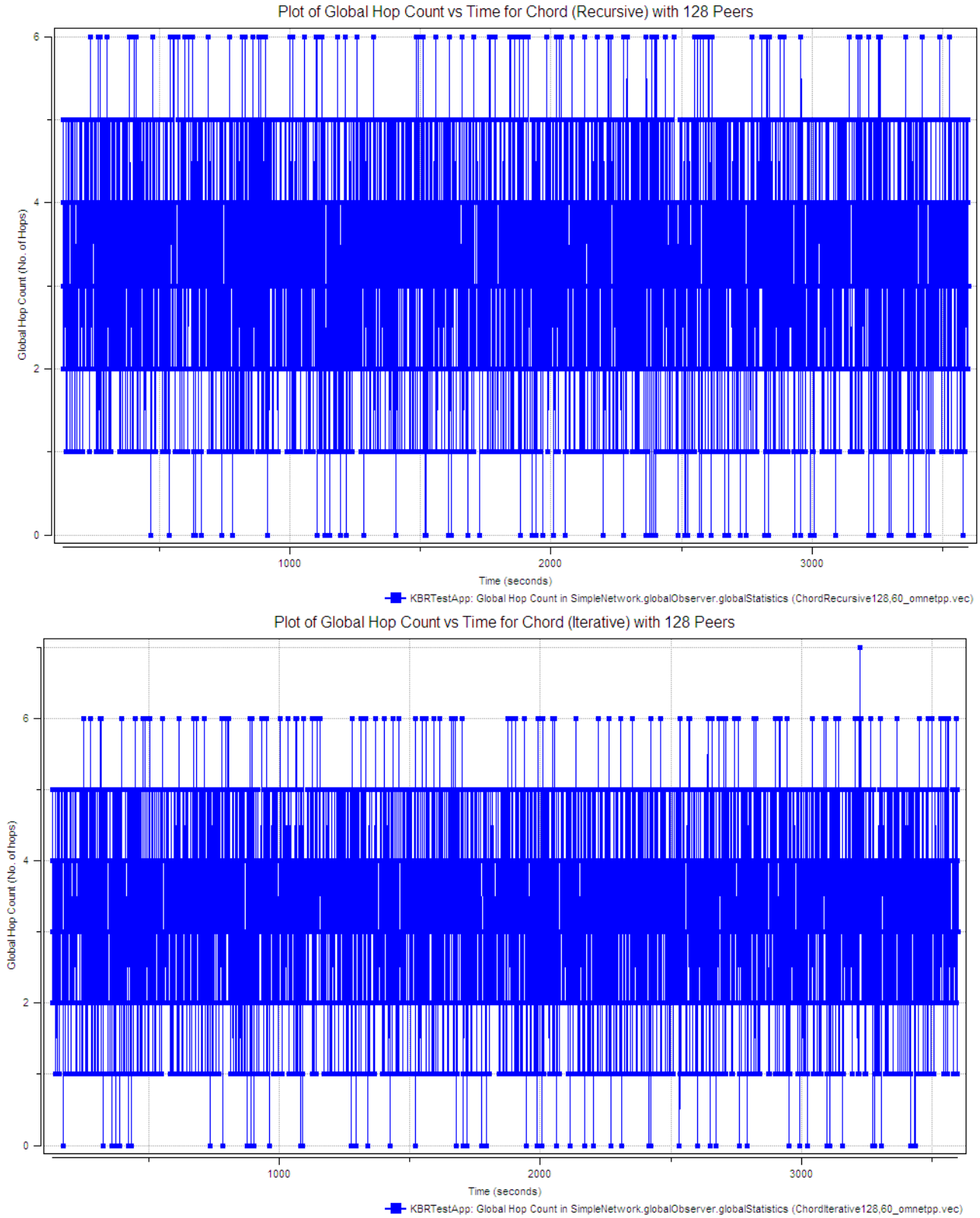


Figure 12(d) Global Hop Count vs. Time for Group 1 (top) and 2 (bottom) with 128 peers

Observation: At 128 peers, both recursive and iterative Chord frequently require an average of 6 hops in order to reach the recipient. Hop count in iterative Chord jumps to 7 hops once.

Trace Comparison of Group 2 vs. Group 3: Current Delivery Ratio vs. Time

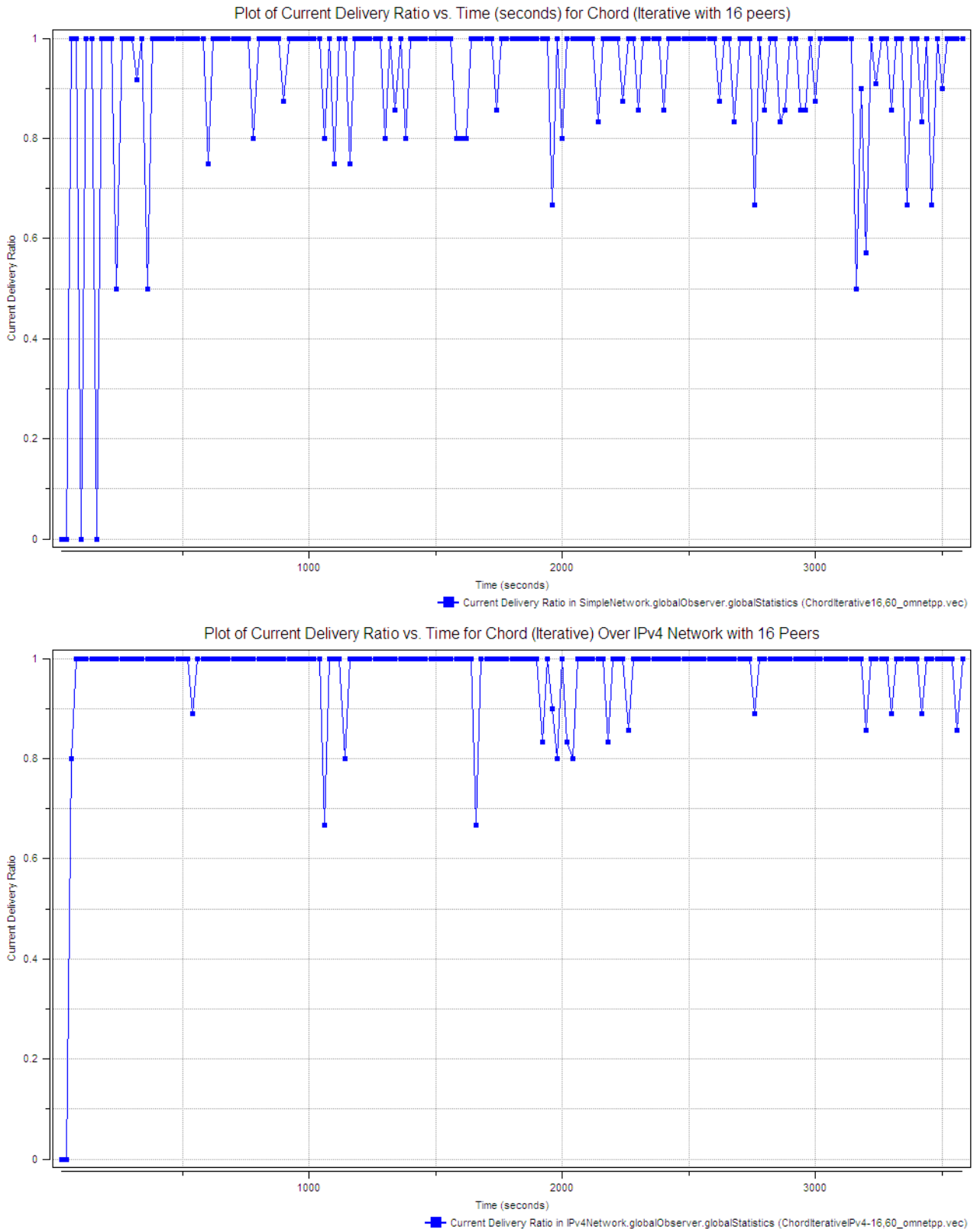


Figure 13(a) Current Delivery Ratio vs. Time for Group 2 (top) and 3 (bottom) with 16 peers
Observation: With 16 peers Iterative Chord over IPv4 network shows faster achievement of 100% delivery, with fewer and smaller magnitude of fluctuations in the percentage of delivered messages.

