The Benefit of Stacking Multiple Peer-to-Peer Overlays

Tobias Amft Kalman Graffi

Heinrich Heine University, Düsseldorf, Germany
Computer Science Department
Technology of Social Networks Group

Technical Report TR-2017-002
Heinrich Heine University, Düsseldorf, Germany
Computer Science Department

September 2017

The Benefit of Stacking Multiple Peer-to-Peer Overlays

Tobias Amft and Kalman Graffi
Technology of Social Networks Group, Heinrich-Heine University Düsseldorf, Germany
Email: [amft, graffi]@cs.uni-duesseldorf.de

***Abstract*—In the past decades, various concepts of peer-to-peer (*P2P*) systems have found their way to applications in industry. Each peer-to-peer application defines a set of basic requirements: being reliable, robust, efficient, sometimes anonymous, sometimes location-dependent. Due to the large variety of overlay protocols and distributed services existing on the Internet, it is very likely that multiple overlays coexist on single nodes. In this paper, we investigate the synergy between multiple coexisting overlays and broaden the insights about stacked and parallel executed overlays. Specifically, we identify typical functionalities and requirements of existing overlay systems and derive a core set of modules necessary to create a complete overlay. Furthermore, we discuss how to avoid redundant functionalities in coexisting overlays and how to reduce operating costs. Finally, the benefits of overlay synergies are demonstrated based on simulations of Chord, Geodemlia and a Gnutella-like flooding overlay running together on one peer. It can be seen that overlay synergies between arbitrary types of overlays may result in increased failure tolerance and reduced communication costs.**

I. Introduction

Various peer-to-peer (P2P) overlays and applications have been proposed during the last two decades as they define a scaling alternative to regular client-server based applications. The main distinguishing characteristic of P2P systems (overlays + applications) is the construction of a distributed infrastructure which is utilized and provided by participating nodes (peers) at the same time, mostly without any single server organizing the intercommunication of peers. Prominent examples are file-sharing systems like Bittorrent[1] which disseminate data between peers without the help of a central server. In this way, systems in this category avoid server bottlenecks and failures, also their performance scales with the number of participating peers.

Reseach on peer-to-peer systems has lead to a huge diversity of different overlay solutions, each shipped with its own message forwarding strategy (routing algorithm) tailored for one specific purpose. Overlays in the group of unstructured overlays usually provide simple forwarding mechanisms in which data is transmitted to no specific peer, but rather to randomly selected peers. Example overlays in this category are Gnutella[2] or Bubblestorm[28]. In the group of structured overlays, peers are sorted according to an individual identifier which is considered in forwarding decisions. Chord[27], Pastry[25], and Kademlia[23] are three of the best studied examples of the so called distributed hash tables (*DHT*). Due to their system wide structure, structured overlays are able to identify and find (look up) other peers responsible for a certain part of the overlay, whereas objects in unstructured overlays have to be searched explicitly.

[1]http://www.bittorrent.com/
[2]http://rfc-gnutella.sourceforge.net/

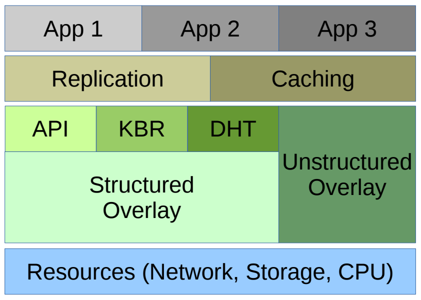


Fig. 1. Example of an overlay stack that consists of different modules which in combination form a full P2P system.

P2P applications dominated Internet traffic almost to decades and are now displaced by streaming services. the likely tendency is that the interest and the practical usage of P2P systems will probably hold on for the next few years. Since most overlays serve one specific purpose only, it is likely that multiple overlays might coexist on a single peer. The introduction of different overlay layers and modules which can be stacked together might lead to better performance than a parallel execution of all plain overlays. Using specialized modules only and combining them to a desired overlay avoids to implement Important functionalities multiple times and leads to reduced operating costs. Maniymaran et al. [22] discuss in their work a joint overlay consisting of an unstructured and a structured component. They show that only parts of the respective overlays have to be implemented to obtain full functionalities. Lin et al. [20] thereupon classify common synergies of gossip-based overlays and demonstrate how to benefit of them. We extend previous work with new insights on coexisting overlays in this paper.

A. Contribution

In this paper, we broaden the knowledge about coexisting overlays through the investigation of typical overlay characteristics. We introduce the idea of an overlay stack which is the combination of individual modules which all serve one specific purpose to reach a desired behavior. An example of how different overlay modules could be combined is roughly given in Figure 1. In Section III-B, we propose a simple way to avoid duplicate costs by using one DHT as basis for different P2P applications. We extend this approach in Section III-C with the idea of an overlay stack, which allows to add new overlay functionality without effort. Instead of a shared DHT, the overlay stack takes advantage of a common routing table, which is used and maintained by different overlay modules, like routing algorithms and update mechanisms. With knowledge about overlay stacks at hand, we aim to find basic functionalities and requirements of different overlays. Therefore, we summarize and characterize typical overlay parts, we identify interdependencies between them, and we propose possible optimizations that lead to an observable improvement of stacked overlays in comparison to single executed overlays. In specific, our contribution is to answer the following questions:

- *Which basic functions and core elements are required to form an overlay?* - In Section II, we identify a core set of modules which are required to form any desired overlay. We also identify which functionality of an overlay is application specific and which is overlay specific.
- *Do interdependencies between two overlay networks or modules exist?* - We identify redundant functionality in stacked overlays and show which parts of an overlay collaborate. Further, in Section III-D, we present the design of an overlay stack and discuss ways to remove redundant parts so that costs are reduced. In our evaluation in Section IV, we investigate in how far a coexistence of overlays or overlay modules affect their performance.
- *What is the quality, cost, and limitation of coexisting overlays executed in parallel?* - We identify and describe in Section IV the benefit of different overlay modules combined in an overlay stack in comparison to unmodified overlays running in parallel. We show that the operating costs of coexisting overlays can be lowered while their quality remains.
- Finally, we show that coexisting overlays can improve the stability of the whole system so that overall robustness against churn increases.

B. Outline

The remainder of the paper is structured as follows: Section II describes core modules which can be found in typical overlays. In Section III, we discuss possibilities of coexistence and identify core modules and patterns that are repeated in many overlays to provide specific functionalities. In Section III-C, we present the idea of an overlay stack and a common routing table for the optimization of coexisting overlays. We perform simulations to show benefits of coexisting overlays and present the results in Section IV. In Section V, we summarize existing work related to the field of stacked and coexisting overlays. We conclude our studies and our evaluation in Section VI.

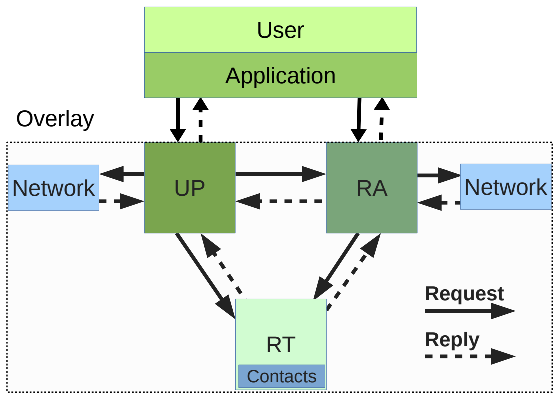


Fig. 2. Each overlay consists of mainly three modules: a routing table, a routing algorithm, and diverse update mechanisms.

II. Basic Overlay Modules

Overlays consist of different core modules which in combination result in a unique behavior of the P2P overlay. We describe a set of core modules for overlays and discuss in how far they can be optimized.

The **overlay identifier** describes the identity of a peer in an overlay. In some structured overlays like Chord [27], Pastry [25], or Kademlia [23], peers select a number out of a given identifier space, e.g. all numbers in the interval $[0, 2^{160})$ or $[0, 2^{128})$ as identifier. Data objects that are stored in a DHT are usually assigned to a number in the responding identifier space, so that always a responsible peer can be assigned to the data object. In this case, the overlay identifier is called overlay key.

As **overlay contacts** we denote the combination of an overlay identifier and one or more IP-port tuples. Overlay contacts are used as containers to store overlay identifiers and related IP-port tuples for any peer. Those containers can be send to other peers to announce a peer's identity in the network.

Peer-to-peer overlays operate above an existing **network** (e.g. the Internet) which provides end-to-end communication. The link between network and overlays should be implemented in a modular way so that it can be exchanged easily and all modules of an overlay stack have access to the same communication basis. In this way it is possible to optimize communication mechanisms separately from the logic of implemented overlays.

Routing table - Per definition, overlays are networks operating on existing networks and extend them with new routing structures. For this reason, every overlay maintains overlay contacts in a certain data structure, e.g. a list, a special table, or only a single link. The part of the overlay which is supposed to store overlay contacts is called the routing table (**RT**). The routing table is mostly a passive module which never requests data from any other overlay modules. On the contrary, other modules can request the routing table for overlay contacts or inform the routing table about new contacts. Most routing tables order overlay contacts according to a specific criterion, e.g. ordered by numbers, by distance, etc. Important is that overlay contacts can be reordered in any routing table at any time. The state of any routing table can be reconstructed at any moment, if all necessary overlay contacts exist. This leads to the following possible optimizations of routing table modules:

1) All overlay contacts can be put into one common routing table, which offers methods to add any overlay contact. Internally, the overlay contacts can be sorted according to their purpose. Whenever an overlay contact is queried, the routing table decides which contact fits best.
2) Some structured overlays like Chord [27] and Pastry [25] operate with similar overlay identifiers. A peer is thus able to re-use overlay identifiers across several distinct overlays. Unstructured overlays can then benefit from a common routing table in the way that all known contacts could possibly be used as next hops during forwarding.
3) As described in [22], only parts of different routing tables have to be implemented to obtain full functionality, also with a common routing table, overlay contacts can be re-used.

Routing algorithm - Each overlay defines at least one routing algorithm (**RA**) whose purpose it is to forward messages until a proper recipient is found. Each peer receiving a message decides locally, which known contact in the routing table would be best suited as next hop for a given message. The common goal of all peers is to forward and deliver messages as fast as possible to a certain target node. The routing algorithm needs access to the routing table to obtain knowledge about other contacts and possible next hops.

Overlays are typically characterized by the routing algorithms they use. Each routing algorithm requires special overlay identifiers and expects a certain order in known overlay contacts. We observe that routing algorithms can further divided into a core algorithm and an overlying operation. The core algorithm is responsible for forwarding messages according to predefined rules towards one or more target peers. It can be best described as implementation of the route(key $\rightarrow$ K, msg $\rightarrow$ M, nodehandle $\rightarrow$ hint) method described in the KBR API [9]. The overlying operation decides when an algorithm is successful or not, and it determines the purpose of an operation. Usual operations are for example Chord's *find_successor* operation, or *put* and *get* operations for DHTs.

The strict separation into pure algorithm and operation allows exchange between different implementation, so that for example Chord's *find_successor* operation could be based on an unstructured flooding algorithm. Since routing algorithms are essential parts of an overlay, it is hard to optimize them. One possible optimization could be to summarize multiple lookups or searches to one single lookup or search.

In general, routing algorithms can be categorized differently according to specific purposes. In the following we describe the three most prominent examples.

The first category of algorithms comprises *iterative routing algorithms* (Fig. 3). In this category, a node p usually asks well selected contacts from its routing table for better contacts

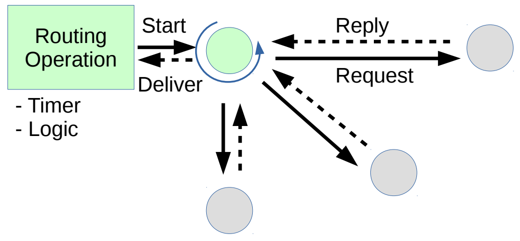


Fig. 3. Iterative routing algorithm.

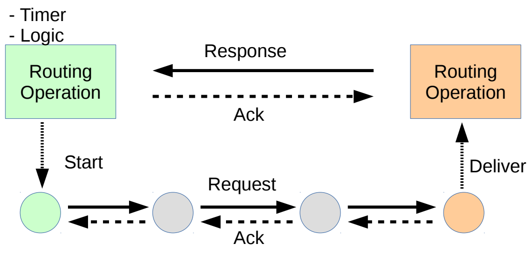


Fig. 4. Recursive routing algorithm.

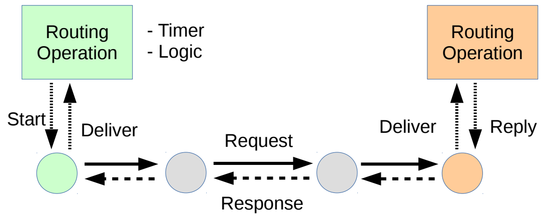


Fig. 5. Fully recursive algorithm.

(related to a given key) they know. Peers that are learned in this way are further asked for better contacts. In this way, peer p consecutively contacts selected peers until a peer responsible for a certain key is found. The control of the routing mechanism remains at the initiator of the lookup or search (peer p) at any time, which decides when to stop the forwarding mechanism.

Recursive routing algorithms make up the second category of routing algorithms (Fig. 4). In this class, the initiator p of a search of lookup selects one peer from its routing table as next hop and forwards a corresponding lookup/search message to it. Every peer receiving this message decides locally to whom the message should be forwarded, with the goal being to reach a responsible target node. The control about the lookup/search is given to another node with every forwarding decision. At some point, a node q decides to be responsible for an incoming message which contains the contact data of initiator p. In this way, q is able to contact initiator p directly and answer the request, thereupon peer p stops the routing operation successfully.

Fully recursive routing algorithms is the third category, as seen in Figure 5. They are similar to algorithms from category

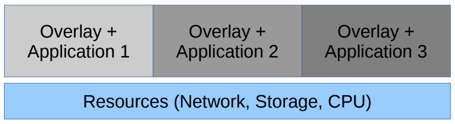


Fig. 6. Multiple overlays coexist without knowledge and further connection.

two with one exception, a responsible peer q does not respond directly to incoming requests. Instead the response is routed back the way it was forwarded. In this way, only the contacts of next and previous hops are exposed during communication, the sender and receiver of a message remain anonymous.

Update mechanisms - Overlays which are exposed to *churn*, which is a frequent join and leave activity of participants, need mechanism to stabilize the overlay structure regularly. Different types of update mechanism can be observed. On the one hand active update mechanism are frequently executed to learn about participating nodes in the network. Passive update mechanisms on the other hand typically just add peers that are discovered during lookups, requests, forwarding, or other processes. One special type of update mechanism is the join operation which introduces new participating nodes to the network. Its goal is mainly to find a certain set of nodes that are initially added to the routing table. Considering update mechanisms, they are a good point to optimize the overlay stack, since for every purpose only one stabilize-mechanism is needed. Redundant updates can be omitted. According to the components of the stacked system, it may happen that one update mechanism, e.g. for a structured overlay, automatically discovers contacts that can be reused for multiple routing tables.

III. Types of Overlay Synergies

In this section, we describe possible synergies of peer-to-peer overlays, we identify basic parts typical peer-to-peer overlays implement, and we derive a core set of functionalities each overlay provides. To explain the behavior of coexisting overlays, we compare different synergies and focus on the following points:

- Structure of overlays during coexistence.
- Requirements to the overlay implementation.
- Functionalities a synergy allows.
- Interdependencies between overlays and their parts.
- Costs, quality, and limitations of specific overlay synergy.

A. Coexisting Overlays Without Knowledge

In the first category of coexistence, overlays and related applications executed on one peer are not aware of each other. Each peer-to-peer system (overlay + application) serves one specific task, e.g. file-sharing, social interaction, or anonymous communication. Multiple overlays could coexist on one peer in this way, each occupying resources of the host system. Figure 6 shows an example in which multiple overlays run in parallel on one host, consuming its resources like storage, bandwidth, CPU, etc.

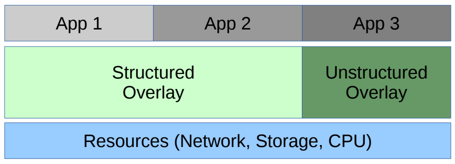


Fig. 7. A common DHT can be used as basic lookup service for multiple overlay applications.

The coexistence of multiple overlays in this class does not **require** changes in structure or implementation of the corresponding overlays. New overlays are implemented without knowledge about other solutions. **Functionality** is fully provided by overlays in this category. Since no structure or implementation needs to be changed, each overlay is able to serve the special purpose it is designed for. On the other hand, to add new functionality to the system, one either has to implement an additional overlay or has to integrate new features into an existing implementation. No **interdependencies** arise between the coexisting overlays, as each solution can be executed independently. The performance of overlays in this category is influenced either by their own characteristics or by the specific resources of the hosting peer. For example, small bandwidth on the hosting peer could restrict the function of multiple highly active overlays running in parallel.

In a performance-based comparison of this category, it is seen that **resource costs** (bandwidth consumption, storage consumption, CPU usage, etc.) increase linearly with the addition of new overlays to the hosting peer. The **implementation costs** are the sum of all implementation costs for the respective overlays. The benefit with this approach is that overlays remain their full functionality without general loss of **quality**. Further gain on the dynamism of the synergy system is that any overlay can be added or removed from the hosting peer without effect on the remaining overlays. Overlays in this synergy are not **limited** in their functionality, but the resources of the hosting peer influence the overlays directly.

Therefore it can be said that this class of synergies allow quick changes to the overall system, but summarizes in return all costs of participating overlays.

B. Common Overlay as Basis

In the second category of coexisting overlays, a reduction in the overhead of parallel executed overlays is made by the introduction of a common base for coexisting overlay applications. Assuming that many applications can be built on top of existing solutions for both structured and unstructured overlays, we shift the diversity of coexisting peer-to-peer systems from the routing layer to the application layer. One DHT is enough to provide a basic lookup service to identify responsible peers to a given key. As a complement, an unstructured solution then provides search functionality in finding objects in the network. The DHT provides store and retrieve functionalities according to the KBR interface [9] and enables, in most cases, message forwarding in logarithmic time. We have already shown in previous work [3], [17], [2], that multiple contrasting application types can be build on top of DHTs like Chord or Pastry. The applications benefit from the layered structure since both the DHT and the application can be optimized separately, new applications can be added to the system at any time. An example of coexisting applications atop a combination of one structured and one unstructured overlay is given in Fig. 7.