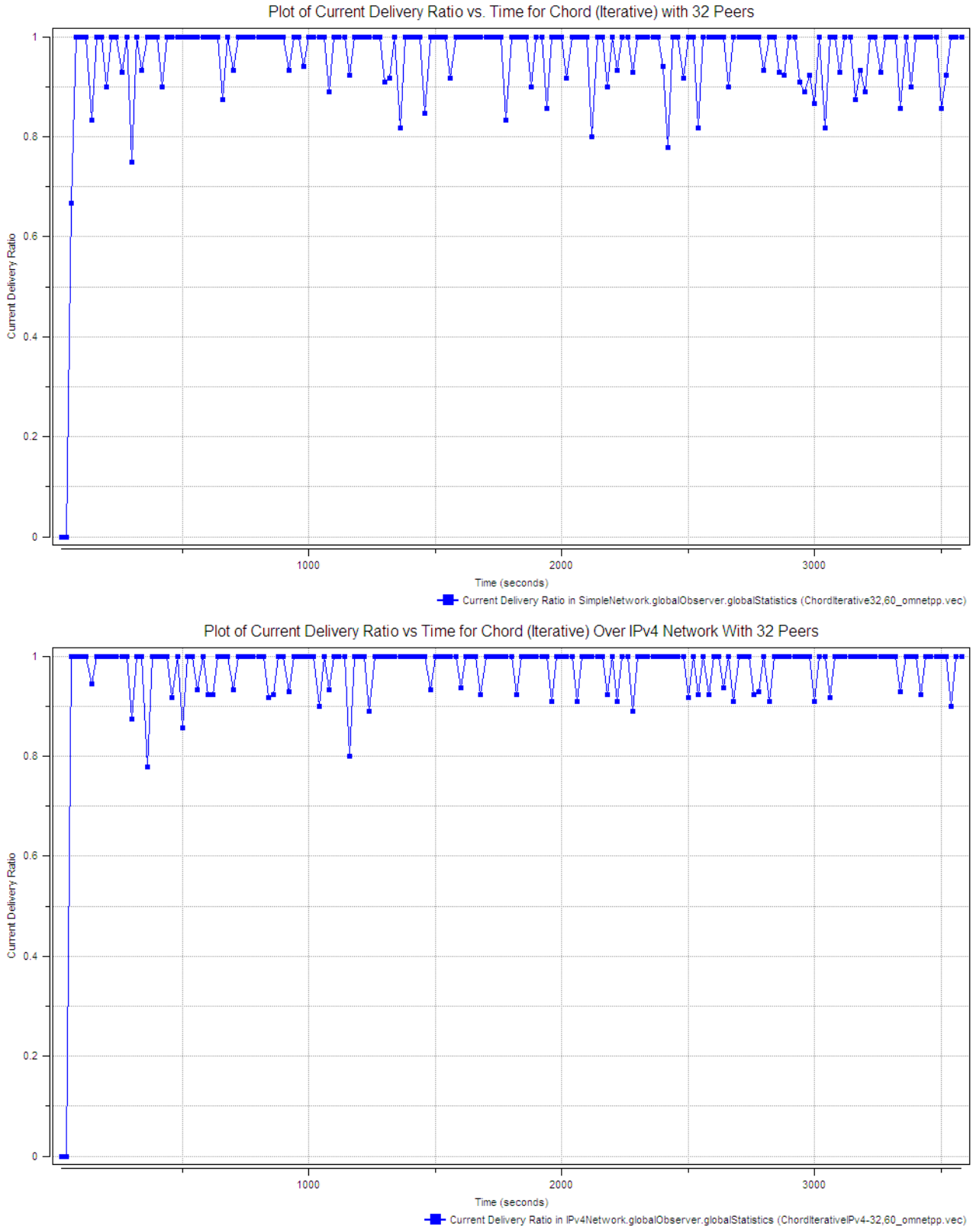


Figure 13(b) Current Delivery Ratio vs. Time for Group 2 (top) and 3 (bottom) with 32 peers
Observation: Although both behaviors may seem similar at this point, Chord over IPv4 still exhibits more reliability, with less frequent deviations below a 90% delivery ratio.

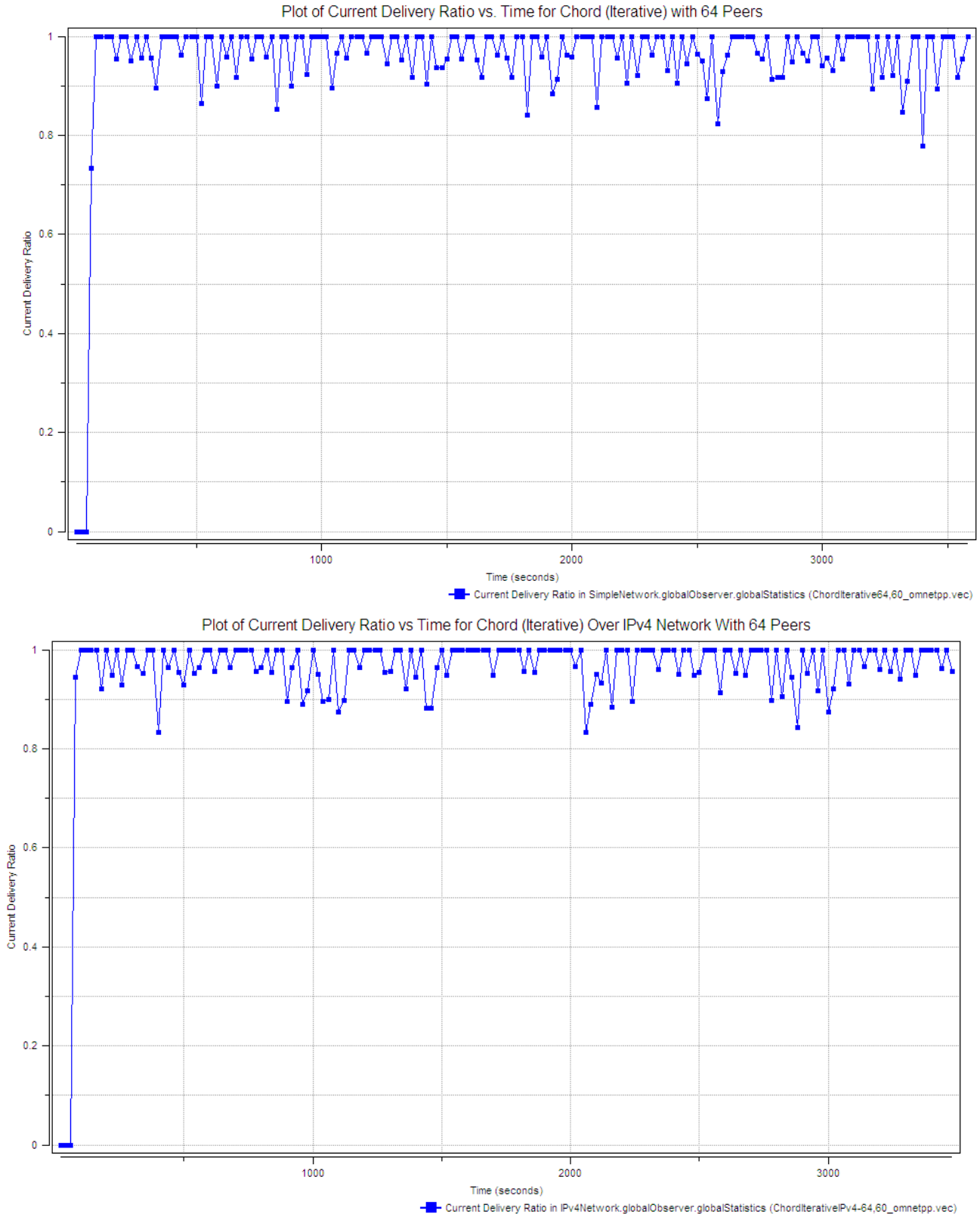


Figure 13(c) Current Delivery Ratio vs. Time for Group 2 (top) and 3 (bottom) with 64 peers

Observation: Further doubling in network size results in more frequent deviations from the 100% mark, though both networks still manage to maintain a reasonably consistent delivery ratio of above 90% most of the time.

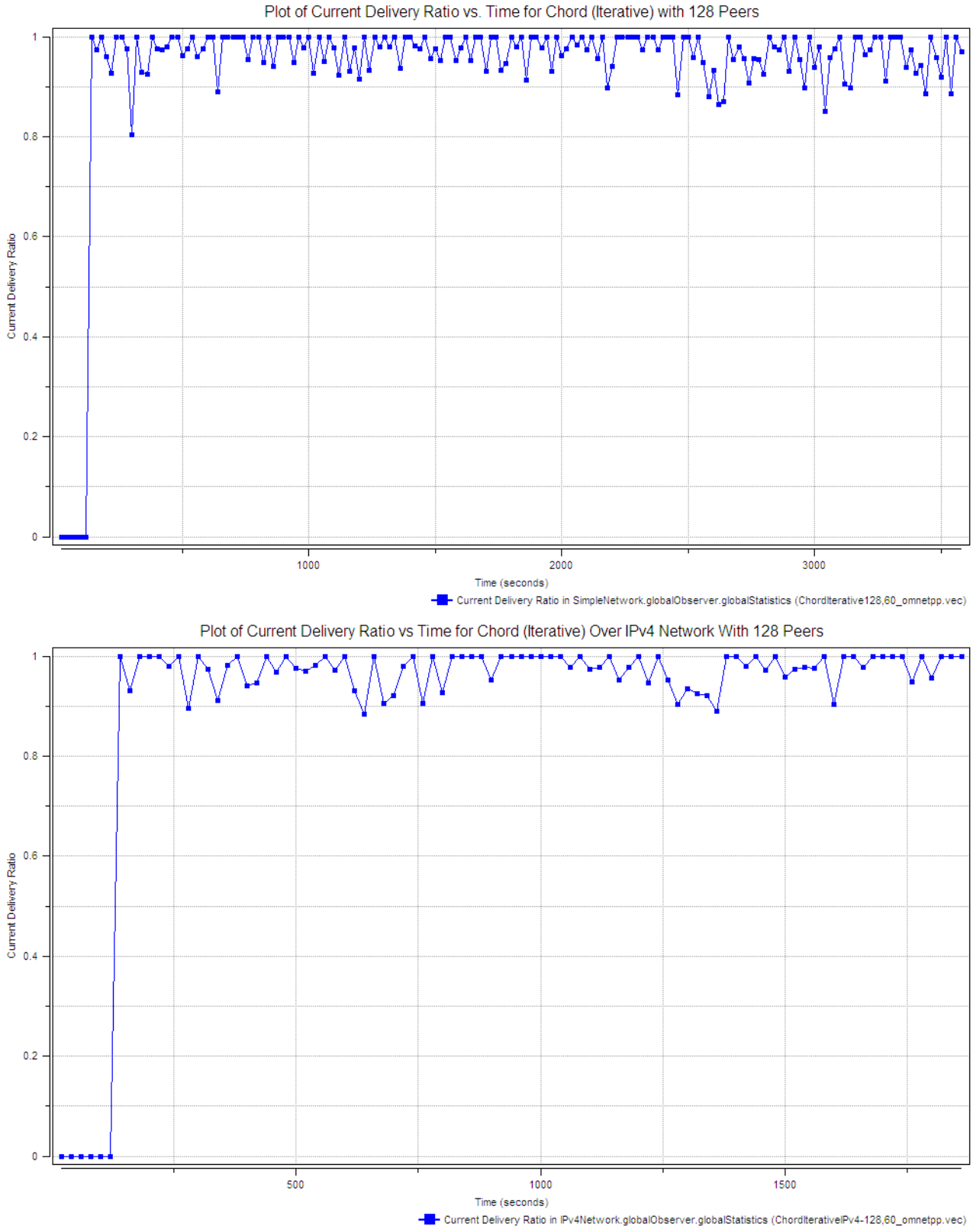


Figure 13(d) Current Delivery Ratio vs. Time for Group 2 (top) and 3 (bottom) with 128 peers
Observation: At the final simulation limit of 128 peers, both networks manage to maintain a delivery ratio above 90% most of the time, though iterative Chord over an IPv4 network still manages fewer fluctuations.

Trace Comparison of Group 2 vs. Group 3: Global Hop Count vs. Time

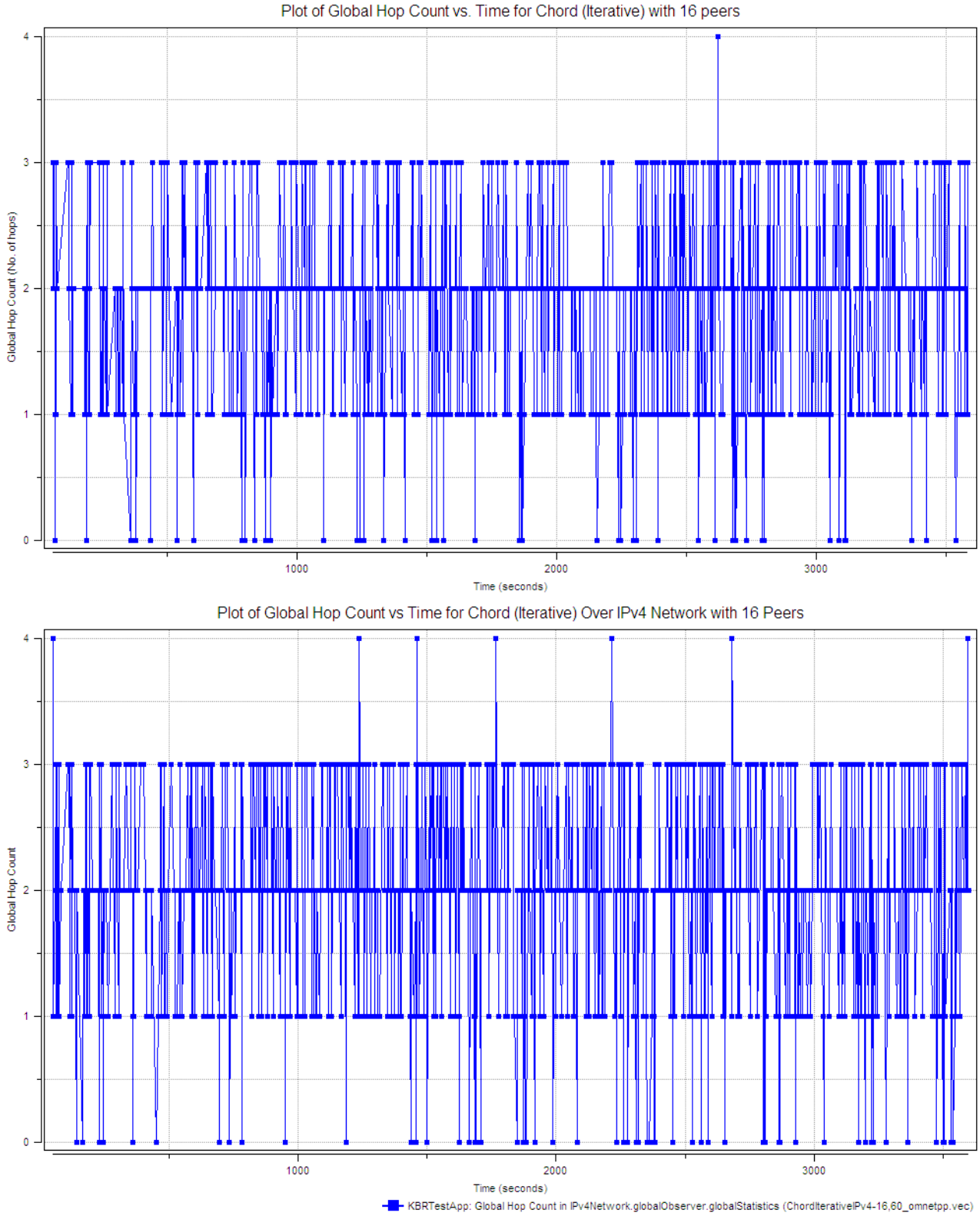


Figure 14(a) Global Hop Count vs. Time for Group 2 (top) and 3 (bottom) with 16 peers

Observation: Iterative Chord over IPv4 network needs an average of 4 hops more often; all the nodes are evenly clustered around two access routers, limiting the number of alternative routes available.

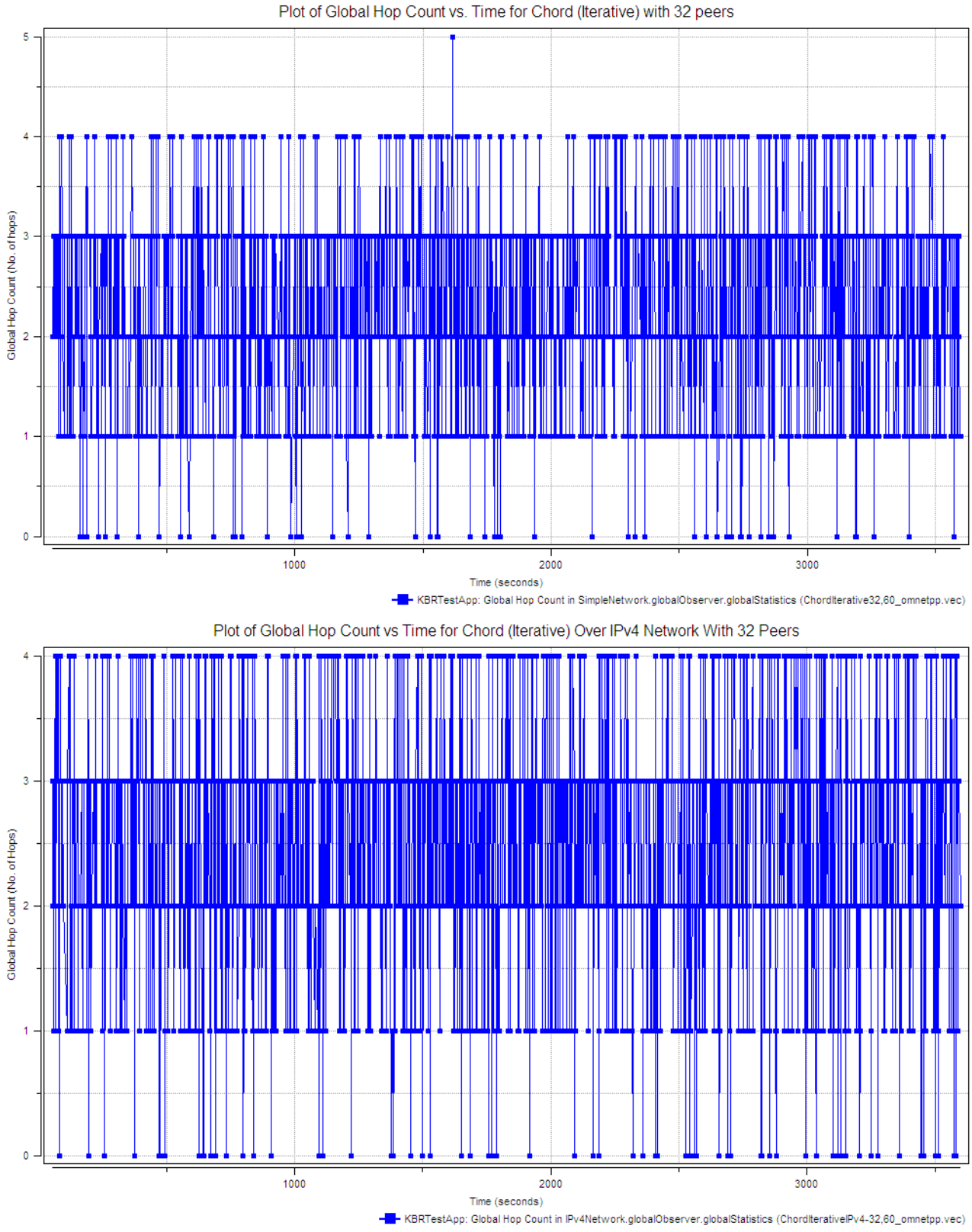


Figure 14(b) Global Hop Count vs. Time for Group 2 (top) and 3 (bottom) with 32 peers