This category of synergy **requires** a strict separation of peer-to-peer systems into the two components of basic overlays and applications. The task of the basic overlay is to provide communication in form of lookup service or (file-) search. The applications on top implement further specialized functionalities. **Functionality** is moved to the component in which it is used. Redundant functionalities, such as similar routing mechanisms and maintenance strategies, are avoided through the use of a joint component for a specific purpose. Additional functions can be added as new application on top of the existing base overlay. Although different applications are not **dependent** on each other, they are coupled to a common structured or unstructured overlay. Unlike for the case discussed in Section III-A where multiple overlays consume the resources of the hosting peer, here the **interdependencies** between overlay parts are shifted to another layer. Applications are build atop an underlaying base overlay which provides communication. The quality of this overlay influences the applications on top.

Performance-based comparisons show that **resource costs** are partially shared in this kind of synergy, and of note is that the costs produced by periodically executed stabilization algorithms are avoided. Instead of running one DHT for each application, the basic costs can be reduced to one DHT that serves every application. Also unstructured overlays benefit from the synergy, because most routing algorithm in this category do not require any order on neighboring peers. However, the traffic costs for lookups and search operations initiated by the specific applications can not be avoided. **Implementation costs** are reduced with this approach since the overlay synergy can be extended with new applications at any time. The **quality** of the system in terms of successful lookups and search operations is preserved in this class of coexistence, as the basic overlay part is completely implemented on the hosting peer. Depending on the base, the quality can be better in comparison to the first category. Overlays like Pastry or Geodemlia which update their structure on lookup activity will benefit from multiple, sometimes simultaneously started lookups. The **limiting** factor in this category are the basic overlays which provide communication. All applications have to fit the utilized core overlays. Failures in the basic overlay will affect the applications located atop. Arsham Sabbaghi Asl showed in his master's thesis [4], that coexisting applications that are coupled through a common DHT can influence each other massively. Depending on the applications stacked upon the DHT, and depending on the DHT itself, bad behavior affects coexisting applications as well as good behavior does.

C. *Specialized Overlay Modules*

In this third category of coexistence, we divide overlays into smaller parts (*modules*) which in combination result in a full peer-to-peer overlay (Fig. 2). The problem of category two, in which one or two basic overlays provide core communication functionalities to applications located on top, is that the basic overlays limit the functionalities of the whole system. Each implemented application relies on the underlaying lookup service and performs as good as this core, and some overlay structures are too specialized to be covered by a usual DHT like Chord [27] or Pastry [25]. If the synergy had to be extended with further overlay specific functionalities, such as a location-aware search or a special anonymization service, one fundamental overlay would not be enough to provide the necessary basics for communication. In other words, having one structured and one unstructured overlay alone as basis will never provide a complete core set of functionalities.

To overcome this limitation, we divide overlays in small modules, each fulfilling one special task. The stacking of different modules leads to a certain behavior and thus to a certain kind of overlay. In this section, we identify and describe typical overlay modules and their characteristics. With this knowledge at hand we are able to identify redundant functionality and suggest optimizations to the modules. In this way, functionality is shifted to smaller modules which can be stacked together to form a certain protocol or service which offers same functionalities as one common overlay (Section III-B). Moreover, the basic overlay, structured or unstructured, can be formed through the stacking of different modules to provide its service. If needed, additional overlay functionality can be added without (re-)implementation of a full overlay. In most cases, the addition of a few modules is sufficient.

The division of overlays and applications in small functional parts **requires** a lot of discipline during implementation. Good coding principles like modular programming foster the effect of modular overlay implementations. Clear interfaces and strict separation into compact modules allows to exchange overlay modules with more optimized solutions. **Functionality** is clearly separated and implemented in specific modules. To obtain a desired behavior of a peer-to-peer system, different modules have to be combined. Therefore, a redundant functionality can be avoided as every required function is implemented only once in the overlay stack. New **interdependencies** then arise in the modular implementation synergy, as applications are no longer dependent on one basic communication overlay. Moreover, modules are dependent on the functionalities of other specialized modules. Consequently, dependencies between whole overlays are outsourced to smaller specific modules.

A performance-based comparison of this class shows that **resource costs** produced by redundant stabilization operations are reduced in this category of synergy. The redundant costs are avoided as the necessary overlay modules are initiated once per peer. **Implementations costs** appear higher at first sight in comparison to coexisting overlays in category one,

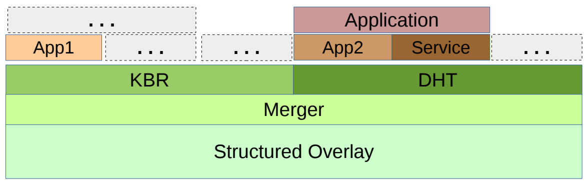


Fig. 8. The overlay stack can be extended vertically, by adding new applications, services, or other modules on top of existing modules. Grey boxes with three dots indicate possible spaces for new modules.

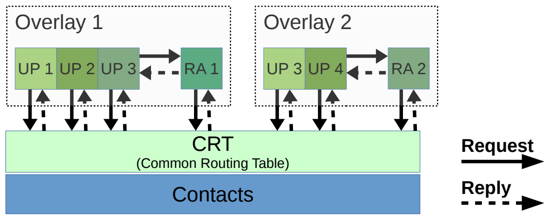


Fig. 10. A common routing table can serve multiple overlays at once.

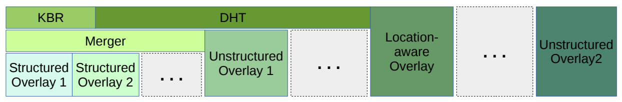


Fig. 9. The overlay stack can be enhanced horizontally, by adding new overlays, or other modules into existing layers of the stack. Grey boxes with three dots indicate possible spaces for new modules.

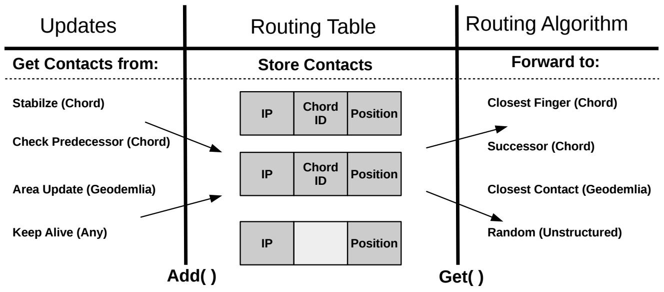


Fig. 11. A common routing table gives access to all overlay contacts stored on a peer.

which are not aware of each other. This is because implementations of coexisting overlays have to follow similar design principles and possess clear interfaces so that a combination of implementation parts is possible. In the end, this makes the addition of new functionalities an easy task. The **quality** of stacked overlays is similar to existing overlay implementations, since modules can be combined anytime to form any specific overlay. Depending on the implemented modules, the overall quality of the system can benefit from the interaction of modules. Overlays that check neighboring peers upon request will benefit from the coexistence of other overlay modules which start communication processes frequently. The range of functionality is not longer **limited** in this category, due to the ease of adding functions to the overlay stack. Similar concepts and modules can be reused and optimized separately.

D. The Overlay Stack

In this section, we describe the idea of an overlay stack, which is to combine different modules to obtain one or multiple overlays or overlay applications. The previous sections summarize three possible synergies of overlay systems running on one peer, the overlay stack can be, but has not to be, a combination of all three synergies. Moreover, the overlay stack is the idea to combine different overlay parts in a way that a certain behavior is obtained and second, that the implementation of duplicate modules and functionalities is avoided.

In Section III-B we show that multiple applications can be built on top of a shared DHT which provides basic lookup functionality. Figure 8 gives an overview how the overlay stack can be extended vertically by adding new applications on top of a common DHT or other existing applications. Nevertheless, using a common DHT as underlying lookup service might not be suited as basis for all possible applications, for example in cases in which more specialized routing mechanisms are desired. Overlays for anonymous communication for example are difficult to be built on top of a normal DHT, as we elaborate in our paper [3].

To enable a horizontal growth of the overlay stack (Fig. 9), which is the addition of new routing protocols as basis for applications on top, we introduce a *common routing table*. Similar to a common DHT which offers basic routing functionality to applications, a common routing table offers a shared storage for overlay contacts and can be accessed by other overlay modules in the overlay stack, like those explained in Section II. Figure 10 shows an example in which different routing algorithms and update mechanisms access one routing table.

Examining the overlay modules described in Section II, we notice that routing table modules are the only modules which are acting passively in an overlay. Overlay update mechanisms and routing algorithms define the behavior of an overlay and serve one specific purpose, for example to update successor pointers in Chord. However, overlay contacts as described in Section II can be used by multiple routing algorithms in most cases. Flooding-based overlays for example do not expect an overlay contact to have a certain identifier. Only IP-address of a peer and port of the respective application are needed to forward messages so that is does not matter which type of overlay contact is used, be it a Chord contact or a Geodemlia contact.

As to be seen in Figure 11, overlays running on the same peer can take advantage of a shared routing table which is used by each overlay running on the peer. The common routing table is filled like standard routing tables: whenever a suitable contact is identified by an overlay mechanism, it is added to the routing table. Each overlay module, be it a routing algorithm or a stabilization method, no matter to which overlay it belongs, has access to all contacts in the routing table. The common routing table may either pre-sort existing contacts according to the different overlays specified, or contacts can be sorted according to a specific purpose upon request from other overlay modules. In addition to the common routing table, our overlay stack implements a common module for network functionality which offers basic communication functions to all other overlay modules. Every time an overlay module of peer A sends a message to another peer B, the common network module adds all overlay contacts used by peer A to the message. In this way, peers learn about all identities of their surrounding neighbor contacts. The base communication layer is designed not to manipulate overlay messages directly, because the overlays can encrypt their messages.

In Section IV, we compare the efficiency of a modular overlay stack with common routing table and common network functionality to original overlay implementations running in parallel.

IV. Evaluation

For our simulations we used the event-driven simulator PeerfactSim.KOM [12] [11] [14]. The number of simulated nodes is set to 1000, which is not very high for standard peer-to-peer simulations, but allows us to identify and study possible benefits of coexisting overlays. Churn is activated throughout all simulation.

The goal is to show that costs of coexisting overlays in terms of traffic can be lowered through the reduction of stabilize-mechanisms. At the same time the quality of the system can be maintained through different synergies of overlay modules. Moreover, we show that the stability of two coexisting overlays can be increased through the use of a common routing table (*CRT*).

A. Simulation Setup

In our simulations we study the coexistence of two structured overlays, namely Chord [27] and Geodemlia [16] and one unstructured, Gnutella-like overlay which uses flooding to disseminate data. We choose these three overlays as they are very contrasting in their functionalities and requirements. We show with this setup, that even a synergy between overlays with different types of identifier spaces results in benefits.

Peers in Chord have a one-dimensional identifier and are connected to peers that have the next higher identifier and to peers that gave next lower identifier. In this way, Chord is able to assign a responsible peer to any given identifier. Geodemlia on the other hand, is a location-aware overlay which maps data and peers to physical locations and allows a user to search for location-based data. To do so, nodes in Geodemlia have a two-dimensional identifier which represents the coordinates they are located. Peers in Geodemlia additionally store contact data to near and distant nodes so that messages can be routed towards a desired location. The identifiers both overlays use are contrasting in their structures. Peers in Chord are identified by an integer number out of the interval $[0, 2^{160})$, whereas nodes in Geodemlia are identified by their physical location.

General Settings	
Simulator details	PeerfactSim.KOM [12] [11] [14], simple network module, no packet loss, exponential churn
Standard Settings in Scenarios	
Churn start	Minute 30
α	The interval in which lookups, area searches, and flooding attempts are started is $\alpha = 120s$ (approximately 10 percent of peers start lookups in each overlay)
Common routing table (*CRT*) size	$\log_2(N) * (f_c + f_g + f_f)$ with N being the size of the network, here $N = 1000$, f_i being a factor for Chord (f_c), Geodemlia (f_g), or Flooding (f_f)
Simulation time	60 minutes
A) Normal Overlay Behavior	
Scenario	Join phase until Minute 20, lookups and searches from Minute 20 on. α = 120s, 240s, 480s
Overlay Settings	Chord (unmodified), Geodemlia (unmodified), Flooding (unmodified), overlay stack running Chord, Geodemlia, Flooding simultaneously with CRT
B) Different Update Times	
Scenario	Join phase until Minute 20, lookups and searches from Minute 20 on. Chord update intervals = 30s, 60s, 120s, 240s, 300m
Overlay Settings	Chord (unmodified), overlay stack running Chord, Geodemlia, Flooding simultaneously with CRT
C) Sudden Leave Events	
Scenario	Join phase until Minute 20, lookups and searches from Minute 20 on. Sudden leave events affecting 20%, 40%, 60%, 80% of all nodes
Overlay Settings	Chord (unmodified), Geodemlia (unmodified), Flooding (unmodified), overlay stack running Chord, Geodemlia, Flooding simultaneously with CRT
D) Isolation Events	
Scenario	Join phase until Minute 20, lookups and searches from Minute 20 on. Sudden isolation event affecting 20%, 40%, 60%, 80% of all nodes
Overlay Settings	Chord (unmodified) (with and without Ring Reunion merger), Geodemlia (unmodified), Flooding (unmodified), overlay stack running Chord, Geodemlia, Flooding simultaneously with CRT (with and without Ring Reunion merger)
E) Sudden Join Events	
Scenario	Join phase until Minute 20, lookups and searches from Minute 20 on. Sudden bootstrapping (no full join) affecting 20%, 40%, 60%, 80% of all nodes
Overlay Settings	Chord (unmodified), Geodemlia (unmodified), Flooding (unmodified), overlay stack running Chord, Geodemlia, Flooding simultaneously with CRT

TABLE I
Simulator Setup and Different Scenario Setups.

In Chord, peers periodically update contact information to other peers to keep the network in a stable state, whereas in Geodemlia, contact information are checked for validity if they are used. New contacts are actively looked up in Chord and contacts in Geodemlia are learned from bypassing messages. The flooding-based overlay has no specific requirements on its routing table or on the overlay contacts it uses. Upon receiving a message, this approach selects a subset out of all known contacts and forwards the message to all nodes in this subset.

In our simulations, we compare the unmodified, original protocol versions of Chord, Geodemlia, and the Gnutella-like flooding overlay, each running individually, to the overlay stack approach in which all overlays are running in parallel and a common routing table exists to maintain overlay contacts. We simulate churn according to an exponential, in which, most peers leave the network after short online times and few nodes stay online for longer times before leaving the network. The offline time of peers on the other hand is much shorter so that peers join the network again, shortly after they have left. Churn begins at Minute 30 in all our simulations (if not declared differently), the mean session length is set to approximately 30 minutes. Each simulation stops after 60 minutes.

To be able to rate the success of the individual overlay protocols, we analyze if lookups in Chord, area searches in Geodemlia, or flooding attempts are successful. For this reason, random lookups in Chord, area searches in Geodemlia, or flooding attempts are started in our simulations, so that 10 percent of all peers executing Chord start lookups every α seconds, 10 percent of all peers executing Geodemlia start area searches every α seconds, and 10 percent of all peers executing Gnutella start to flood the network every α seconds.

The common routing table in our overlay stack is limited in its capacity to store overlay contacts, in order to save storage. The size of the routing table is calculated the following: $\log_2(N) * (f_c + f_g + f_f)$ with N being the size of the network, here $N = 1000$, f_i being a factor for the weight of the respective overlay. As Chord needs a successor list of size $\log_2(N)$, successor and predecessor pointers, as well as approximately $\log_2(N)$ contacts in its finger table, we set $f_c = 3$. We set $f_g = 12$ because Geodemlia in our evaluation stores contacts according to 4 directions in 12 different buckets, each having 3 slots. As the flooding-based overlay reuses Chord and Geodemlia contacts, we set $f_f = 0$.

Next, we describe the scenarios in detail, an overview about the simulated scenarios is also given in Table I.

Scenario A) Normal Overlay Behavior: In this scenario, we test the basic behavior of our overlay stack and show that its overlays perform at least as good as the original overlays. To do so, we compare the unmodified versions of Chord, Geodemlia and Gnutella individually to an overlay stack in which Chord, Geodemlia, and Gnutella share and use a common routing table. Additionally, we compare the unmodified overlays to respective versions, in which the original routing table is exchanged with a common routing table, but no other overlays are accessing this table. Our intention here is to show that the common routing table performs as good as a normal routing table and we want to get a first insight into the performance of our overlay stack. The setup is the following, from Minute 0 to 20, peers join the network. Thereafter, peers execute lookups, area searches, or search operations according to the respective overlay. In this scenario, the periodic lookup interval α is set to 120, 240, and 480 respectively.

Scenario B) Different Update Times: In this scenario, we investigate the impact of Chord's update mechanisms on the Chord protocol. We compare the unmodified Chord protocol to the overlay stack in which Chord, Geodemlia, and Gnutella share common routing table. With this setup, we want to find out, if Chord inside the overlay stack benefits from other overlays in the overlay stack in terms of robustness and self-stabilization. In this setup, peers join the network from Minute 0 to 20 and start lookups (or area searches, or flooding attempts) thereafter. In this scenario, the periodic lookup interval α is set to 120 seconds, so that each peer in an overlay starts a lookup, area search, or flooding attempt every 2 minutes with a probability of 10 percent. The interval for Chord's update mechanisms is set to 30s, 60s, 120s, 240s, and 300m.

Scenario C) Sudden Leave Events: In this scenario, we compare the unmodified protocols to the overlay stack in the case of sudden leave events of a big part of the overlay. Again, peers join the network in the first 20 minutes of the simulations and start lookups, searches, or flooding-attempts afterwards. At Minute 30, X percent of the peers of a certain overlay leave the network suddenly. We set X to 20, 40, 60, 80 percent of the network. For the overlay stack, we simulate separately that peers leave Chord, Geodemlia, or Gnutella. With this scenario, we show the robustness and self-stabilization process of the overlay stack.

Scenario D) Isolation Events: We compare the unmodified version of Chord to the overlay stack during isolation events in this scenario. Peers join the network in the first 20 minutes of this scenario and start lookups, searches, or flooding-attempts afterwards. 20, 40, 60, or 80 percent of the network fall victim to a sudden isolation event at Minute 30. Peers inside the isolated network are able to communicate with each other. Communication to nodes outside the isolated region is not possible. Furthermore, we compare the behavior of the protocols with and without the Ring Reunion merging algorithm for Chord-like overlays which we presented in [1]. We show with this scenario, that overlays coupled through a common routing table influence each other so that modules benefit from update and maintenance algorithms belonging to other modules.