Observation: Maximum hop count in both networks remains mostly at 4 hops; iterative chord on a simple network reaches a maximum of 5 hops once. The access and backbone routers in the IPv4 network provide at least one constant path between the two clusters, resulting in at least one sure alternative.

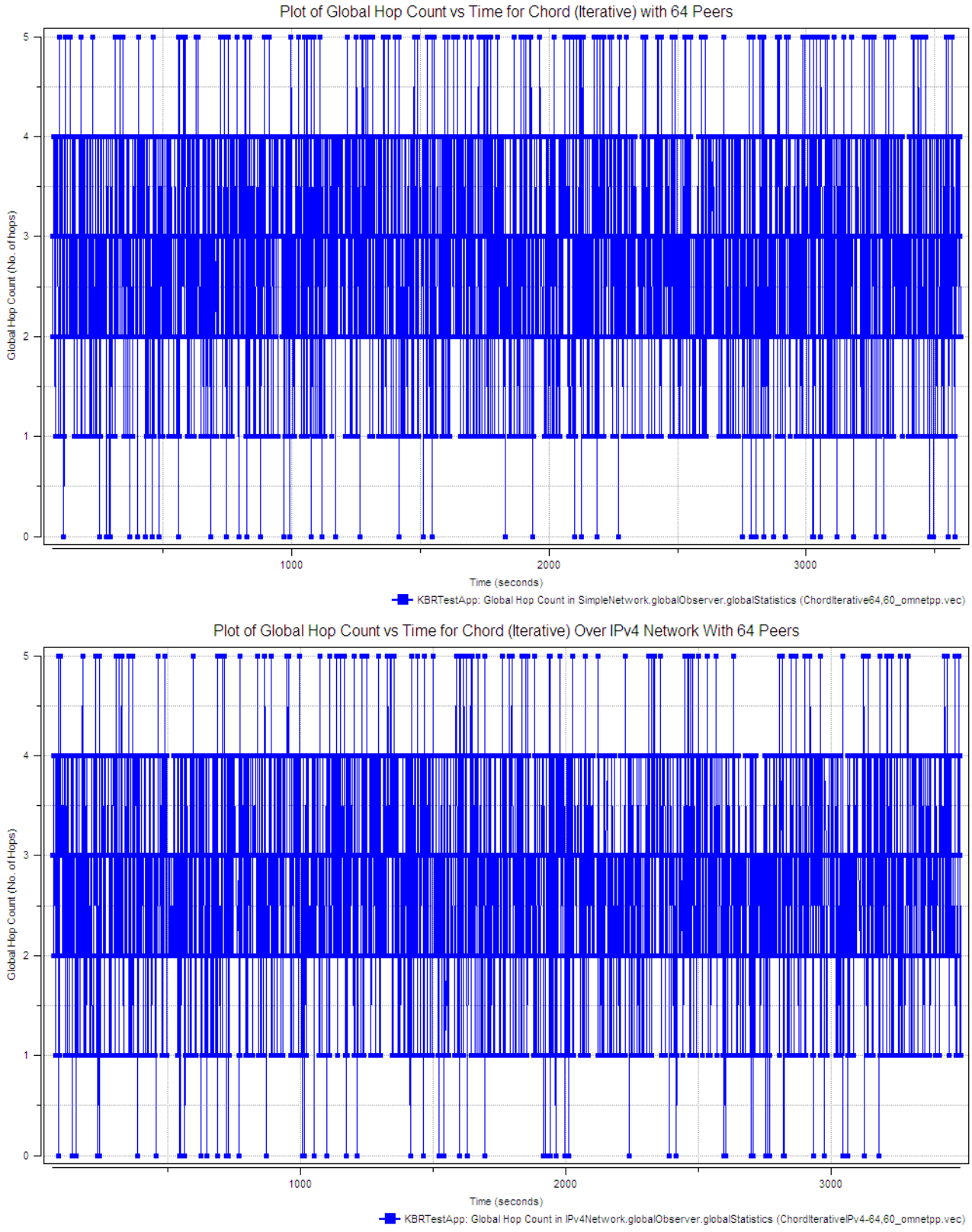


Figure 14(c) Global Hop Count vs. Time for Group 2 (top) and 3 (bottom) with 64 peers

Observation: Doubling network size to 64 peers results in almost similar behavior. Both networks remain mostly bounded between 1 and 4 hops, but require an average of 5 hops much more frequently.

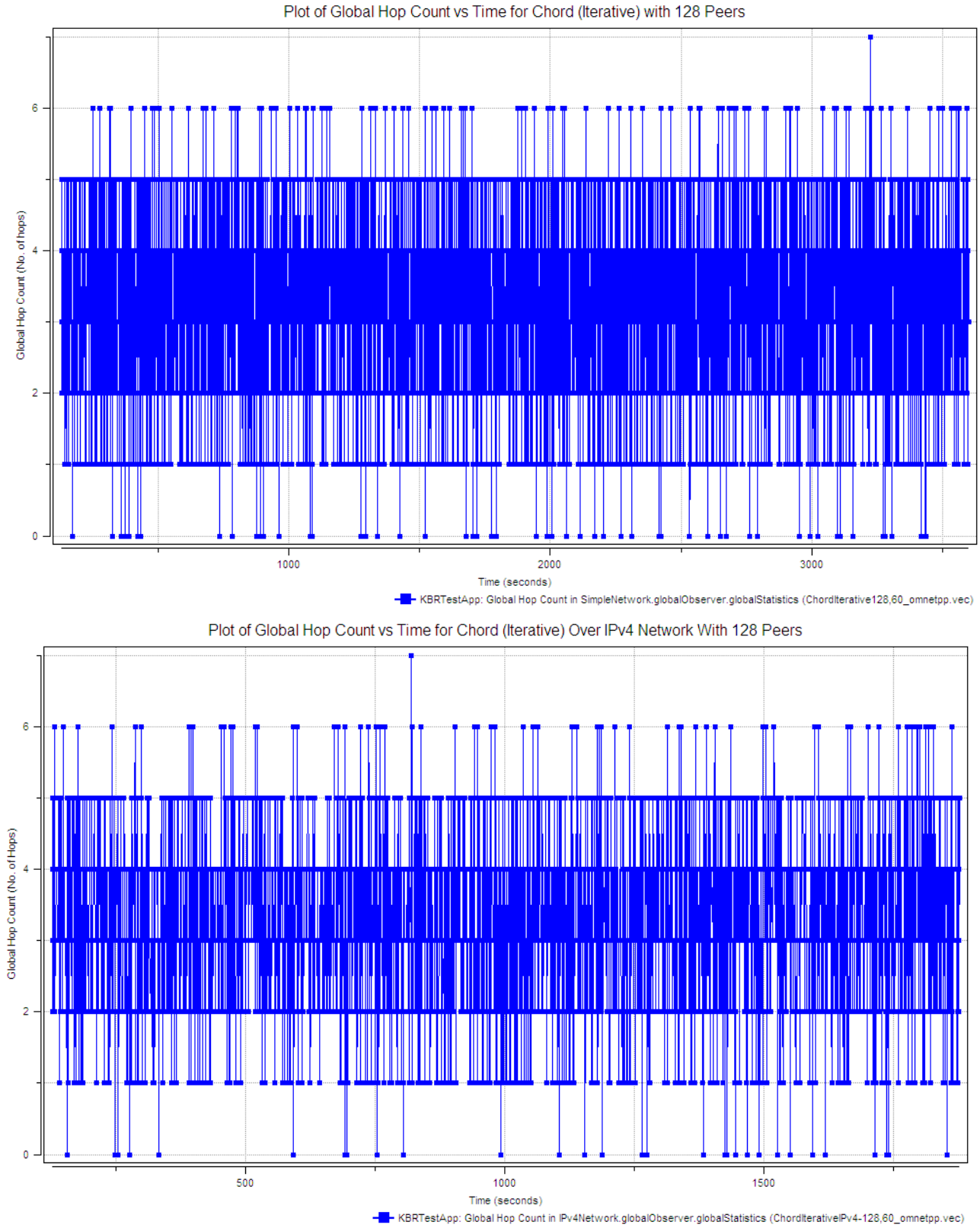


Figure 14(d) Global Hop Count vs. Time for Group 2 (top) and 3 (bottom) with 128 peers

Observation: The final level of exponential growth sees both networks concentrated around a bound of 1 to 5 hops on average, with frequent requirements of 6 hops and once even 7. However, the IPv4 network has less intense fluctuation in the 2 to 4 hop region, due to the capabilities of the IP protocol suite.

Trace Comparison of Group 1 vs. Group 4: Current Delivery Ratio vs. Time

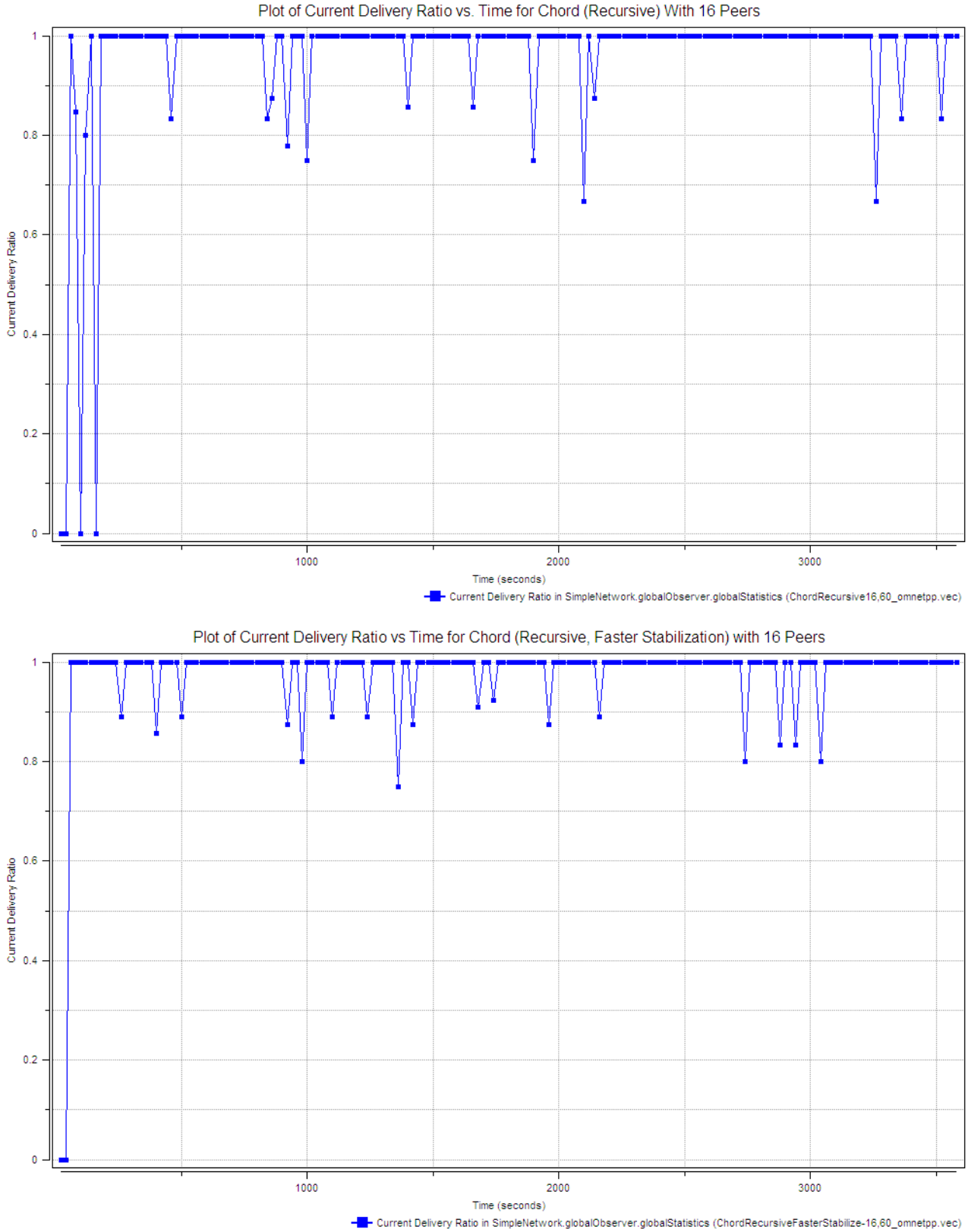


Figure 15(a) Current Delivery Ratio vs. Time for Group 1 (top) and 4 (bottom) with 16 peers

Observation: Cutting the period between stabilization and finger table fixes by half results in faster initial achievement of 100% delivery ratio. The magnitude of occasional fluctuations is also lower

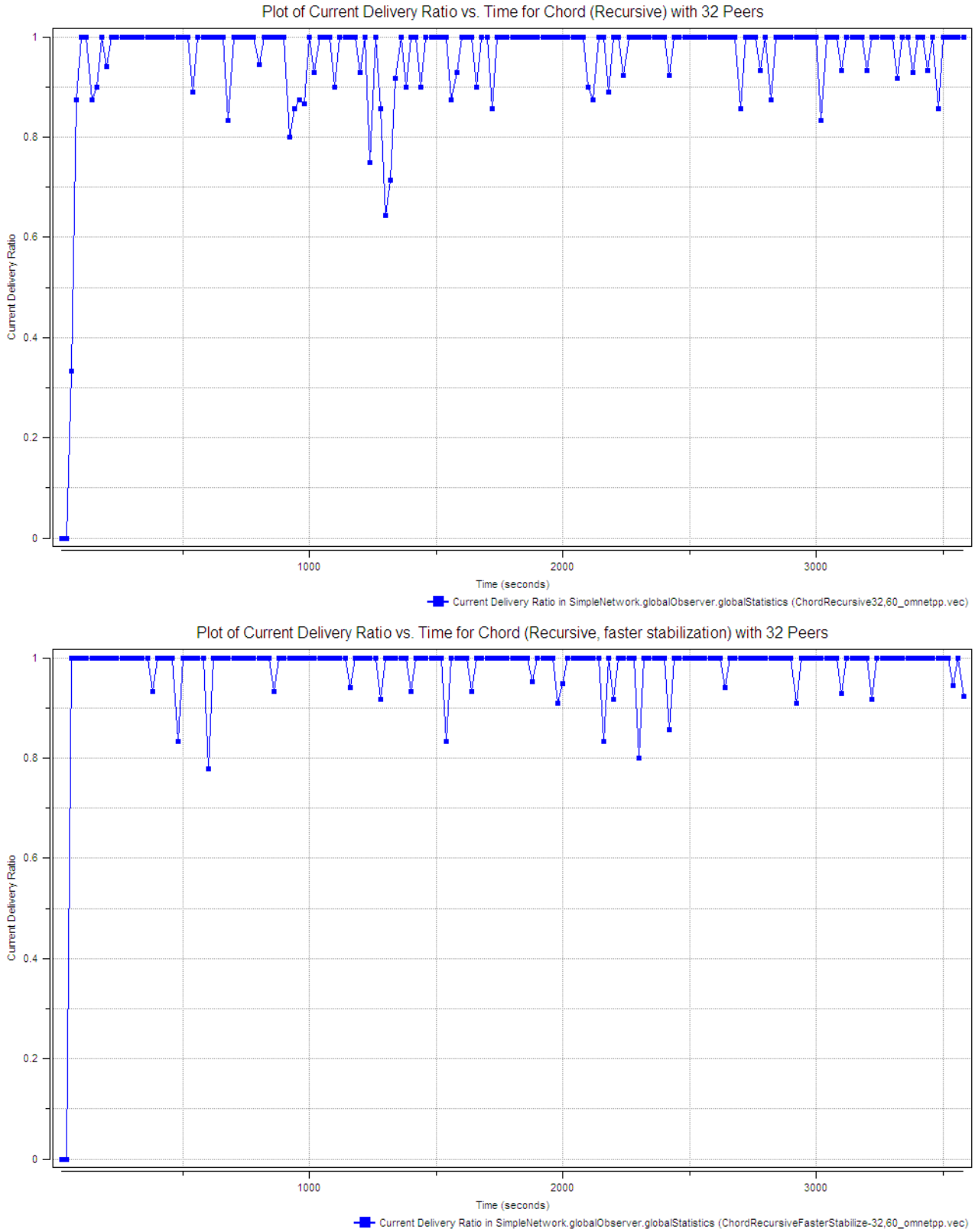


Figure 15(b) Current Delivery Ratio vs. Time for Group 1 (top) and 4 (bottom) with 32 peers
Observation: At 32 peers, faster stabilization results in less frequent fluctuations of lower magnitude; fluctuations go below 90% less often, resulting in greater stability of the network. Delivery remains at 100% longer with exponential growth in Group 4.