Scenario E) Sudden Join Events: In this scenario, we compare the unmodified overlays to the overlay stack in the case of sudden join events. We show with this scenario, that overlays in the overlay stack benefit from the common routing table during the join phase. In specific, we show that bootstrapping is possible without explicitly joining other peers in the network. In this setup, peers join the network in the first 20 minutes and start lookups, searches, or flooding-attempts thereafter. Then at Minute 30, X percent of the peers suddenly start Chord, Geodemlia, or Gnutella, without operating a full join process which is to contact other peers initially to fill the routing table of the respective overlay. We set X to 20, 40, 60, 80, 100 percent of the network.

B. Metrics

To analyze the overlay stack with its common routing table and to compare it with the original overlay protocols, we need some metrics describing the success and the costs of the simulated protocols. In the following, we describe the most Important metrics used in our simulations.

- **Success** and failure of lookup or search operations can be measured depending on the three overlays. To measure success of a lookup in Chord, we check if the successor found by Chord's *find_successor* operation is correct. A successor of an identifier is considered to be correct, if no other node is online which would be a better successor of the identifier. To measure success of an area search in Geodemlia, we check if the *find_nearest_nodes* operation finds all nearest nodes according to a specified location. The quality of flooding-based overlays can be measured by introducing the *half-life* metric which denotes how long it takes to flood half of the network and thereby reach half of the nodes with a query message.

- **Hops and delivery time** are important metrics to rate the efficiency of a lookup, area search or flooding attempt. The number of hops or amount of visited nodes tells us how many nodes are affected by a lookup or search, the delivery time states, how fast the related operation performs.
- **Traffic costs** in terms of sent and received messages, as well as total bytes sent and received per time and per peer are needed to value the costs produced by an overlay solution.

C. Simulation Results

Simulation Results	
Metrics	Per Minute: Successful / Initiated lookups, Delivery Time, Visited Nodes, Total Bytes Sent Per Peer / Message: Delivery Time, Visited Nodes, Total Bytes Sent
Parameters	Scenario A) I = Periodic Lookup Interval(120s, 240s, 480s) Scenario B) I = Chord Update Interval (30s, 60s, 120s, 240s, 300m) Scenarios C and D) B = Size of Group B (80%, 60%, 40%, 20%) Scenarios E) B = Size of Group B (100%, 80%, 60%, 40%, 20%)
A) Normal Overlay Behavior	
A.1	Unmodified Chord vs. Chord in Overlay Stack
A.2	Unmodified Flooding vs. Flooding in Overlay Stack
A.3	Unmodified Geodemlia vs. Geodemlia in Overlay Stack
B) Different Update Times	
B.1	Unmodified Chord vs. Chord in Overlay Stack
C) Sudden Leave Events	
C.1	Unmodified Chord vs. Chord in Overlay Stack
C.2	Unmodified Flooding vs. Flooding in Overlay Stack
C.3	Unmodified Geodemlia vs. Geodemlia in Overlay Stack
D) Isolation Events	
D.1	Unmodified Chord without Ring Reunion Algorithm vs. Chord in Overlay Stack without Ring Reunion Algorithm
D.2	Unmodified Chord with Ring Reunion Algorithm vs. Chord in Overlay Stack with Ring Reunion Algorithm
D.3	Unmodified Flooding without Ring Reunion Algorithm vs. Flooding in Overlay Stack with Ring Reunion Algorithm
D.4	Unmodified Geodemlia without Ring Reunion Algorithm vs. Geodemlia in Overlay Stack with Ring Reunion Algorithm
E) Sudden Join Events	
E.1	Unmodified Chord vs. Chord in Overlay Stack
E.2	Unmodified Flooding vs. Flooding in Overlay Stack
E.3	Unmodified Geodemlia vs. Geodemlia in Overlay Stack

TABLE II
OVERVIEW: SIMULATION RESULTS.

In the following, we evaluate the results of our simulations according to the scenarios presented in Section IV-A. Table II gives an overview on how our simulation results are numbered and organized. We did not print error bars in our figures, because the results of our simulations are very close, and they are not needed to roughly describe the behavior of our overlay stack.

For each scenario described in Section IV-A and Table I, we first focus on the behavior of the respective overlay during the simulation time. Therefore, we present for each scenario the success rate of lookups or area searches or flooding attempts, the average delivery time of messages, the average number of visited nodes per lookup or area search or flooding attempt, and the total amount of bytes sent in the network which is approximately the number of received bytes in the network. To rate the quality of a flooding attempt, we observe the time needed, to flood half the network, which is denoted as the half-life period. To show the load per peer and per message, we present secondly the delivery time per message, the number of visited nodes per message and the bytes sent per peer as histograms. In order to be able to compare overlays in the overlay stack directly to the unmodified version, we place the results for an unmodified version of an overlay directly alongside the results of respective overlay the overlay stack. For example, in each Scenario X.1, where X is A,B,C,D,or E, the results for unmodified Chord are directly placed to the results of Chord in the overlay stack. Depending on the different scenarios, we vary the interval in which lookups, searches, and flooding attempts are started (Parameter I in Scenario A), we vary the interval of update messages in Chord (Parameter I in Scenario B), and we alter the number of nodes leaving, joining, or getting isolated (Parameter B in Scenarios C,D,E). The results for each scenario are further described in the following.

1) Scenario A.1 Chord: In Figures 12 and 13, we compare the unmodified version of Chord to its opposite in the overlay stack for different lookup intervals. Focusing on the ratio of successful lookups in both variants, it can be seen that Chord inside the overlay stack, which is uses a common routing table, solves as many queries correctly as the original Chord variant. Considering the average delivery time, it can be seen that the overlay stack version of Chord delivers messages in approximately half the time compared to the unmodified version. Reason for this observation is that the common routing table in the overlay stack offers more and better fingers than in the original protocol. As a result, approximately half the number of nodes have to be visited during a lookup and therefore less messages and less traffic has to be consumed in the overlay stack. Comparing Figures 12(d) and 13(d), it can be seen, that Chord consumes approximately the same traffic regardless how often lookups are started, reason is, that most lookups and thus most traffic is produced by Chord's update mechanisms.

2) Scenario A.2 Flooding: Considering Figures 14 and 15, we observe that the half-life period of flooding attempts in the overlay stack, which is to reach half the nodes in a network through a flooding attempt, is approximately half the size compared to the unmodified flooding overlay. In both versions, we limit the number of forwarded copies of a message to $\log_2(N)$ per node only, with N being the size of the network (here: 1000 nodes). The benefit of the overlay stack compared to the unmodified single overlay version is, that the common routing table offers a great variety of possible contacts to the routing algorithm instead of limiting the set of contact nodes to $\log_2(N)$ nodes only. As a result, less routing loops occur and more nodes can be visited during a flooding attempt, as to be seen in Figures 15(c) and 14(c). Considering the total number of bytes sent, it can be seen that the overlay stack version consumes more traffic than the unmodified flooding version.

3) Scenario A.3 Geodemlia: The comparison between the unmodified Geodemlia version in Figure 16 and the Figure 17 shows no special behavior of the overlay stack version or the original version of Geodemlia. Nevertheless, it can be seen, that both approaches behave similarly and the overlay stack version of Geodemlia performs at least as efficient as the unmodified version. Investigating the average number of visited nodes per message per time, it can be seen that the Geodemlia algorithm visits almost all nodes in the network to find the nearest nodes of an area.

4) Summary Scenario A: In Scenario A we have seen, that overlays in the overlay stack behave similar to the unmodified version of the respective overlay. Depending on the overlay, Scenario A shows that under normal conditions operation costs can be reduced with the common routing table. Considering Chord for example proves, that costs for lookups can be halved, comparing the flooding approaches, it can be seen, that routing quality can be increased without cost growth.

5) Scenario B.1 Chord: We compare the unmodified version of Chord to the overlay stack version of Chord in Figures 18 and 19. Comparing the ratios of successful lookups in both approaches, it can be seen that lookups are more successful the more updates are executed in Chord. Focusing the unmodified version of Chord in Figure 18, we observe that lookups are not successful if no updates are made (I=300m). Chord in the overlay stack on the opposite is able to perform successful lookups even if no updates are made (I=300m). In general it can be seen that the overlay stack version of Chord performs lookups more successful than the original protocol. The number of total bytes sent in Figure 19(d) supports our assumption made in Section IV-C1 that the traffic consumption in Chord is mainly induced by its update mechanisms. It follows theoretically, that using the overlay stack with common routing table, less updates are needed to perform as good as the unmodified Chord variant with more updates. As a result, the overlay stack can help to reduce traffic overhead while Chord's lookup performance is increased.

6) Scenario C.1 Chord: In Scenario C, we investigate the impact of sudden leave events on the original overlay protocols and on our overlay stack. It can be seen in Figure 20 that Chord is able to remain stable if approximately 20 percent of all peers leave the network suddenly at Minute 30. If more peers leave the network, lookups in Chord are not successful anymore. Considering Chord in the overlay stack on the other hand (Figure 21, we observe lookups to be successful again after a small time, even if 80 percent of the network suddenly turns offline. Observing the traffic overhead, it can be seen that during Minute 30 when most peers leave the network, messages need up to 10 times longer to receive a target node compared to the typical behavior.

7) Scenario C.2 Flooding: Investigating the flooding-based overlay in Scenario C, we observe another benefit of the overlay stack. It can be seen in Figure 22(c), that in the unmodified version of the flooding overlay approximately the same number of nodes are visited per message for different sizes of parameter B. Reason is, that the original routing table does not learn about failing nodes. In Figure 23(c) instead, it can be seen that the number of visited nodes is similar to the number of nodes in the network. This behavior occurs because other overlays in the overlay stack notice contacts to be failed and remove the dead contacts from the routing table. The number of visited nodes increases slowly from Minutes 30 to 60, because some offline peers join the overlay again due to churn.

8) Scenario C.3 Geodemlia: Figures 24 and 25 compare Geodemlia in its unmodified version and Geodemlia as module in the overlay stack. In this Scenario, the unmodified version of Geodemlia is able to perform more successful lookups than the overlay stack version during network failures. This behavior might be a result from restructuring processes in the underlying common routing table. Similar to the flooding overlay in this Scenario, the unmodified version of Geodemlia visits dead nodes during an area search whereas Geodemlia in the overlay stack avoids to contact offline peers. Therefore the number of visited nodes per message does not decrease significantly during the sudden leave event.

9) Summary Scenario C: Scenario C shows that our overlay stack has different impacts on its overlays. On the one hand, Chord and the flooding-based overlay benefit from the overlay stack. On the other hand, Geodemlia performs worse in the overlay stack if big parts of the overlay turn offline suddenly. Nevertheless, in this Scenario, the overlay stack benefits from its modular structure. Individual parts of the stack which are not performing as desired can be exchanged or optimized separately. In this way, the performance of the whole overlay stack can be improved through the individual optimization of each module inside.

10) Scenario D.1 Chord without Merging Algorithm: In Figures 26 and 29 we compare Chord in its original version to Chord in the overlay stack in the presence of isolation events. In Scenario D.1 no additional merging algorithm is enabled which could unify the disrupted overlay parts again. Similar to the results in Section IV-C1, Chord in the overlay stack consumes less resources than the unmodified protocol, as can be seen in Figures 29(b), 29(c), and 29(d). Figures 26(a) and 27(a) show the ratio of successful lookup per initiated lookup. It can be seen, that without merging algorithm, no approach is able to reach 100 percent correct lookups. As we described earlier in our paper [1], during isolation events, multiple rings are formed in Chord. It might be possible that lookups are successful in the separated rings, but our metric requires lookups to be correct globally. As described in Section IV-B, we mark lookups as successful if no better successor for the related identifier can be found. However, in this scenario it is likely that a better successor can be found in the other isolated ring, so that the ratio of successful lookups is equal to the maximum ratio of nodes in a group.

11) Scenario D.2 Chord with Merging Algorithm: Comparing Figure 28 to Figure 29, it can be seen that original Chord and Chord in the overlay stack profit from a merging algorithm like our Ring Reunion Algorithm proposed in [1]. In both variants, Chord is split into multiple rings at Minute 30. The Ring Reunion Algorithm manages in both cases to merge communication islands in the isolate region and around Minute 40 when isolation is over, the separated networks are merged to one ring again. Again it can be seen that the average delivery time of messages and the average of visited nodes per message in our overlay stack have approximately half the value of the original version.

12) Scenario D.3 Flooding: In Figures 30 and 31 we compare the flooding-based overlay without merging algorithm to its opposite in the overlay stack which is affected by the Chord merger. In specific, Figures 30(c) and 31(c) show that the flooding-based overlay in the overlay stack benefits from the Ring Reunion Algorithm which merges the Chord overlay in the overlay stack. We observe in Figure 31(c), that the number of visited nodes per message is reduced from 100 percent to approximately 60 or 80 percent at Minute 30, depending on the size of the biggest group of nodes during isolation. From Minute 40 on, 100 percent of all nodes in the network can be reached again, since the common routing table provides access to peers which have been unreachable during isolation. Without merging algorithm, the number of nodes reached by the flooding message does not increase after the isolation event, as can be seen in Figure 30(c).

13) Scenario D.4 Geodemlia: In Scenario D.4, we compare the unmodified version of Geodemlia to Geodemlia in our overlay stack. We observe that both version do not differ roughly. Althoug the original version of Geodemlia, as can be seen in Figure 32, does not operate a merging algorithm, area searches in the original Geodemlia version are successful after the isolation event. The reason why area searches in Geodemlia are possible after an isolation event is that peers in Geodemlia use a big routing table which is not updated actively. It happens therefore, that previously known contacts remain in the routing table if no better contacts are met. Those contacts which have been known before the isolation event survive the isolation event and can be used again thereafter.

14) Summary Scenario D: We have seen in Scenario D, that overlays in the overlay stack benefit from update mechanisms executed in other overlays. Especially overlay merging algorithms like our Ring Reunion Algorithm [1] can affect all overlays in the overlay stack positively. It can be seen that overlays with small routing table like the flooding-based overlay evaluated in Section IV-C12 benefit the most from commonly used routing tables. We also learn from Scenario D.4 in Section IV-C13, that some overlays have a robust routing table per se. Those overlays do not benefit from the common routing table significantly, but their are not affected negatively either.

15) Scenario E.1 Chord: In Scenario E.1, a group of nodes start the Chord overlay without fully joining the network. We want to find out in this scenario if new functionalities can be bootstrapped in the overlay stack. Comparing the ratio of successful lookups in the original Chord variant (Figure 34) and the overlay stack version (Figure 35), we observe that our overlay stack supports newly activated overlay modules through the common routing table. Figure 35(a) shows, that more than 40 percent of Chord lookups in the overlay stack are successful shortly after up to 100 percent of all nodes start the Chord protocol. In comparison, in the unmodified Chord variant represented in Figure 34(a), more than 40 percent successful lookups are reached if only up to 40 percent of all node start the Chord protocol.

16) Scenario E.2 Flooding: Comparing both flooding-based overlays in Scenario E.2, we observe again, that our overlay stack is able to bootstrap new overlay functionality. As can be seen in Figure 37(c), flooding in our overlay stack reaches more nodes compared the unmodified flooding approach, which can be seen in Figure 36(c). Like in previous scenarios, the half-life period during flooding attempts is also lowered through our overlay stack in this scenario.

17) Scenario E.3 Geodemlia: Also Geodemlia inside the overlay stack benefits from a common routing table in the way that more area searches are successful in the overlay stack version than in the unmodified version. Figures 38 and 39 show that area searches are slightly more successful in the overlay stack than in the single Geodemlia version. It can also be seen, that the overlay stack consumes resources almost equally to the original Geodemlia version.

18) Summary Scenario E: In Scenario E we have shown that our overlay stack supports newly activated overlay modules during the bootstrapping phase. Although the success rate of lookups, area searches, and flooding attempts of the three overlays could not be increased extremely, it can be seen that the overlay stack improves the quality of coexisting overlays during joining phases slightly. It needs further investigation, to find out how new functionalities can be fully started at runtime without loss of quality. Again, the modular structure of our overlay stack fosters the efficient optimization of overlay modules so that the improvement of bootstrapping functionality can be done in each overlay individually.

V. Related Work

The key based routing (KBR) API [9] is an early description of fundamental functionalities for structured (and partially unstructured) overlays. Overlays implementing the KBR interface can be extended with replication mechanisms like PAST [10], publish / subscribe solutions like SCRIBE [26], or further applications. Maniymaran et al. [22] create a joint overlay which comprises a structured and an unstructured overlay. In specific, they identify contacts in the routing table which can be reused in other overlays. Lin et al. [20] classify synergies of gossip-based overlays and demonstrate their benefits within GossipKit [13], a framework for composing overlays. Another overlay describing framework and language has been introduced by Behnel et al. in [6] and [5]. Macedon [24], later Mace [19], is an high-level language for the description and automatic creation of overlays. A finite state machine is used to realize overlays, the re-use of existing components or the coexistence of multiple overlays is not considered. Other solutions, such as OCALA [18] and Oasis [21], introduce new layers in the network stack which enable to switch between different overlays and applications. A convergence layer is used in these works to multiplex and demultiplex traffic which is sent between different overlays. This allows to used overlay-based packet delivery besides regular IP-based routing.

Various existing works on peer-to-peer overlays already follow the principles of overlay and application separation. A case study on how to build layered DHT applications is presented in [7]. A multi-layer frameworks for social networks which places a distributed data plane on top of Pastry [25] is described in [15]. Some works like [8] combine different applications into one overlay.

VI. Conclusion

In this paper we studied the coexistence of different physical overlays on a single peer. We identified a core set of requirements and functionalities overlays typically consist of and derived different types of overlay synergies. For each classification of synergy we discussed possibilities to reduce implementation and execution costs. We introduced the concept of a common routing table which gives coexisting overlays access to all known contacts. In our simulations we investigated the effect of different overlay synergies on the performance and stability of the participating overlays. It can be observed that a shared routing table is able to increase the robustness of the system which in turn reduces traffic costs since less stabilization operations are needed. We further show that our overlay stack allows to add new overlay functionality on top of the common routing table and allows to switch on and off new routing functionalities during runtime. Thus, the bootstrapping of new overlay functionality is possible without costly join procedures.

References

[1] T. Amft and K. Graffi, "A Tale of Many Networks: Splitting and Merging of Chord-like Overlays in Partitioned Networks," Technology of Social Networks Group, Heinrich Heine University, Düsseldorf, Germany, Tech. Rep. TR-2017-001, 2017.

[2] ——, "Moving Peers in Distributed, Location-based Peer-to-Peer Overlays," in *Proceedings of the International Conference on Computing, Networking and Communications (ICNC)*, 2017.

[3] T. Amft, B. Guidi, K. Graffi, and L. Ricci, "FRoDO: Friendly Routing over Dunbar-based Overlays," in *Proc. of Int. Conf. on Local Computer Networks (LCN)*, 2015.

[4] A. S. Asl, "Auswertung von parallel betriebenen, DHT-basierten Peer-to-Peer Overlays," Master's thesis, Department of Computer Science, Heinrich Heine University Düsseldorf, June 2017.

[5] S. Behnel, "Slosl–a modelling language for topologies and routing in overlay networks," in *Int. Conf. on Parallel, Distributed and Network-Based Processing (PDP)*. IEEE, 2007, pp. 498–508.

[6] S. Behnel and A. Buchmann, "Models and languages for overlay networks," in *Databases, Information Systems, and Peer-to-Peer Computing*. Springer, 2007, pp. 211–218.

[7] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. M. Hellerstein, "A Case Study in Building Layered DHT Applications," in *Proc. of ACM Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005.

[8] C. Chow, M. F. Mokbel, and X. Liu, "Spatial Cloaking for Anonymous Location-based Services in Mobile Peer-to-Peer Environments," *GeoInformatica*, vol. 15, no. 2, pp. 351–380, 2011.

[9] F. Dabek, B. Y. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays," in *Proc. of Int. Workshop on Peer-to-Peer Systems (IPTPS)*, ser. LNCS, vol. 2735. Springer, 2003.

[10] P. Druschel and A. I. T. Rowstron, "PAST: A Large-scale, Persistent Peer-to-Peer Storage Utility," in *Proc. of IEEE Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.

[11] M. Feldotto and K. Graffi, "Comparative evaluation of peer-to-peer systems using peerfactsim. kom," in *Proc. of Int. Conf. on High Performance Computing and Simulation (HPCS)*, 2013.

[12] ——, "Systematic evaluation of peer-to-peer systems using peerfactsim. kom," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 5, pp. 1655–1677, 2015.

[13] P. Grace, G. Coulson, G. Blair, L. Mathy, W. Yeung, W. Cai, D. Duce, and C. Cooper, "Gridkit: Pluggable overlay networks for grid computing," *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, pp. 1463–1481, 2004.

[14] K. Graffi, "PeerfactSim.KOM: A P2P System Simulator – Experiences and Lessons Learned," in *Proc. of Int. Conf. on Peer-to-Peer Computing (P2P)*. IEEE, 2011.

[15] K. Graffi, C. Groß, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz, "LifeSocial.KOM: A Secure and P2P-based Solution for Online Social Networks," in *Proc. of IEEE Consumer Communications and Networking Conf. (CCNC)*, 2011.

[16] C. Gross, D. Stingl, B. Richerzhagen, A. Hemel, R. Steinmetz, and D. Hausheer, "Geodemlia: A robust peer-to-peer overlay supporting location-based search," in *Peer-to-Peer Computing (P2P), 2012 IEEE 12th International Conference on*. IEEE, 2012, pp. 25–36.

[17] B. Guidi, T. Amft, A. De Salve, K. Graffi, and L. Ricci, "DiDuSoNet: A P2P Architecture for Distributed Dunbar-based Social Networks," *Peer-to-Peer Networking and Applications*, pp. 1–18, 2015.

[18] D. A. Joseph, J. Kannan, A. Kubota, K. Lakshminarayanan, I. Stoica, and K. Wehrle, "OCALA: An Architecture for Supporting Legacy Applications over Overlays," in *Proc. of Int. Conf. on Networked Systems Design and Implementation (NSDI)*, 2006.

[19] C. E. Killian, J. W. Anderson, R. Braud, R. Jhala, and A. M. Vahdat, "Mace: language support for building distributed systems," in *ACM SIGPLAN Notices*, vol. 42, no. 6. ACM, 2007, pp. 179–188.

[20] S. Lin, F. Taïani, and G. Blair, "Exploiting synergies between coexisting overlays," in *Proc. of Int. Conf. on Distributed Applications and Interoperable Systems (IFIP)*. Springer, 2009, pp. 1–15.

[21] H. V. Madhyastha, A. Venkataramani, A. Krishnamurthy, and T. E. Anderson, "Oasis: an Overlay-aware Network Stack," *Operating Systems Review*, vol. 40, no. 1, pp. 41–48, 2006.

[22] B. Maniymaran, M. Bertier, and A.-M. Kermarrec, "Build one, get one free: Leveraging the coexistence of multiple p2p overlay networks," in *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*. IEEE, 2007, pp. 33–33.

[23] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Proc. of Int. Workshop on Peer-to-Peer System (IPTPS)*, ser. LNCS, vol. 2429. Springer, 2002.

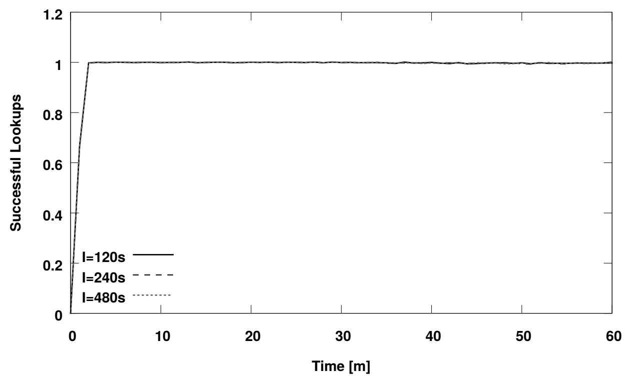
[24] A. Rodriguez, C. E. Killian, S. Bhat, D. Kostic, and A. Vahdat, "Macedon: Methodology for automatically creating, evaluating, and designing overlay networks." in *NSDI*, vol. 4, 2004, pp. 20–20.

[25] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *Proc. of IFIP/ACM Int. Conf. on Distributed Systems Platforms (Middleware)*, 2001.

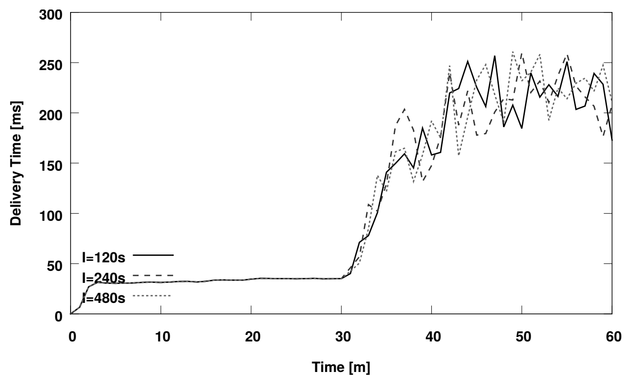
[26] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel, "SCRIBE: The Design of a Large-Scale Event Notification Infrastructure," in *Proc. of Networked Group Communication, Int. COST264 Workshop*, ser. LNCS, vol. 2233. Springer, 2001.

[27] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *Proc. of Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2001.

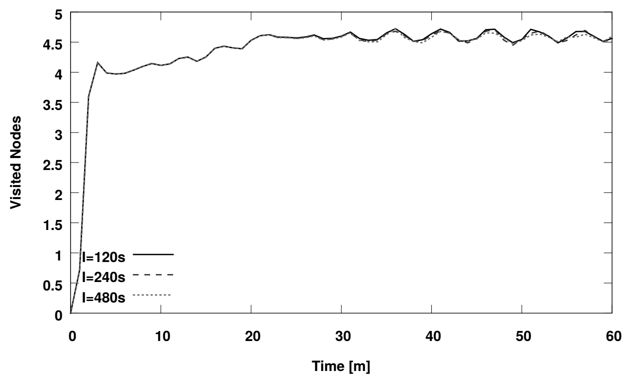
[28] W. W. Terpstra, J. Kangasharju, C. Leng, and A. P. Buchmann, "Bubblestorm: Resilient, Probabilistic, and Exhaustive Peer-to-Peer Search," in *Proc. of ACM Int. Conf. on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2007.



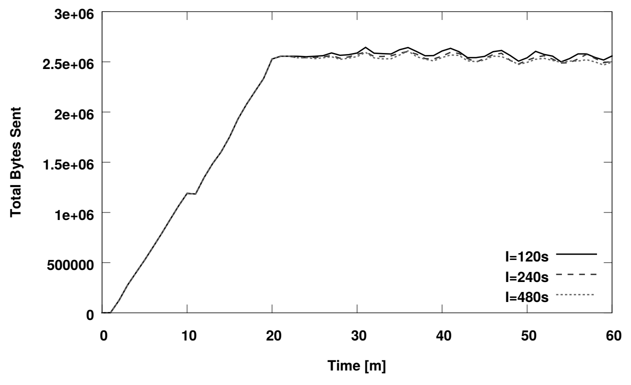
(a) A.1: Successful / Initiated Lookups (Ratio).



(b) A.1: Average Delivery Time.

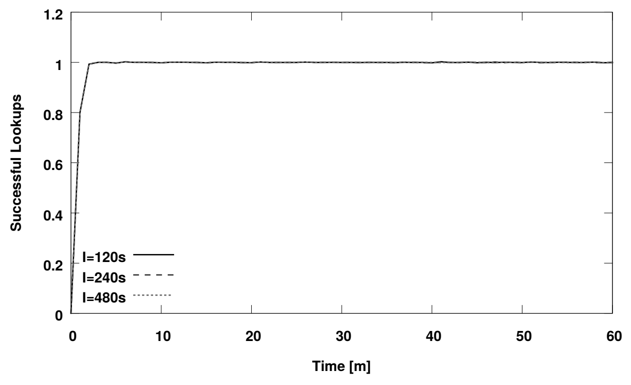


(c) A.1: Average Visited Nodes.

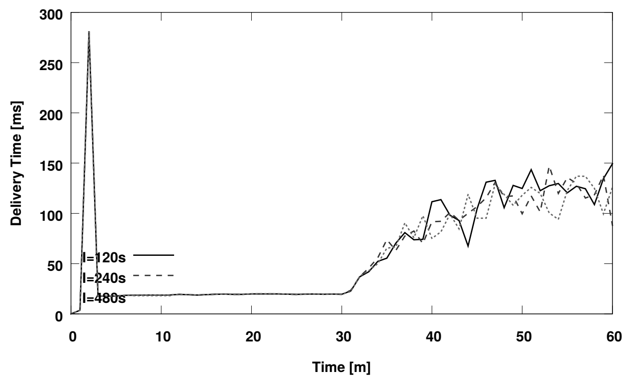


(d) A.1: Total Bytes Sent.

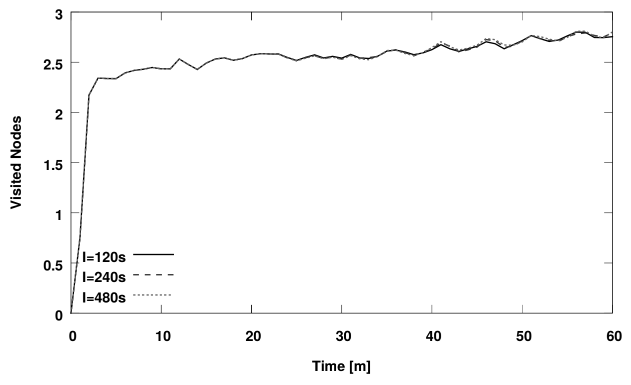
Fig. 12. A.1: Unmodified Chord.



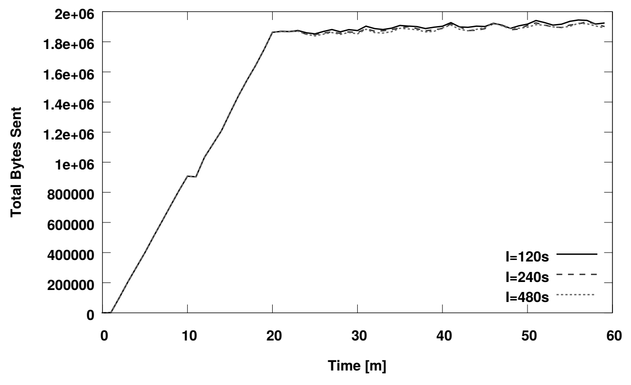
(a) A.1: Successful / Initiated Lookups (Ratio).



(b) A.1: Average Delivery Time.

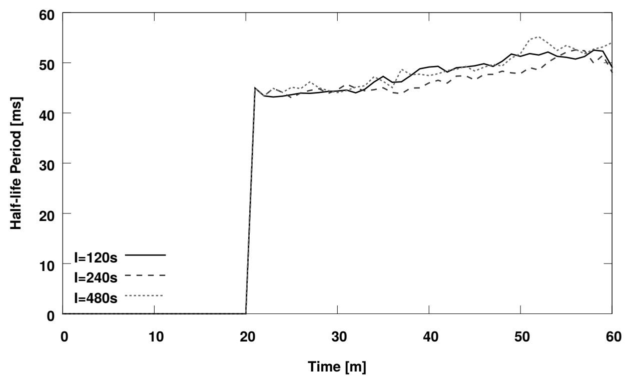


(c) A.1: Average Visited Nodes.

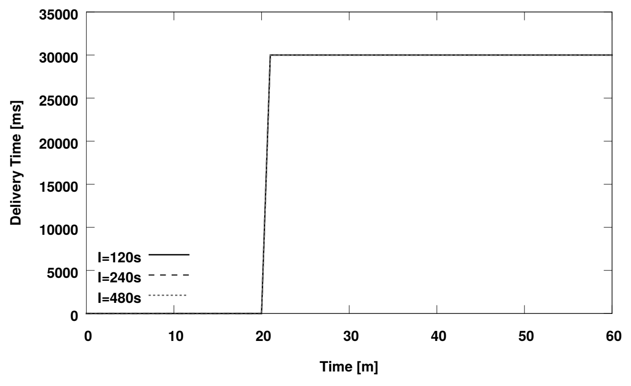


(d) A.1: Total Bytes Sent.

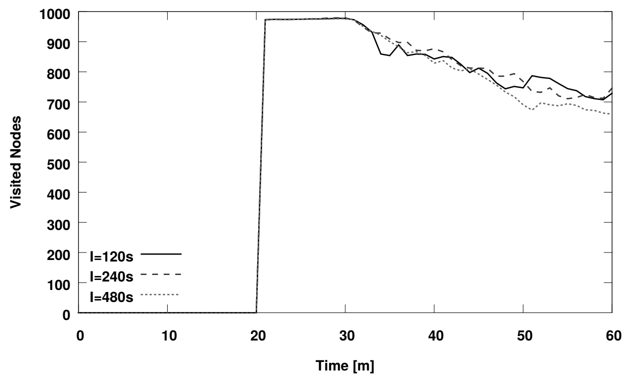
Fig. 13. A.1: Chord in Overlay Stack.



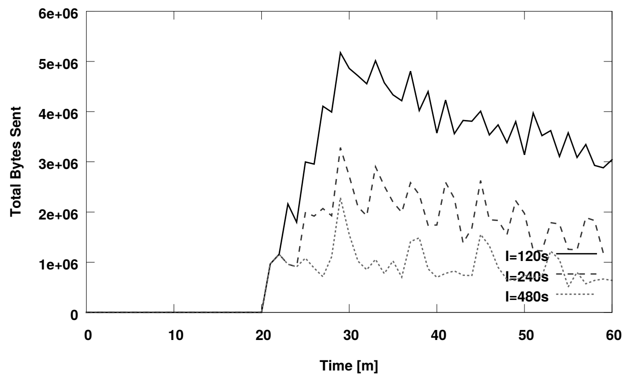
(a) A.2: Half-life Period.



(b) A.2: Average Delivery Time.

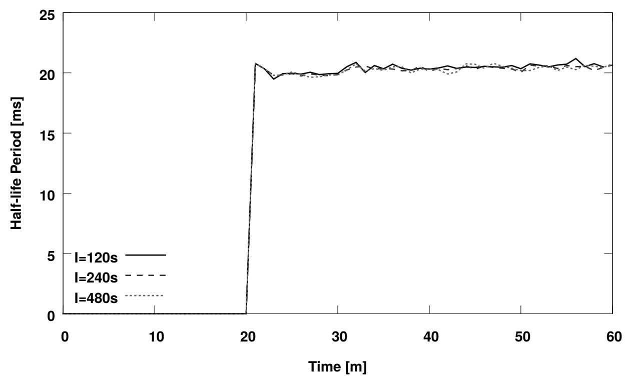


(c) A.2: Average Visited Nodes.

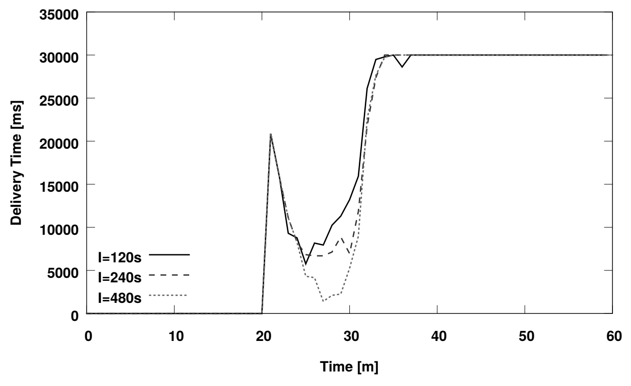


(d) A.2: Total Bytes Sent.

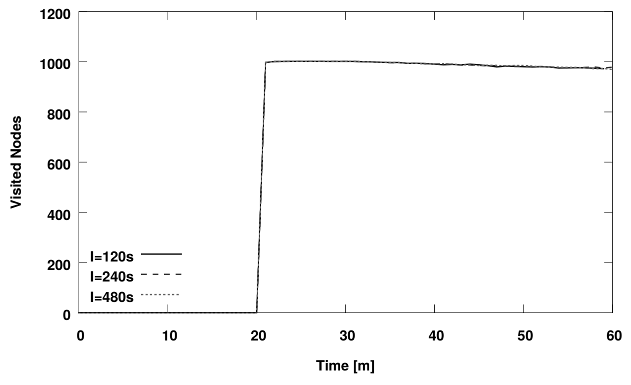
Fig. 14. A.2: Unmodified Flooding Overlay.



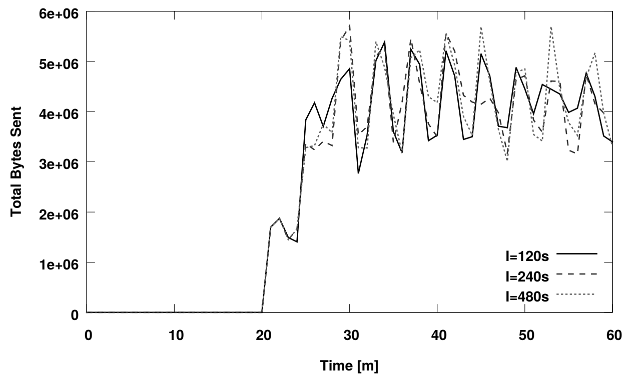
(a) A.2: Half-life Period.



(b) A.2: Average Delivery Time.