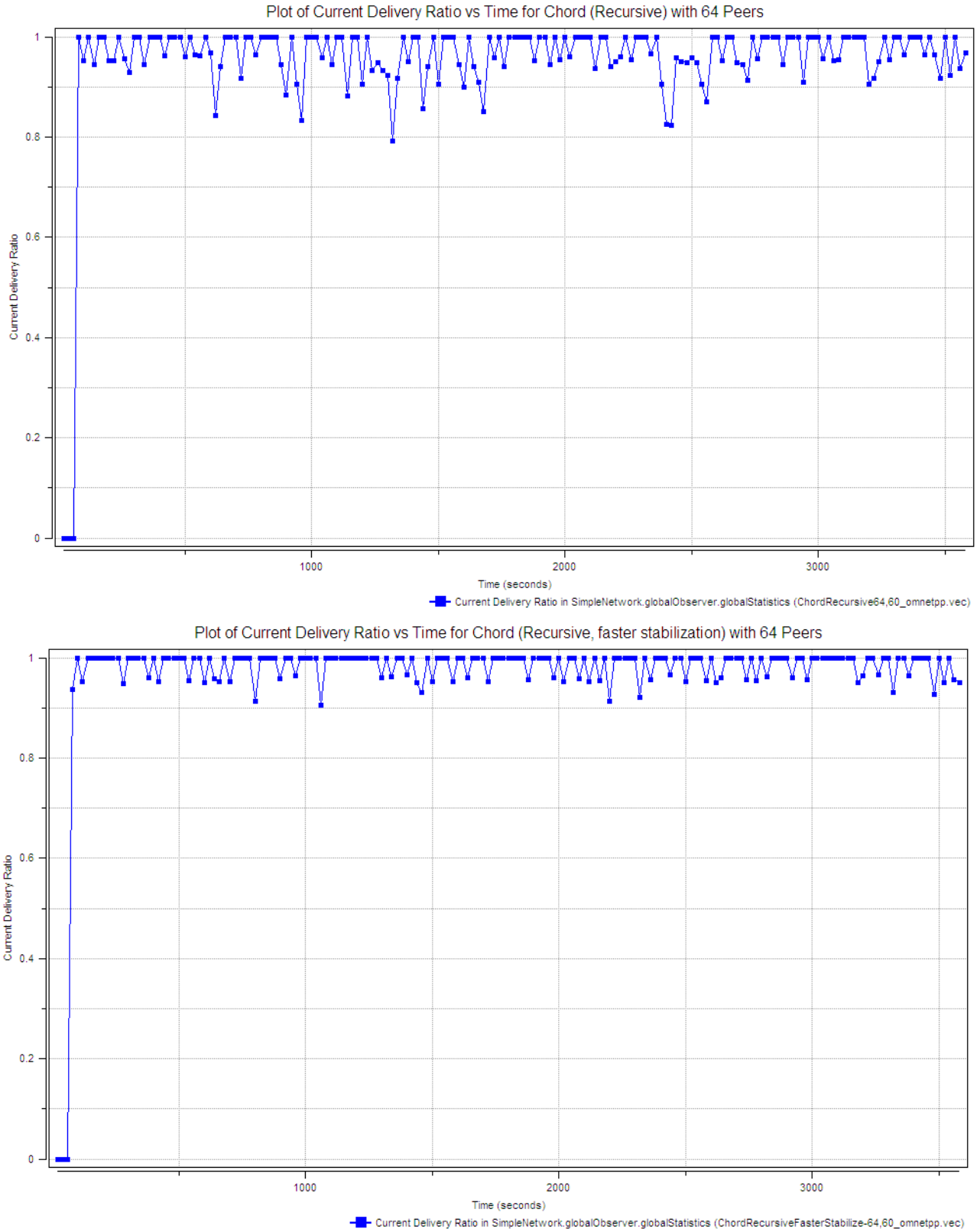


Figure 15(c) Current Delivery Ratio vs. Time for Group 1 (top) and 4 (bottom) with 64 peers

Observation: It is more evident at this stage of network growth that reduction in stabilization and finger table fixing times results in more stability; fluctuations are less frequent and of much smaller magnitude. Delivery Ratio for Group 4 remains on average well above 90% and longer periods at which it is 100%.

Trace Comparison of Group 1 vs. Group 4: Global Hop Count vs. Time

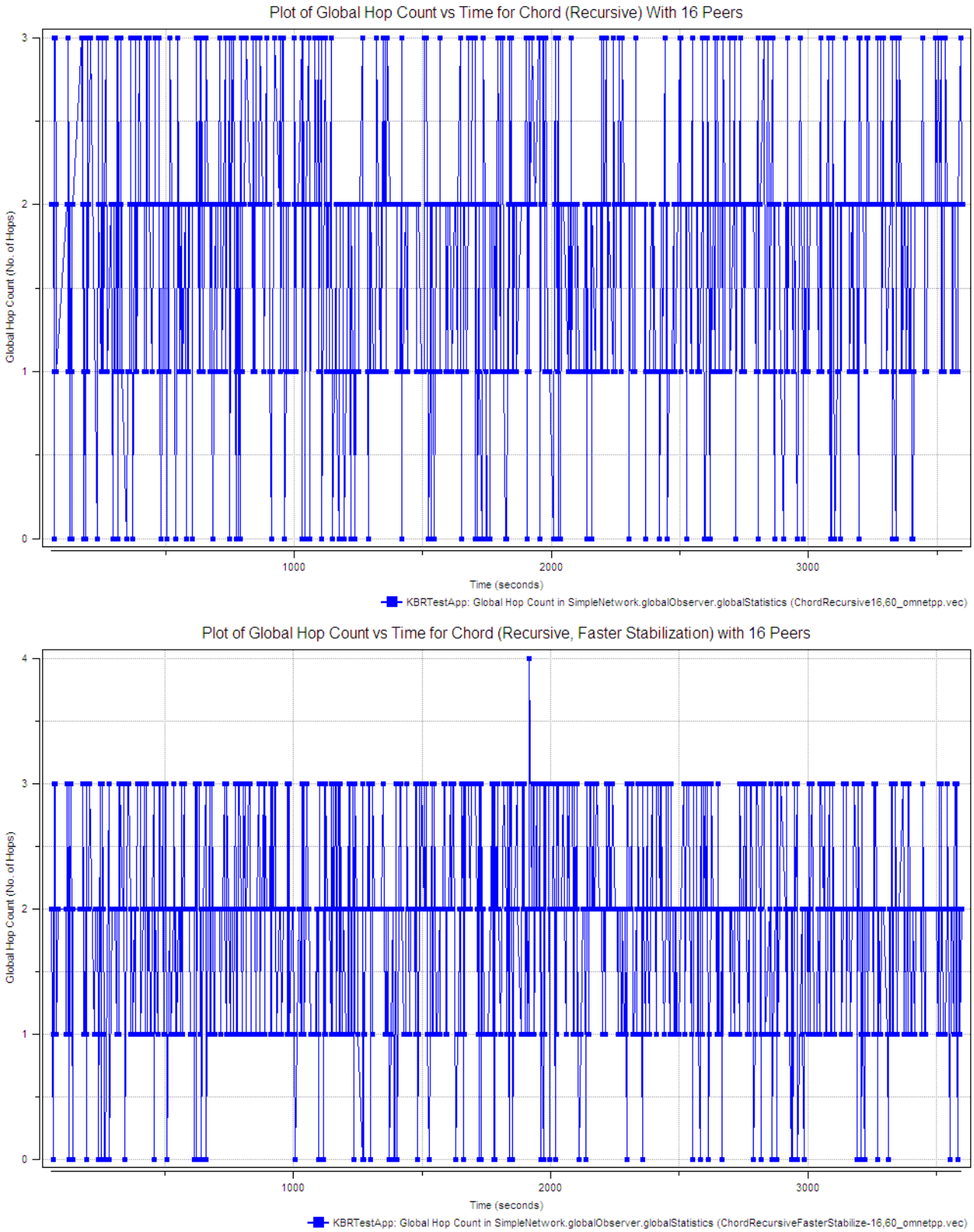


Figure 16(a) Current Delivery Ratio vs. Time for Group 1 (top) and 4 (bottom) with 16 peers

Observation: Doubling the frequency of inspection and updating of identifiers of predecessors and finger table entries results in more frequent fluctuations in the number of hops needed to reach a recipient in the presence of churn. The second network records a maximum of 4 hops at least once.