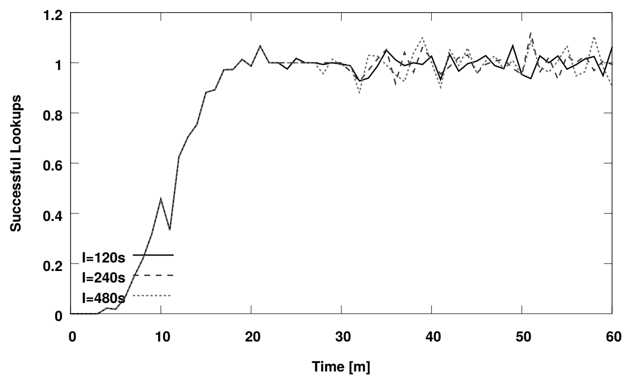


(c) A.2: Average Visited Nodes.

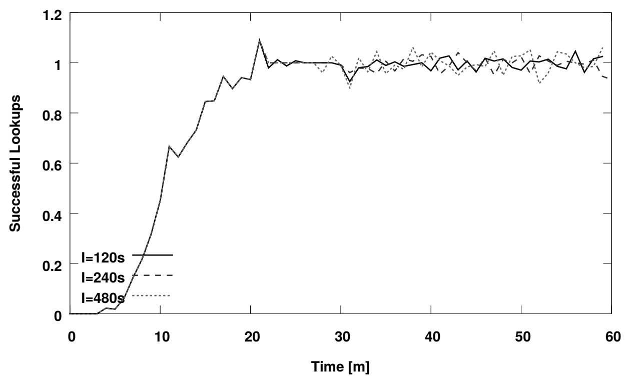


(d) A.2: Total Bytes Sent.

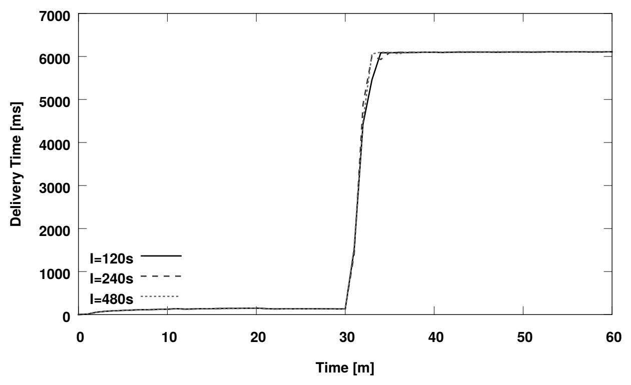
Fig. 15. A.2: Flooding Overlay in Overlay Stack.



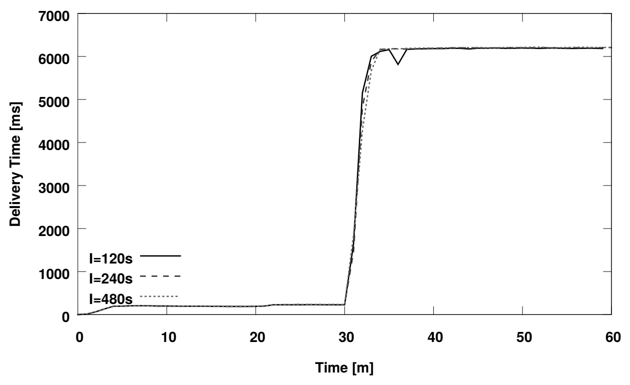
(a) A.3: Successful / Initiated Area Searches (Ratio).



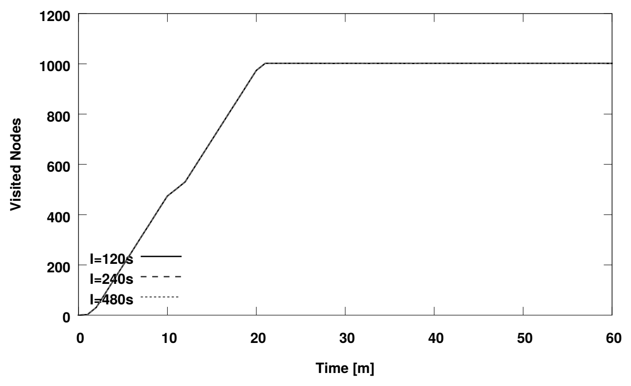
(a) A.3: Successful / Initiated Area Searches (Ratio).



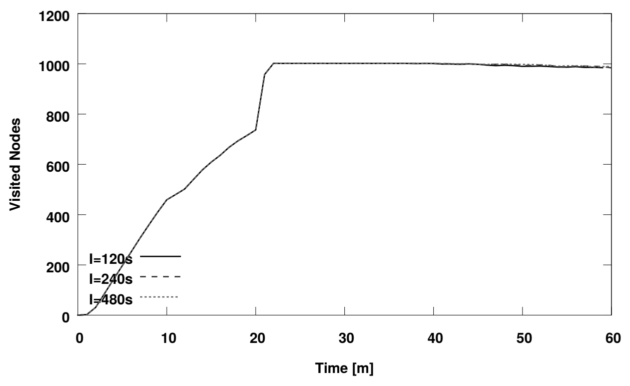
(b) A.3: Average Delivery Time.



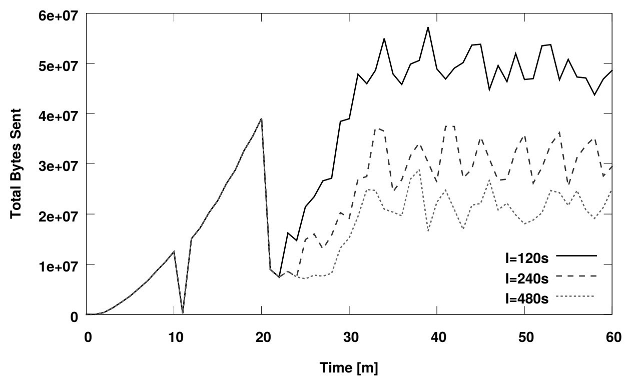
(b) A.3: Average Delivery Time.



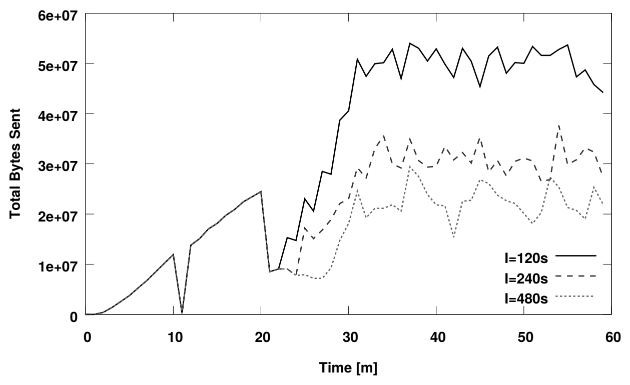
(c) A.3: Average Visited Nodes.



(c) A.3: Average Visited Nodes.



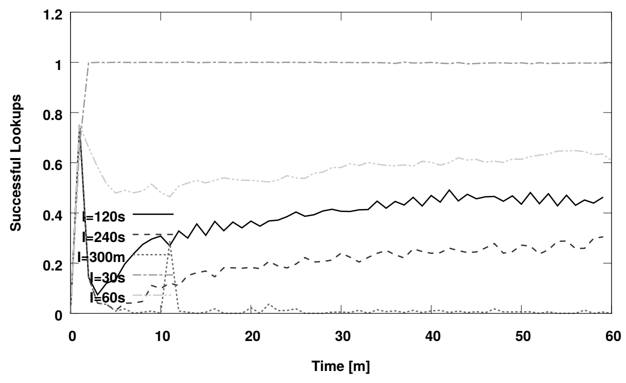
(d) A.3: Total Bytes Sent.



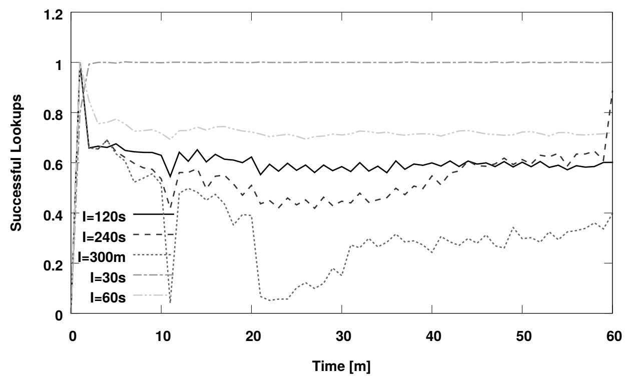
(d) A.3: Total Bytes Sent.

Fig. 16. A.3: Unmodified Geodemlia Overlay.

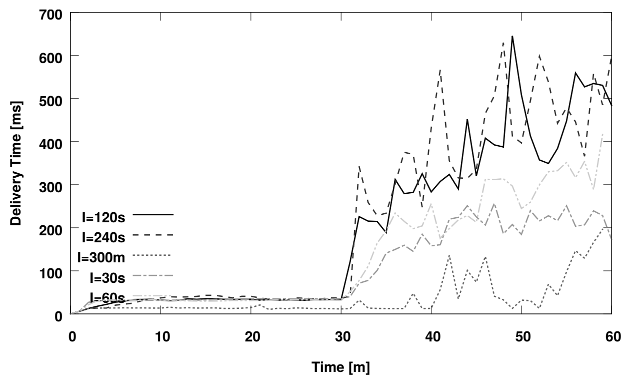
Fig. 17. A.3: Geodemlia Overlay in Overlay Stack.



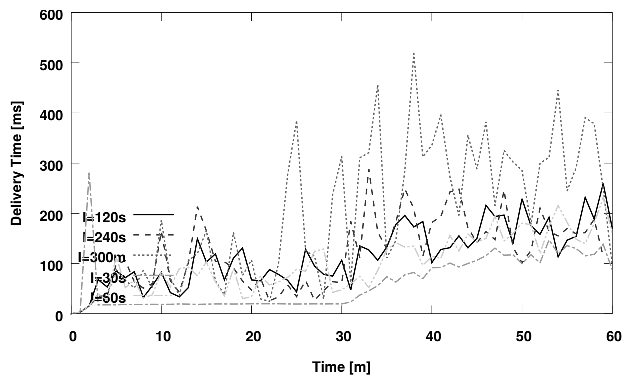
(a) B.1: Successful / Initiated Lookups (Ratio).



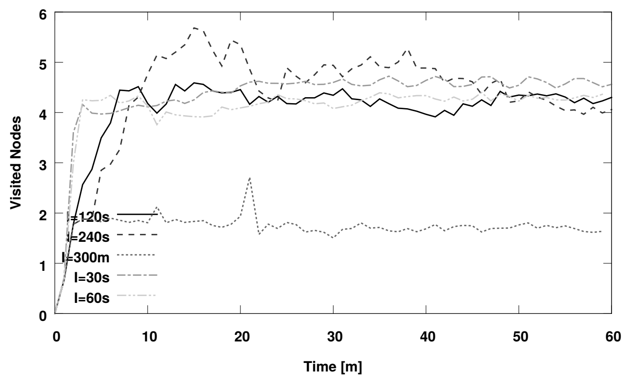
(a) B.1: Successful / Initiated Lookups (Ratio).



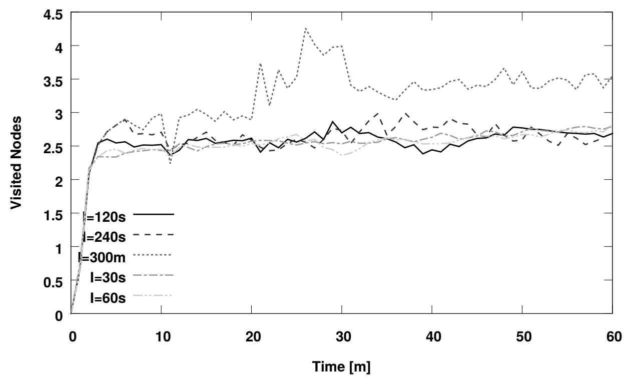
(b) B.1: Average Delivery Time.



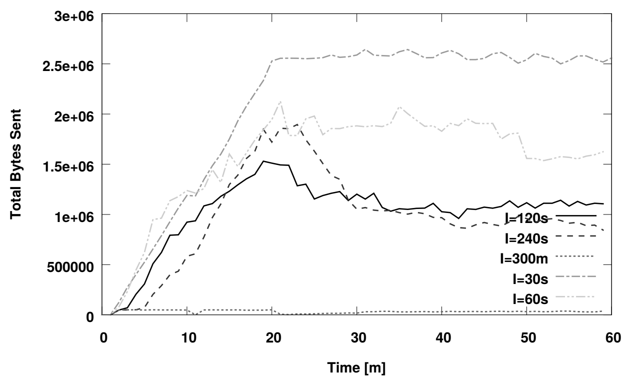
(b) B.1: Average Delivery Time.



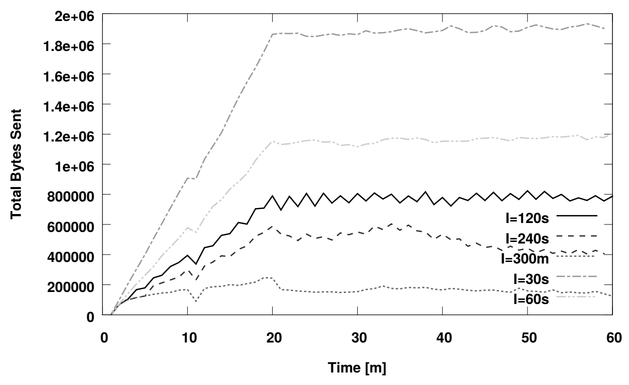
(c) B.1: Average Visited Nodes.



(c) B.1: Average Visited Nodes.



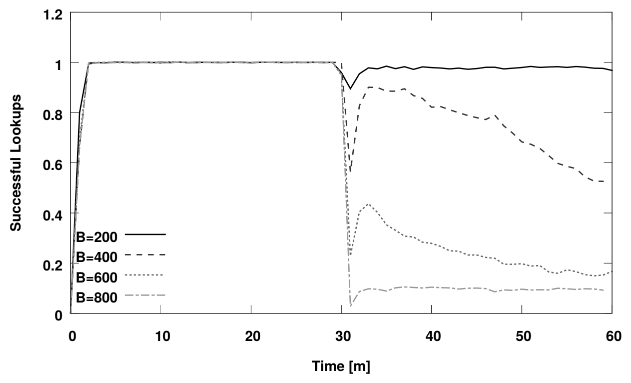
(d) B.1: Total Bytes Sent.



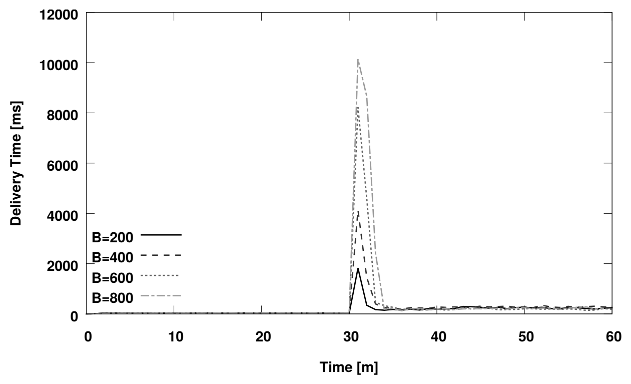
(d) B.1: Total Bytes Sent.

Fig. 18. B.1: Unmodified Chord.

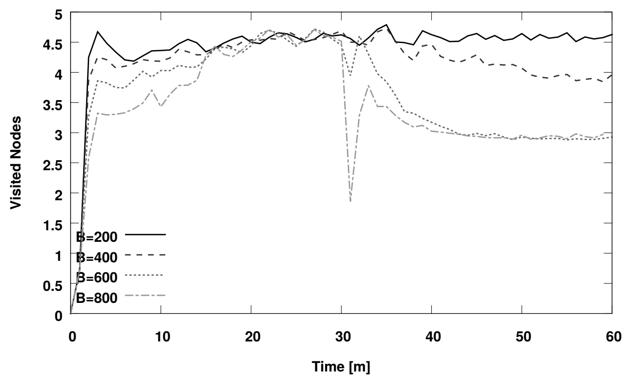
Fig. 19. B.1: Chord in Overlay Stack.



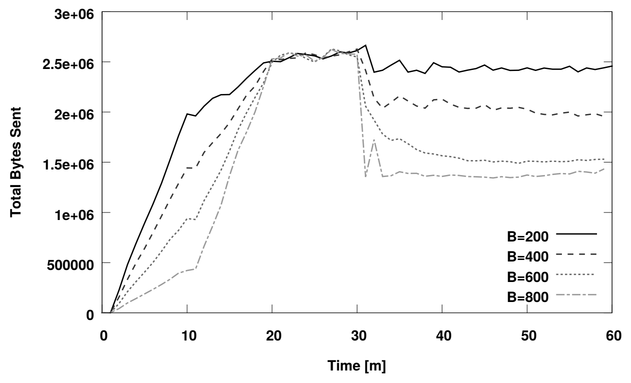
(a) C.1: Successful / Initiated Lookups (Ratio).



(b) C.1: Average Delivery Time.

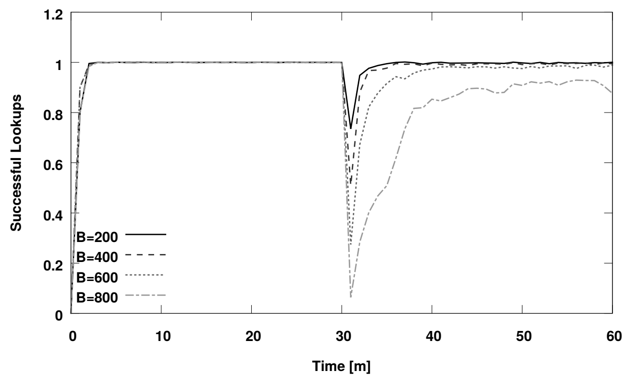


(c) C.1: Average Visited Nodes.

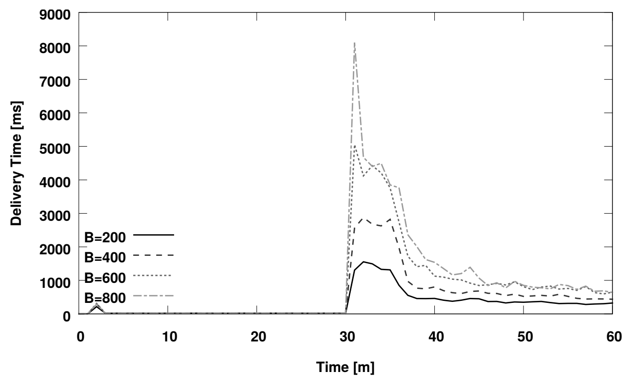


(d) C.1: Total Bytes Sent.

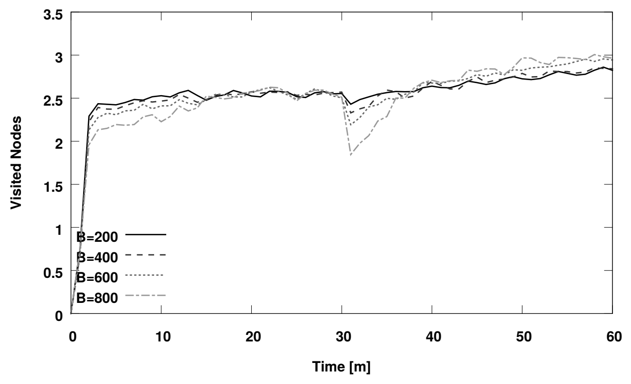
Fig. 20. C.1: Unmodified Chord.



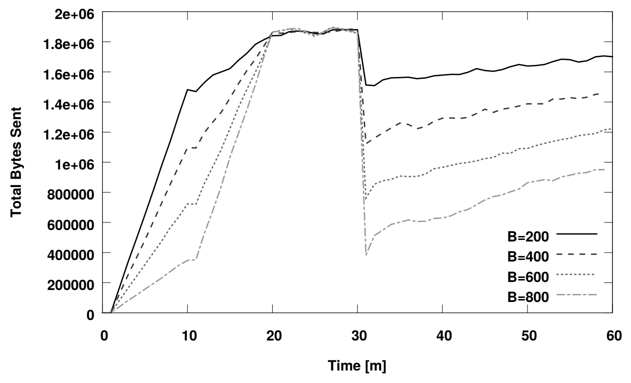
(a) C.1: Successful / Initiated Lookups (Ratio).



(b) C.1: Average Delivery Time.

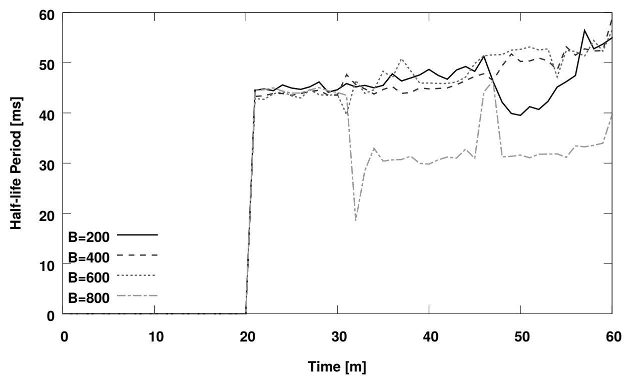


(c) C.1: Average Visited Nodes.

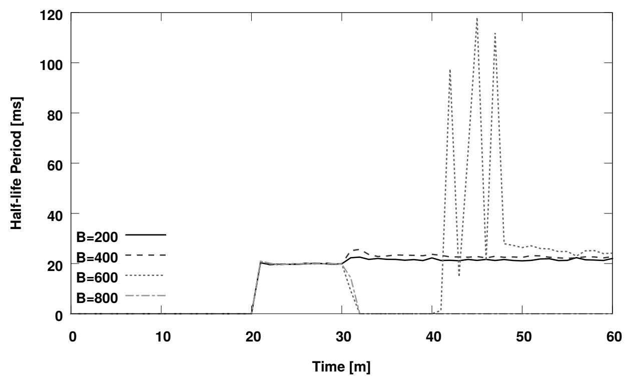


(d) C.1: Total Bytes Sent.

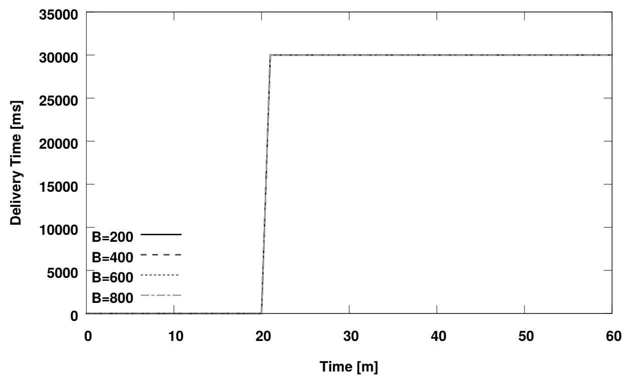
Fig. 21. C.1: Chord in Overlay Stack.



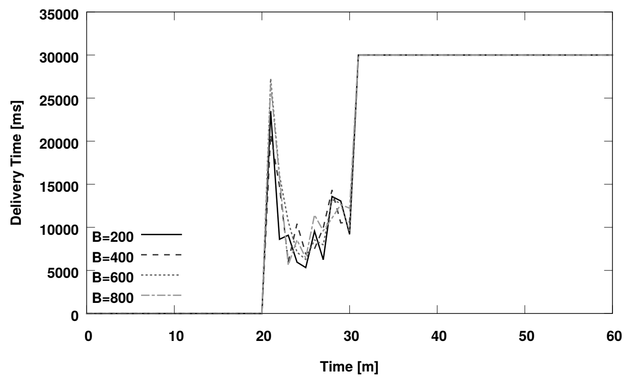
(a) C.2: Half-life Period.



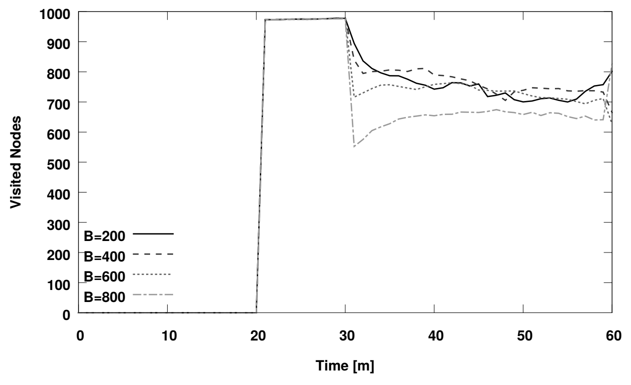
(a) C.2: Half-life Period.



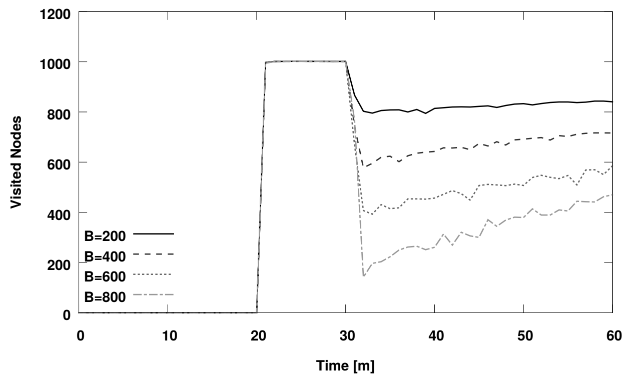
(b) C.2: Average Delivery Time.



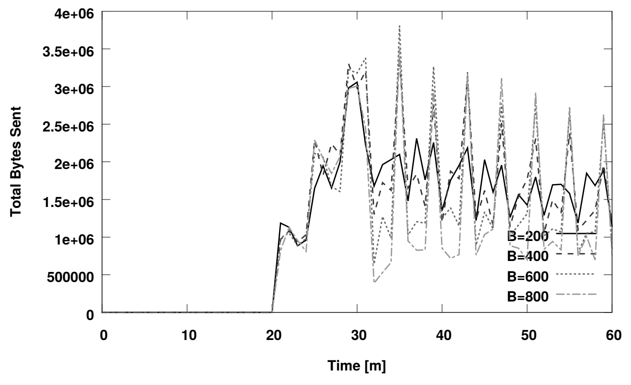
(b) C.2: Average Delivery Time.



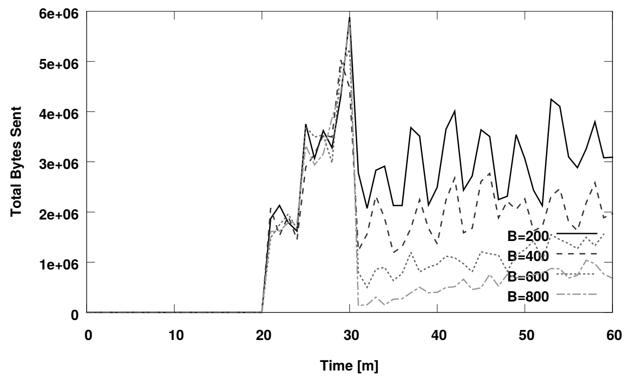
(c) C.2: Average Visited Nodes.



(c) C.2: Average Visited Nodes.



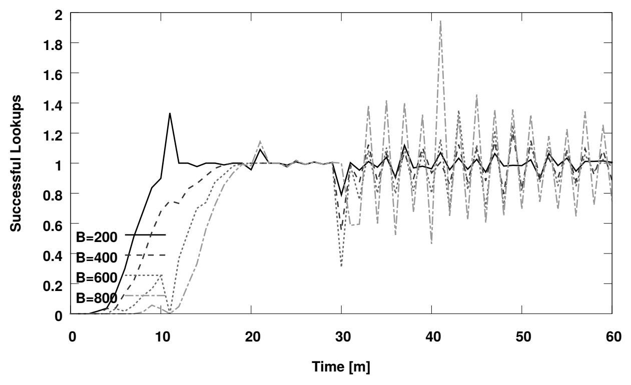
(d) C.2: Total Bytes Sent.



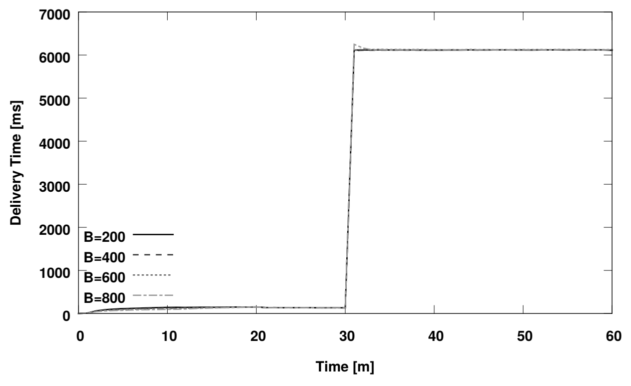
(d) C.2: Total Bytes Sent.

Fig. 22. C.2: Unmodified Flooding Overlay.

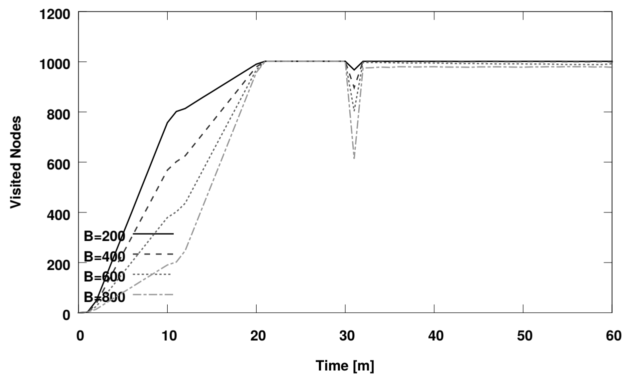
Fig. 23. C.2: Flooding Overlay in Overlay Stack.



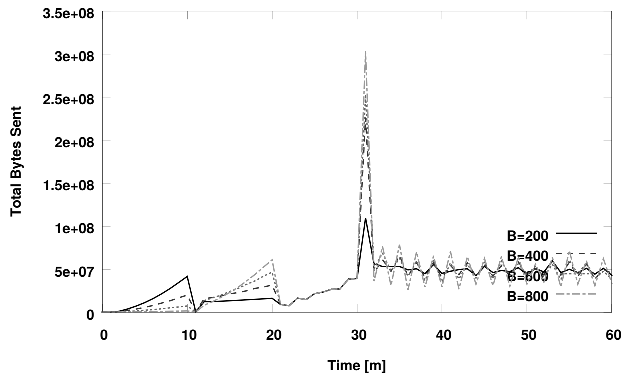
(a) C.3: Successful / Initiated Area Searches (Ratio).



(b) C.3: Average Delivery Time.

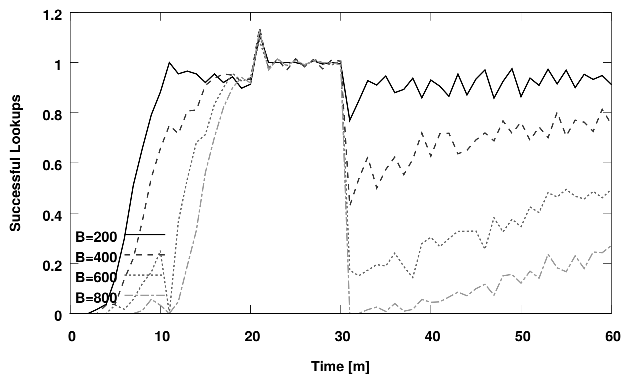


(c) C.3: Average Visited Nodes.

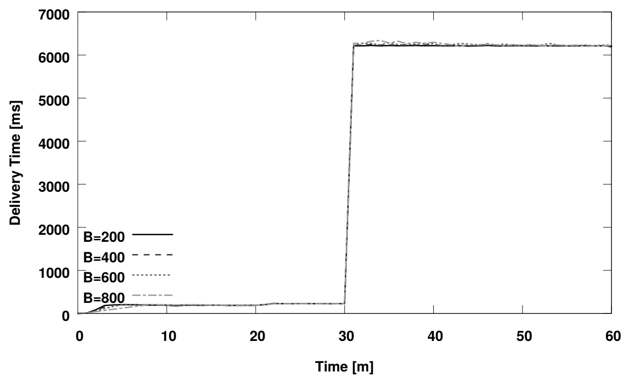


(d) C.3: Total Bytes Sent.

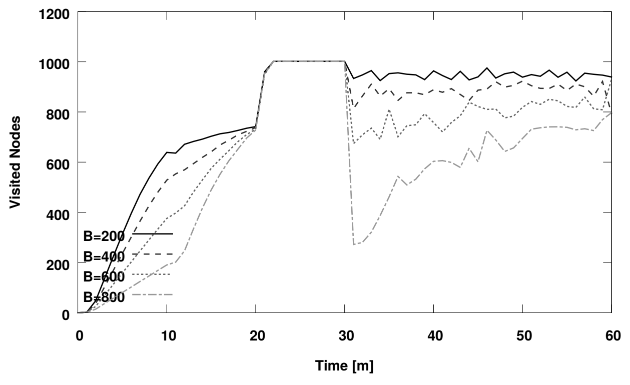
Fig. 24. C.3: Unmodified Geodemlia Overlay.



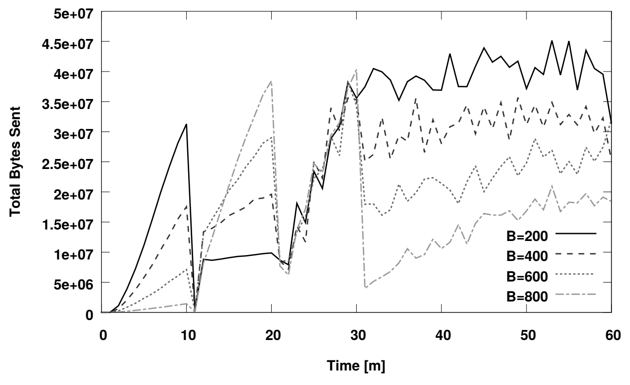
(a) C.3: Successful / Initiated Area Searches (Ratio).



(b) C.3: Average Delivery Time.

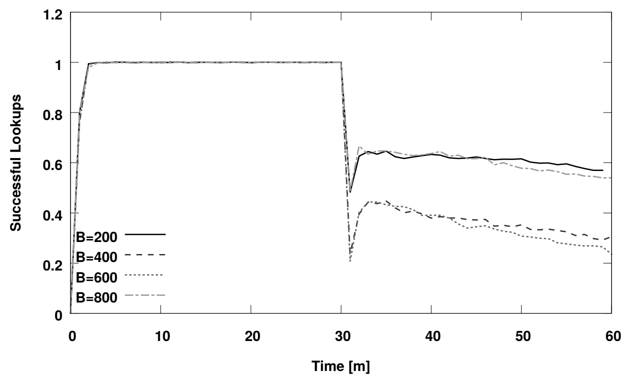


(c) C.3: Average Visited Nodes.

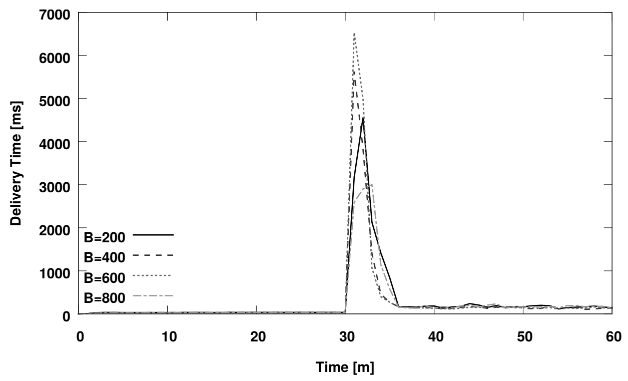


(d) C.3: Total Bytes Sent.

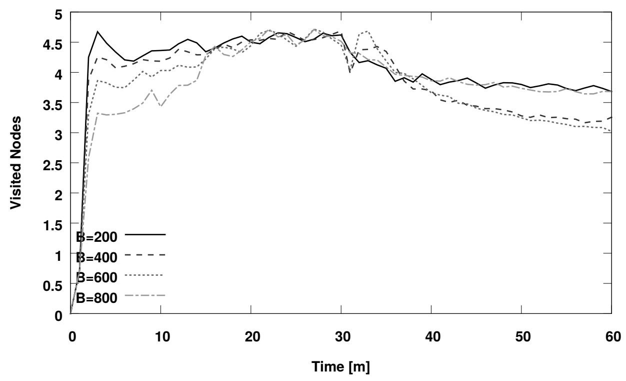
Fig. 25. C.3: Geodemlia Overlay in Overlay Stack.



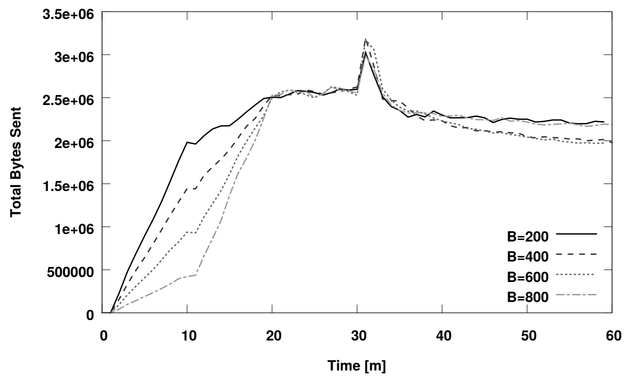
(a) D.1: Successful / Initiated Lookups (Ratio).