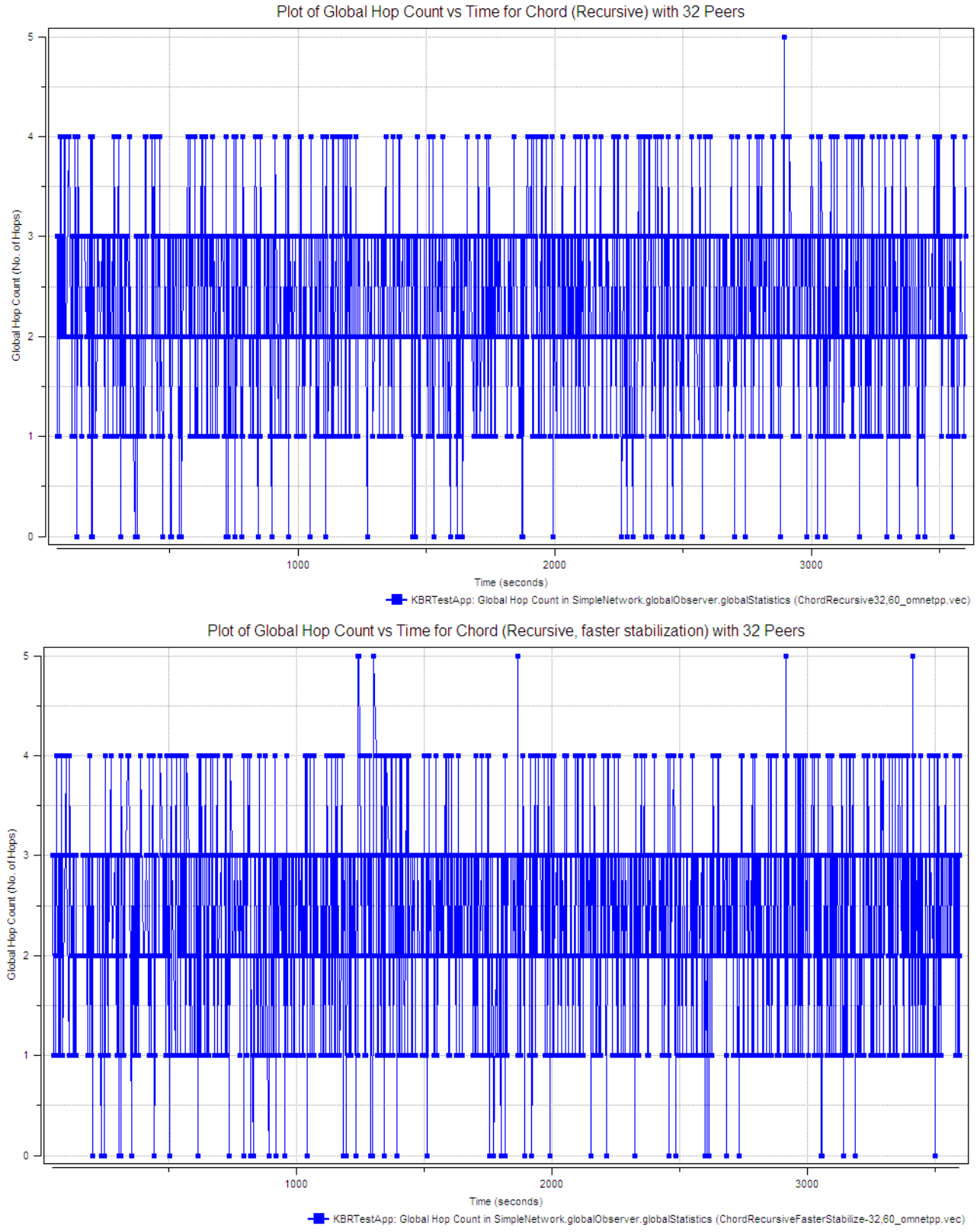
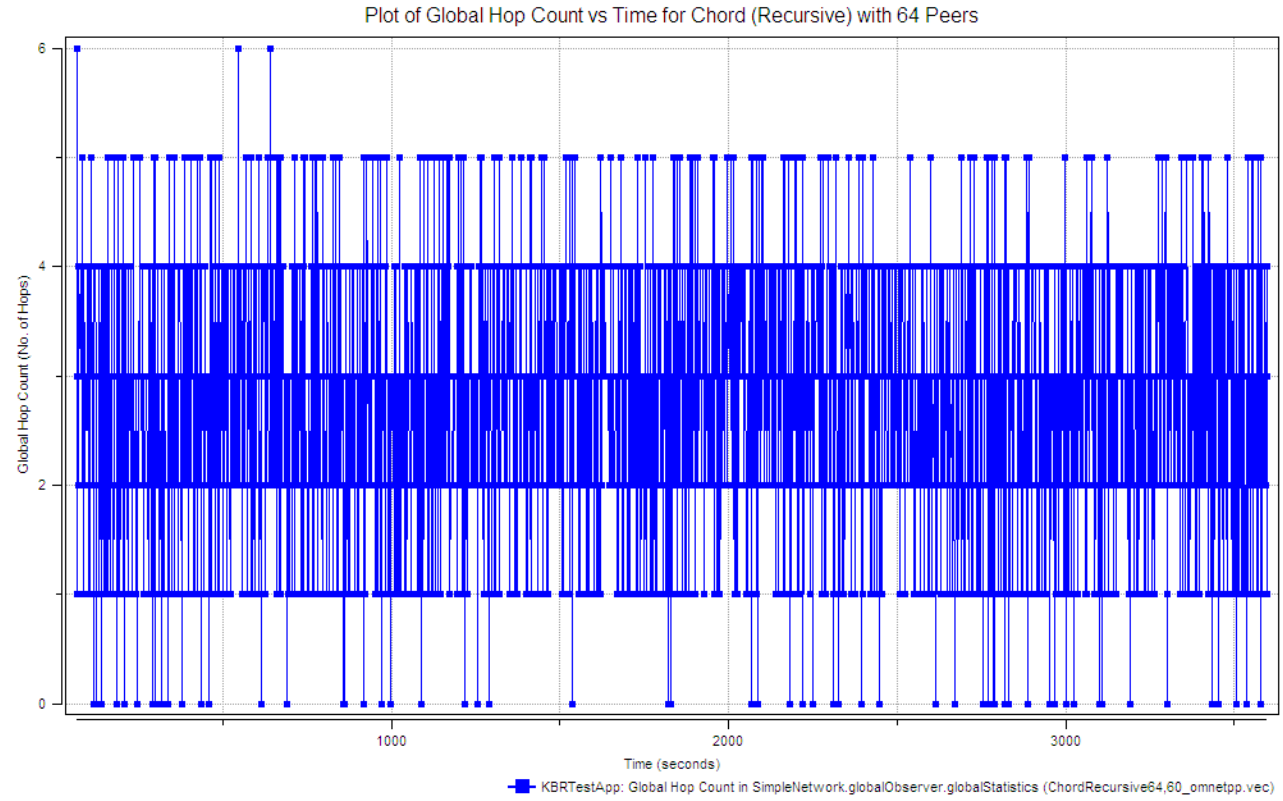


Figure 16(b) Current Delivery Ratio vs. Time for Group 1 (top) and 4 (bottom) with 32 peers

Observation: Doubling of network size results in the network of Group 4 registering a larger hop count more often, though exponential growth also brings the added bonus of more alternative routing paths, acting as an offset to the larger number of hops and resulting in almost similar behavior between the 1 and 3 hop range.



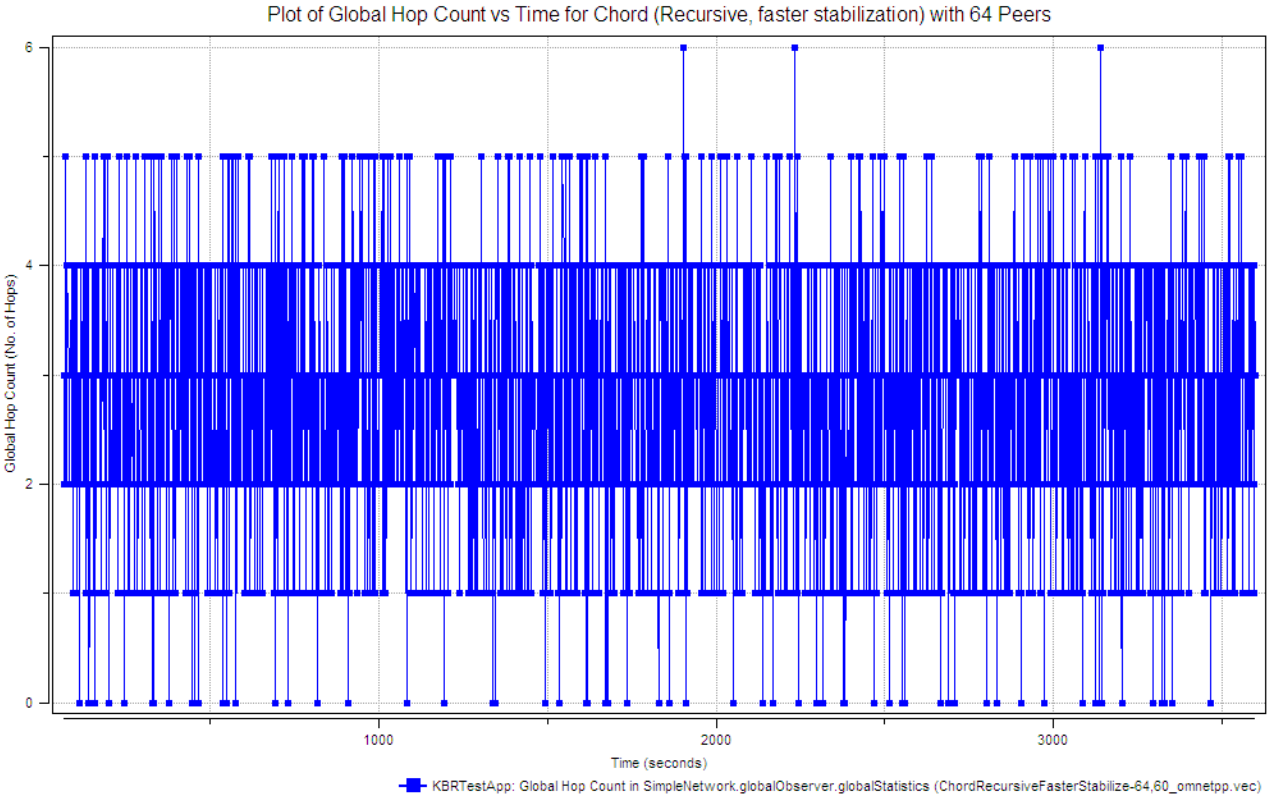


Figure 16(c) Current Delivery Ratio vs. Time for Group 1 (top) and 4 (bottom) with 64 peers

Observation: 64 peers in the network results in an expansion in the most heavily occupied range. The hop count now spans 1 to 4 hops most frequently, with an increased use of 5 hops and expansion beginning in the 6 hop range. However, it should be noted that Group 4 still uses the higher hop range more often.

Challenges and Lessons Learned

Two major challenges were experienced over the course of this project, which are listed and explained below:

- **Time lost with initial attempt using the OPNET simulation tool.**

  The initial plan for the project was to build the simulation model in the OPNET 11.0.A simulation tool environment. However, it was found that OPNET 11.0.A had no tools to simulate P2P networks. An attempt was made to develop a node model, but the learning curve for programming in OPNET was found to be extremely steep, due to the amount of documentation that had to be read in order to understand . The endeavor was abandoned after very little progress had been made by the end of a month, but by then a lot of time had been lost.

- **Shifting to a new simulation tool and an unfamiliar OS**

  After abandoning the work with OPNET as unfeasible within the given timeframe, a move was made to OMNeT++, INET and OverSim. However OverSim is only supported for the Linux operating system and supplied only in the form of uncompiled source code. There was an additional learning curve involved in compiling the INET Framework and OverSim, as well as learning how to use the OverSim and the Linux OS, having had no experience with either of them.

The lessons learned over the course of the project were as follows:

- Designing and implementing a P2P system that makes use of a DHT is an extremely challenging endeavor. There are many factors to be considered and balanced in the appropriate choice of a DHT implementation that produces the lowest overhead in terms of signaling, while at the same time efficiently routing messages to the intended peers and minimizing the effects of churn with nodes joining and leaving.

- The choice of simulation tool is extremely important, as an extremely steep learning curve results in lost time during a project with a short cycle. In addition, it should have good support

for P2P simulation models and an intuitive model interface that makes observation and analysis of network behavior as easy as possible.

- DHT routing parameters need to be chosen carefully, especially those related to dealing with changes in information about neighboring peers in the identifier space, in order to optimize the behavior of the chosen DHT for the specific network architecture and intended application.

Future Work

Given the simulation and analysis work covered in this project, the direction of future work would be to conduct similar analysis on other DHT algorithms, such as Pastry, CAN, Kademlia, Viceroy, Gia, Koorde, Broose and Bamboo. The aim of such work is to compare the performance of various DHT implementations in order to gain an understanding of the strengths and weaknesses of each. This would provide insight as to how certain features of these DHTs could be combined to build a DHT suitable for a certain application, such as P2P networks routing mostly: multimedia-oriented traffic; instant messaging, file sharing, control messages, etc. Another possible direction of inquiry would be an adaptable DHT protocol, that could perhaps shift from one type to another on the fly, so that a peers could decide amongs themselves which DHT protocol might be suitable for use given network situations, and use it accordingly.

Real-life Applications of DHTs

While the topic chosen and work done over the course of this project may seem somewhat abstract and theoretical, this is not so. DHTs are currently implemented and in operation in real-world applications. Some of those applications were encountered previously and are outlined in brief below.

1. **P2P File-Sharing Applications**

   The rise of peer-to-peer file sharing networks was what instigated interest in the peer-to-peer paradigm as a research area. From the initial days of networks using a central server to find other peers and then using peer-to-peer communication for file transfer (Napster), to the flooding network query used in the Gnutella network, P2P file sharing applications have become a major source of observed Internet traffic, accounting for up to 60% or more of traffic observed [1]. It has been observed that the most recent implementations of P2P file-sharing

applications (BitTorrent and Azureus, to name two) implement DHTs into their protocols, thus enabling faster searching and retrieval of files, as well as a greater likelihood of guaranteeing the file being searched for. The user interfaces of the BitTorrent applications is shown below.

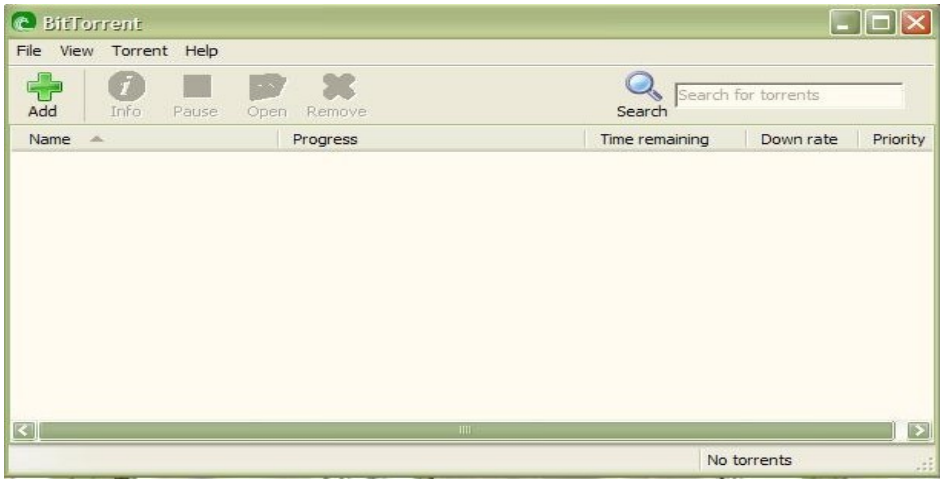


Figure 6. The BitTorrent file-sharing application interface

2. OEM Communications Software

With increasing interest in communication protocols for P2P networks, companies that develop Original Equipment Manufacturer (OEM) communications software for various applications are using DHTs in their software to achieve their product objectives of P2P communication. One such company is SIPeerior Technologies [14], located in Virginia, USA. SIPeerior's website states that its software "enables serverless implementations of real-time applications such as VoIP, IMS, IPTV, streaming media and gaming". SIPeerior uses DHT algorithms such as Chord in its communications software in order for devices using the software to form peer-to-peer networks where data can be distributed and retrieved efficiently. One such example is in the communications devices that might be used by first responders at the scene of an emergency. P2P communications software with DHT protocols integrated allows them to avoid the time spent in setting up a server and configuring their devices to communicate with each other. Instead, OEM P2P communications software such as that supplied by SIPeerior allows them to bring their devices to the scene and simply power them up. The devices will configure themselves, form a P2P network and route messages to each other using the software's built in DHT protocol. Thus, DHTs hold a lot of potential to change the way devices communicate with each other using self-organizing P2P networks.

References

[1] R. Steinmetz, K. Wehrle, “Peer-to-Peer Systems and Applications”, LNCS 3485, Springer-Verlag Berlin Heidelberg 2005.

[2] Hash function, http://en.wikipedia.org/wiki/Hash_Table – last modified April 9th, 2008 at 11:17.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications”, In *Proceedings of the 2001 ACM Sigcomm Conference*, ACM Press, 2001.

[4] RFC 3174 – US Secure Hash Algorithm 1 (SHA1) - http://www.faqs.org/rfcs/rfc3174.html - last accessed on April 18th, 2008.

[5] Manku, “Routing Networks for Distributed Hash Tables”, In *Proceedings of the twenty-second annual symposium on Principles of Distributed Computing (PODC)*, 2003.

[6] J. Li, J. Stribling, T. M. Gil, R. Morris, and F. Kaashoek, "Comparing the Performance of Distributed Hash Tables Under Churn", In *Proceedings of IPTPS*, 2004.

[7] J. Yang, “Measuring the Performance and Reliability of Routing in Peer to Peer Systems”, 2005.

[8] I. Baumgart, B. Heep, S. Krause, “OverSim: A Flexible Overlay Network Simulation Framework”, In *Proceedings of 10th IEEE Global Internet Symposium in conjunction with IEEE INFOCOM 2007*, Anchorage, AK, USA, 2007.

[9] OMNeT++ Community Site - http://www.omnetpp.org - last accessed on April 19th, 2008.

[10]INET Framework - http://www.omnetpp.org/staticpages/index.php?page=20041019113420757 - last accessed on April 19th, 2008.

[11] The OverSim P2P Simulator - http://www.oversim.org - last accessed on April 19th, 2008.

[12] The ScaleNet Project - http://www.scalenet.de/ - last accessed on April 19th, 2008.

[13] Ubuntu Linux Home Page - http://www.ubuntu.com/ - last accessed on April 19th, 2008.

[14] SIPeerior Technologies - http://www.sipeerior.com - last accessed on April 19th, 2008.