(b) D.1: Average Delivery Time.

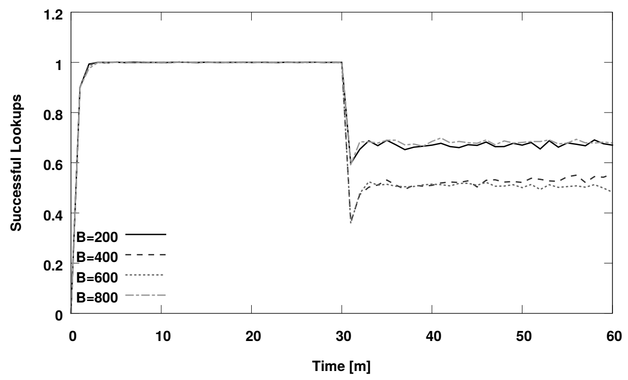


(c) D.1: Average Visited Nodes.

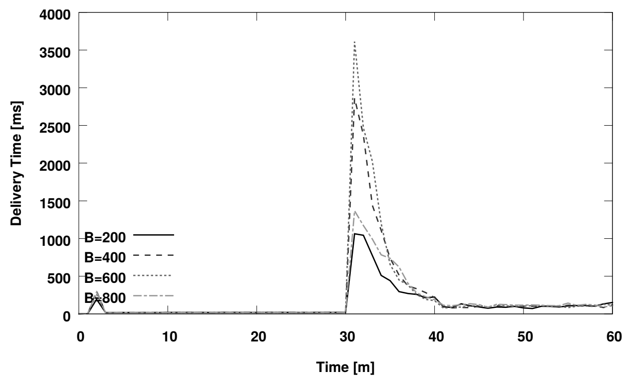


(d) D.1: Total Bytes Sent.

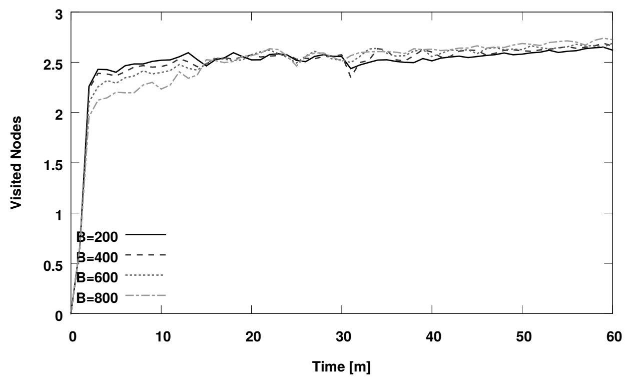
Fig. 26. D.1: Unmodified Chord without Ring Reunion Algorithm.



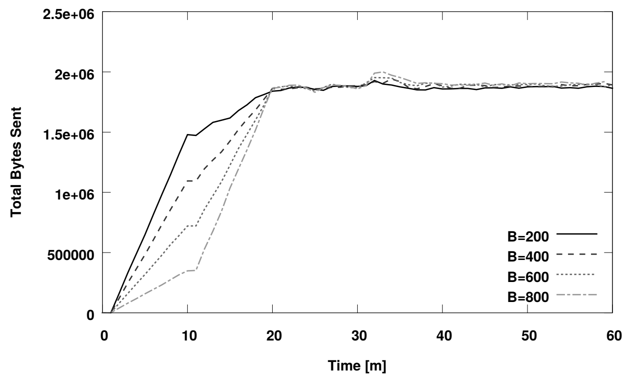
(a) D.1: Successful / Initiated Lookups (Ratio).



(b) D.1: Average Delivery Time.

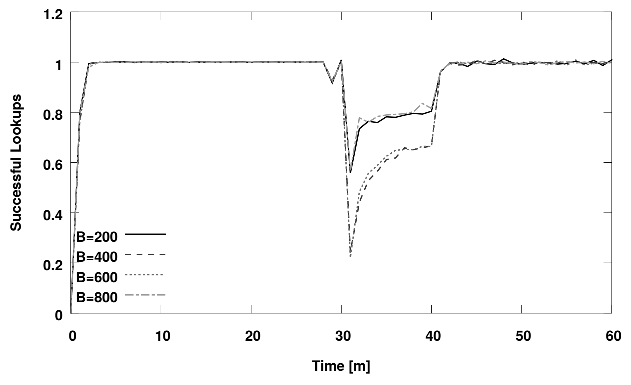


(c) D.1: Average Visited Nodes.

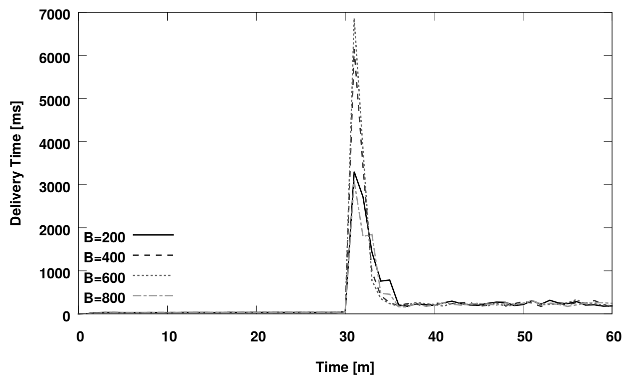


(d) D.1: Total Bytes Sent.

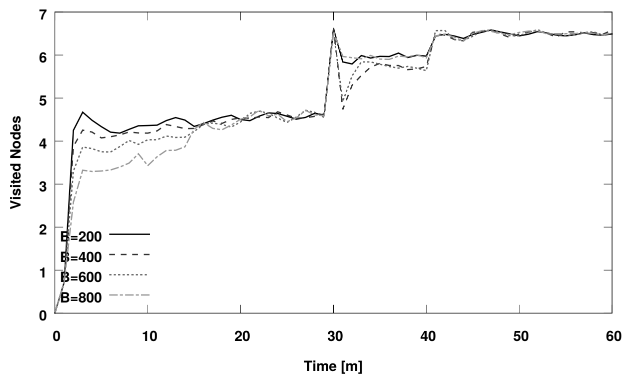
Fig. 27. D.1: Chord in Overlay Stack without Ring Reunion Algorithm.



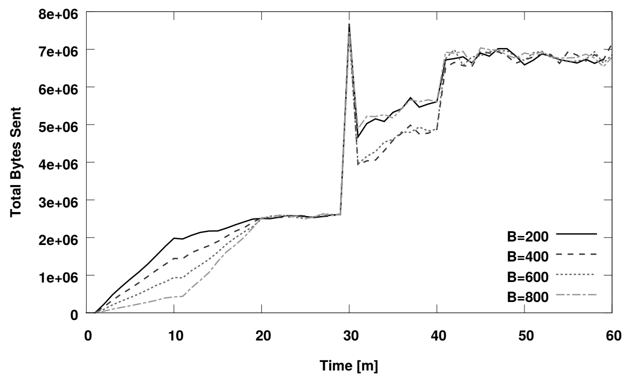
(a) D.2: Successful / Initiated Lookups (Ratio).



(b) D.2: Average Delivery Time.

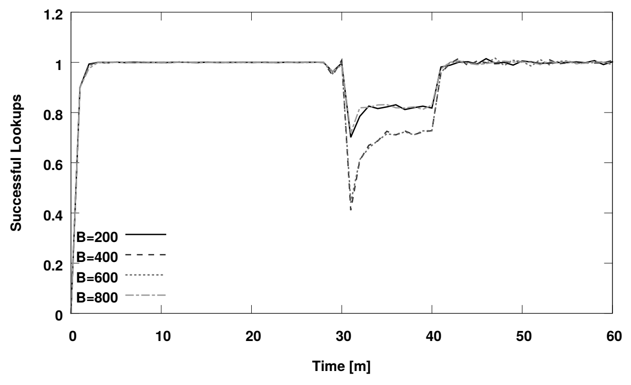


(c) D.2: Average Visited Nodes.

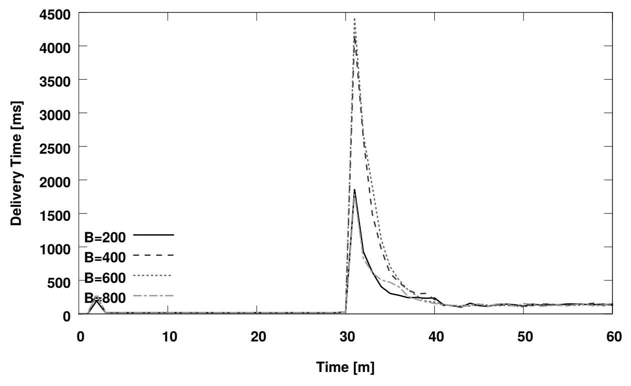


(d) D.2: Total Bytes Sent.

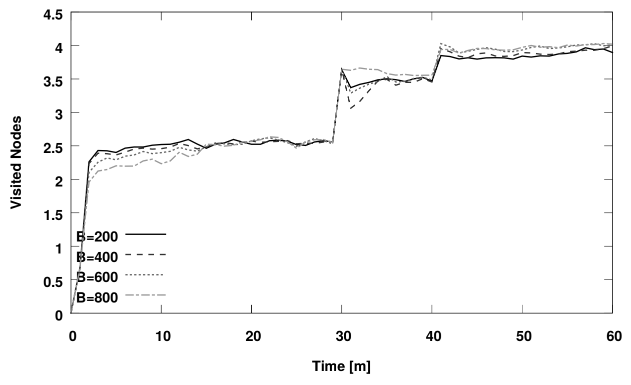
Fig. 28. D.2: Unmodified Chord with Ring Reunion Algorithm.



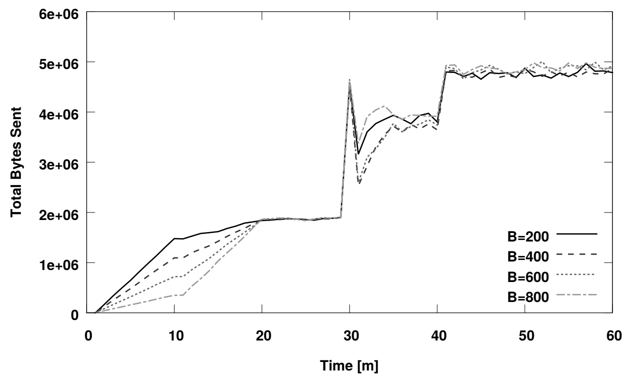
(a) D.2: Successful / Initiated Lookups (Ratio).



(b) D.2: Average Delivery Time.

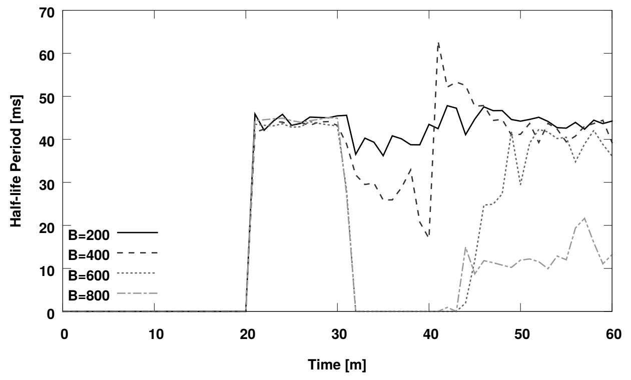


(c) D.2: Average Visited Nodes.

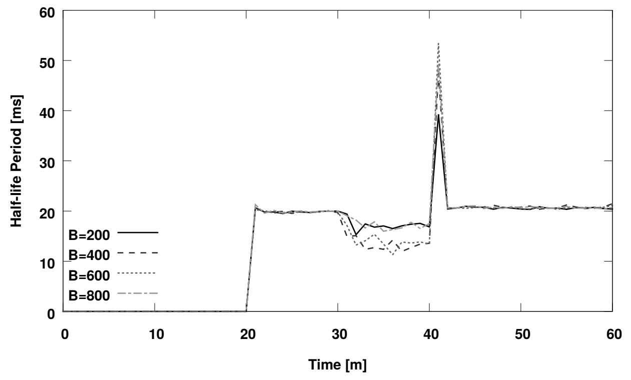


(d) D.2: Total Bytes Sent.

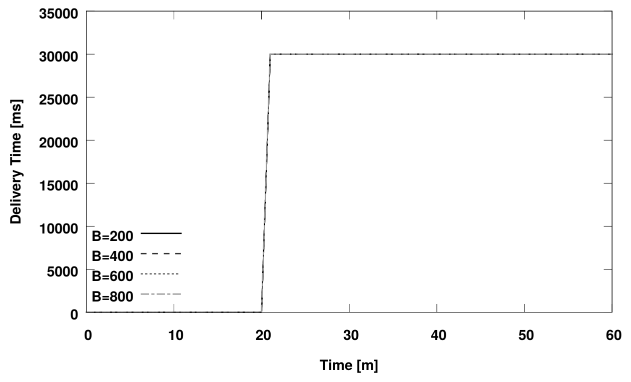
Fig. 29. D.2: Chord in Overlay Stack with Ring Reunion Algorithm.



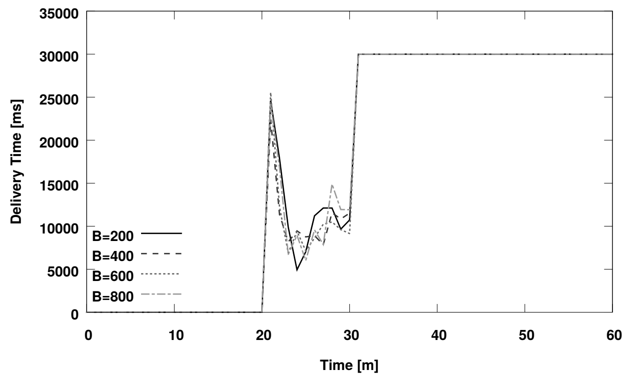
(a) D.3: Half-life Period.



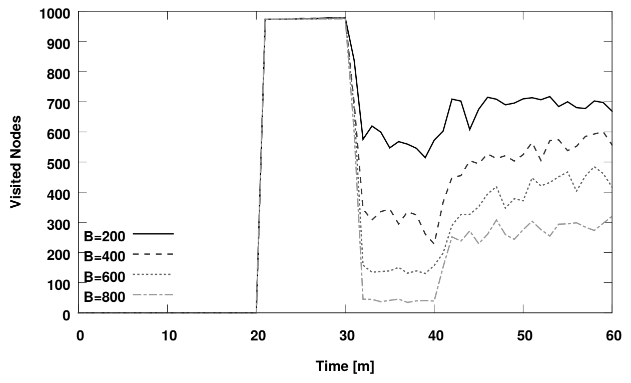
(a) D.3: Half-life Period.



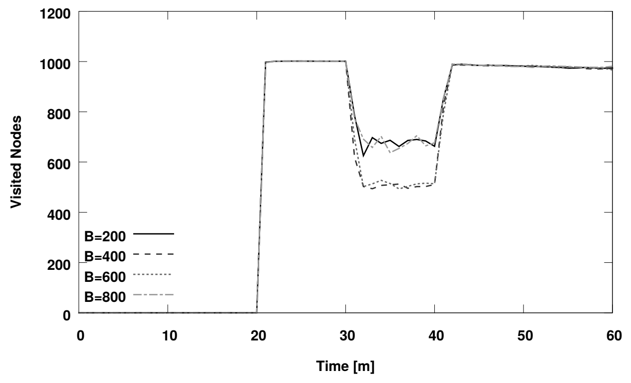
(b) D.3: Average Delivery Time.



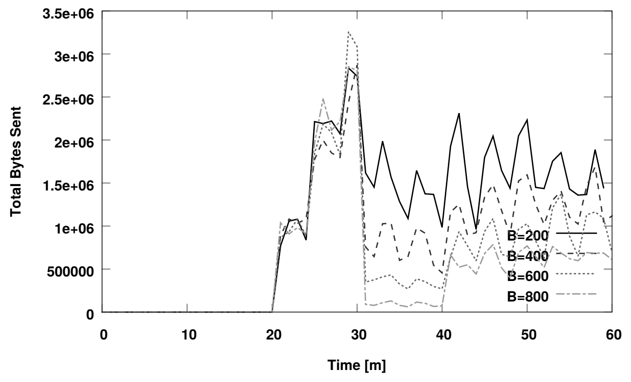
(b) D.3: Average Delivery Time.



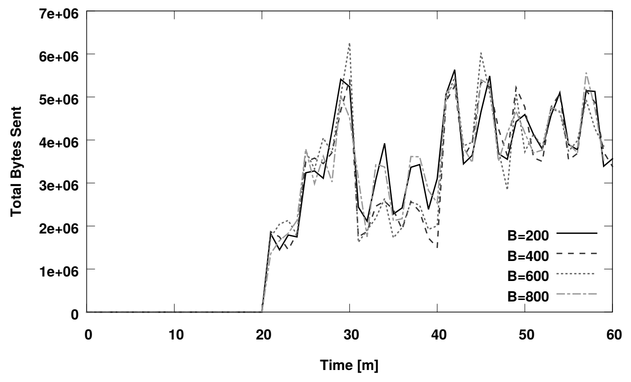
(c) D.3: Average Visited Nodes.



(c) D.3: Average Visited Nodes.



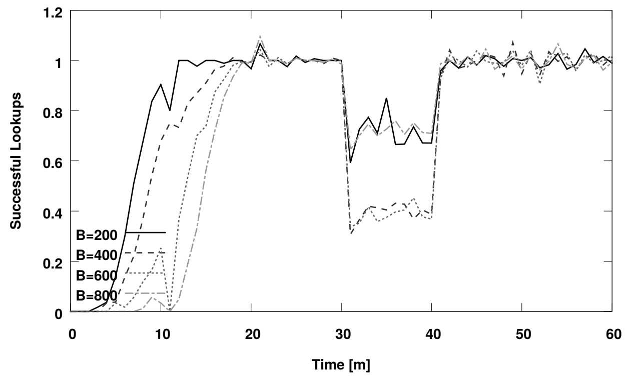
(d) D.3: Total Bytes Sent.



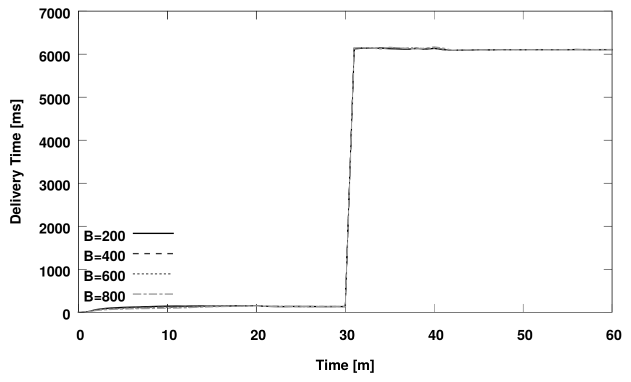
(d) D.3: Total Bytes Sent.

Fig. 30. D.3: Unmodified Flooding Overlay without Ring Reunion Algorithm.

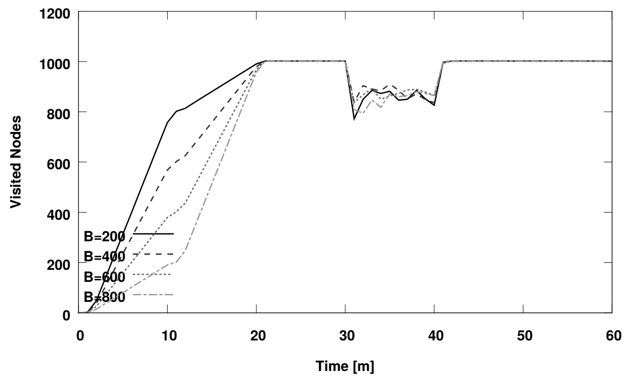
Fig. 31. D.3: Flooding Overlay in Overlay Stack with Ring Reunion Algorithm.



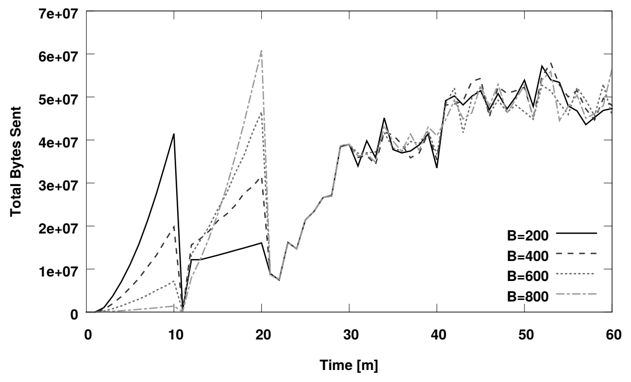
(a) D.4: Successful / Initiated Area Searches (Ratio).



(b) D.4: Average Delivery Time.

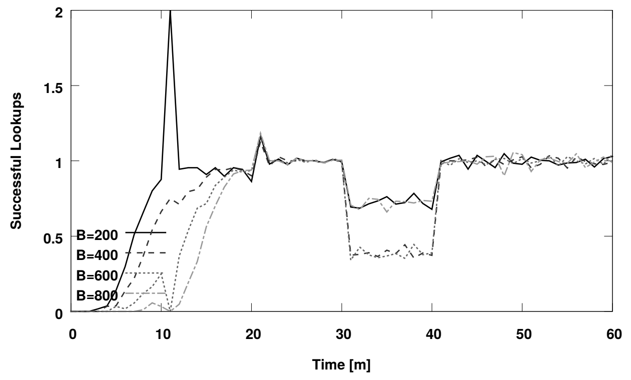


(c) D.4: Average Visited Nodes.

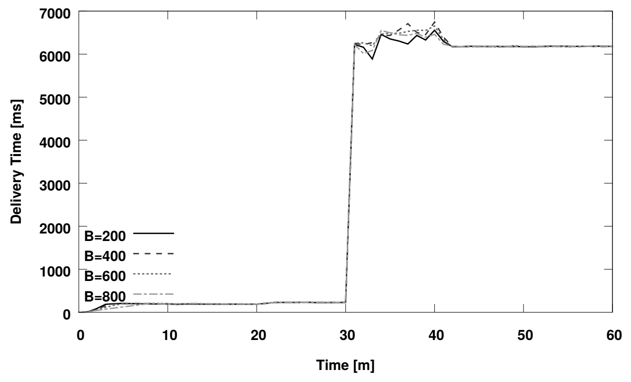


(d) D.4: Total Bytes Sent.

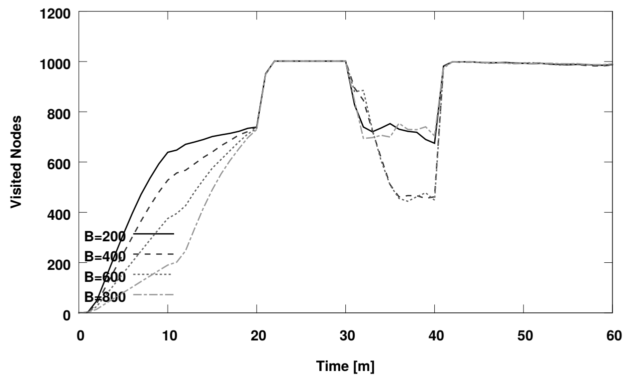
Fig. 32. D.4: Unmodified Geodemlia Overlay without Ring Reunion Algorithm.



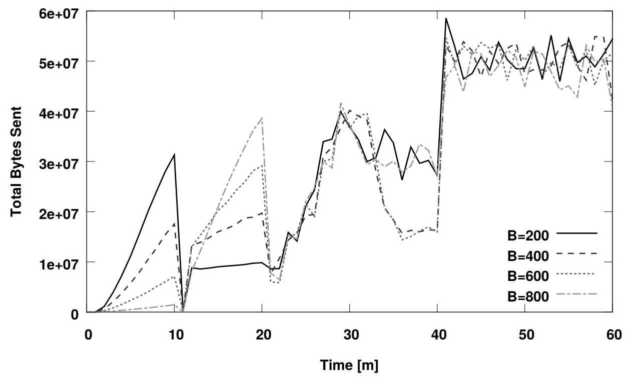
(a) D.4: Successful / Initiated Area Searches (Ratio).



(b) D.4: Average Delivery Time.

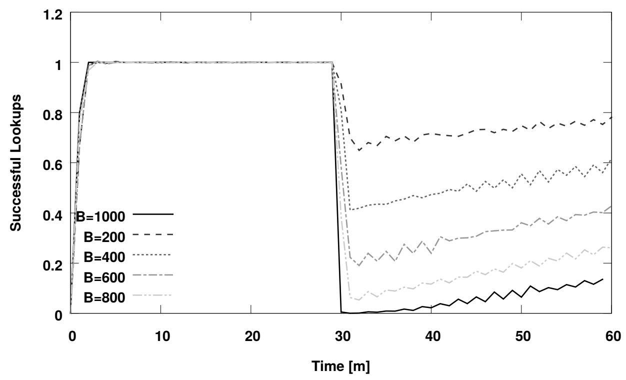


(c) D.4: Average Visited Nodes.

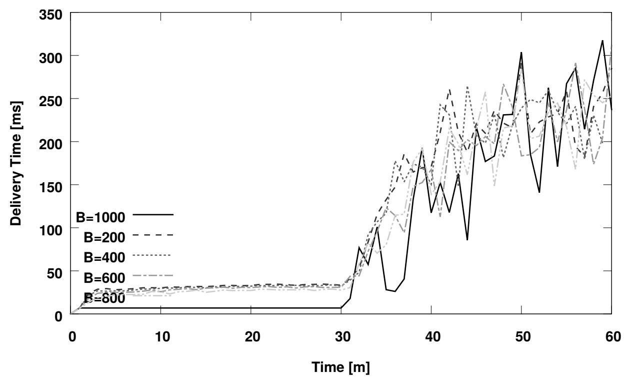


(d) D.4: Total Bytes Sent.

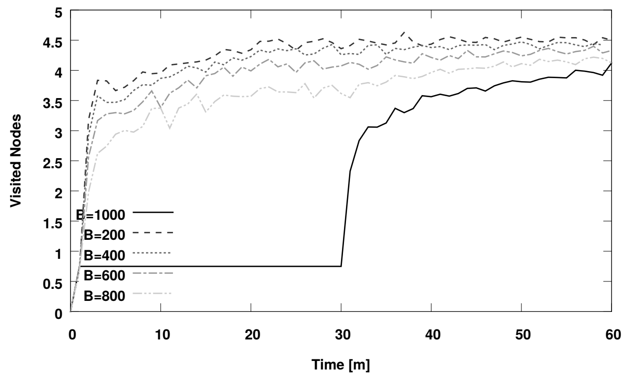
Fig. 33. D.4: Geodemlia Overlay in Overlay Stack with Ring Reunion Algorithm.



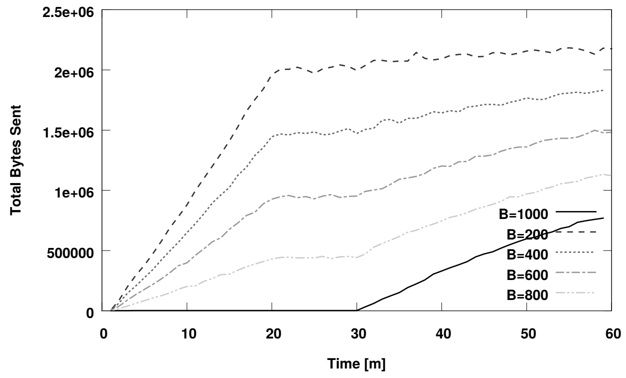
(a) E.1: Successful / Initiated Lookups (Ratio).



(b) E.1: Average Delivery Time.

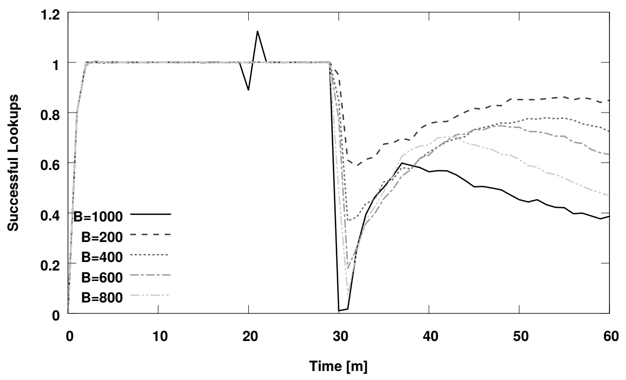


(c) E.1: Average Visited Nodes.

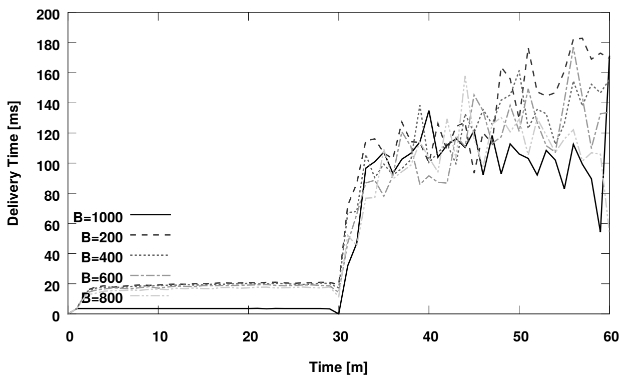


(d) E.1: Total Bytes Sent.

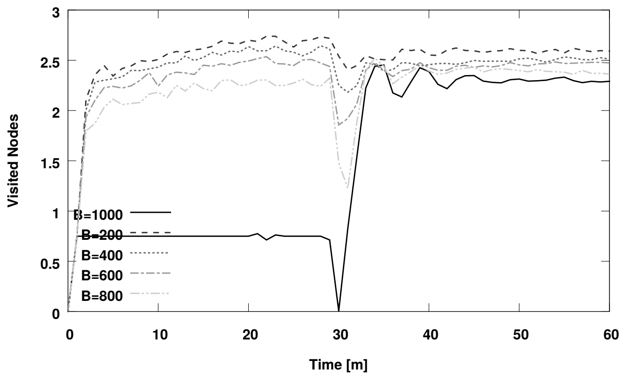
Fig. 34. E.1: Unmodified Chord.



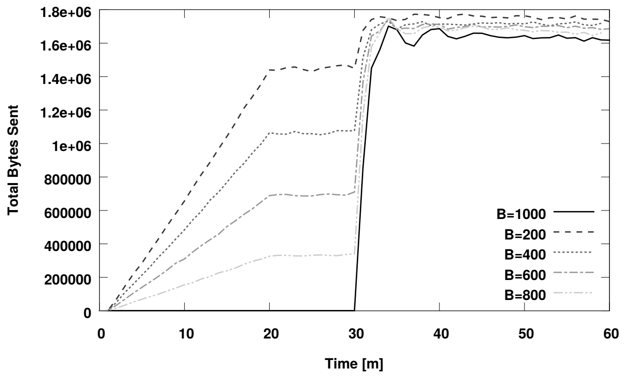
(a) E.1: Successful / Initiated Lookups (Ratio).



(b) E.1: Average Delivery Time.

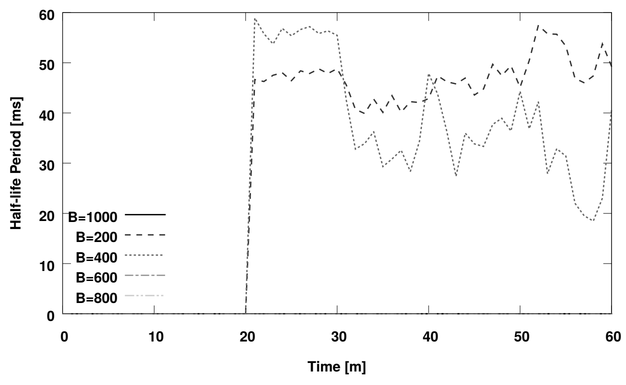


(c) E.1: Average Visited Nodes.

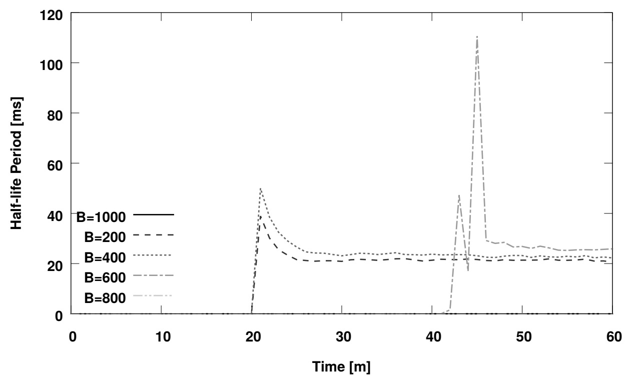


(d) E.1: Total Bytes Sent.

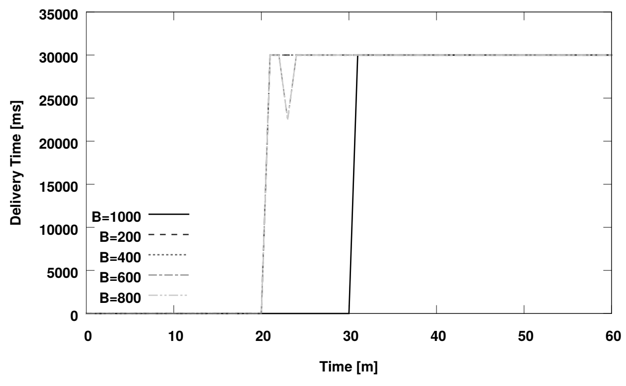
Fig. 35. E.1: Chord in Overlay Stack.



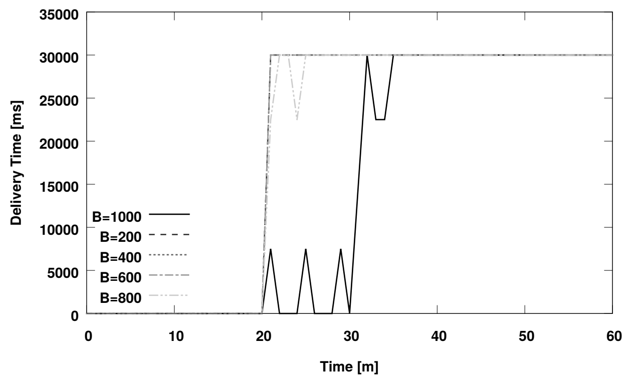
(a) E.2: Half-life Period.



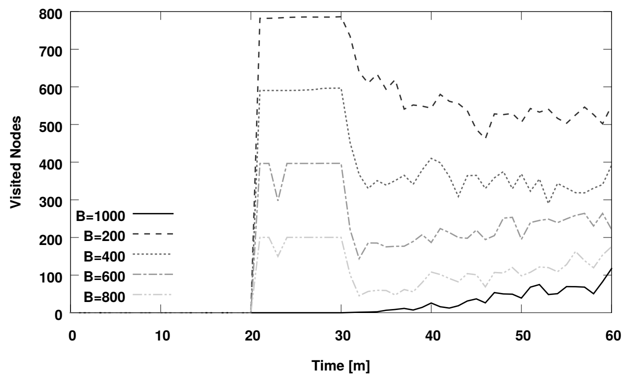
(a) E.2: Half-life Period.



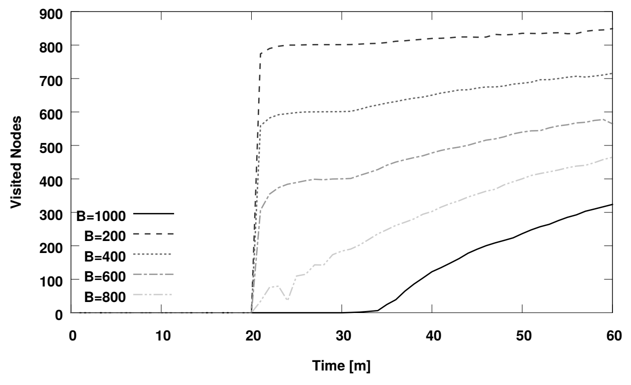
(b) E.2: Average Delivery Time.



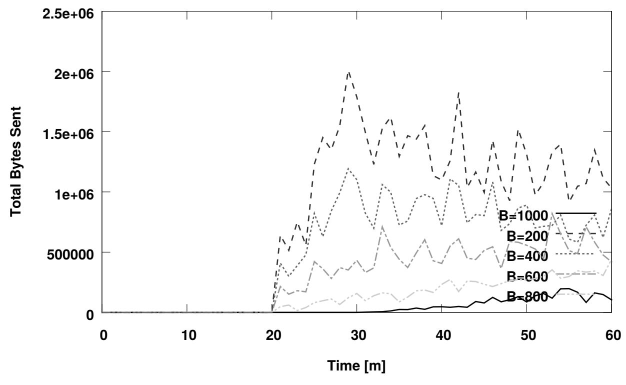
(b) E.2: Average Delivery Time.



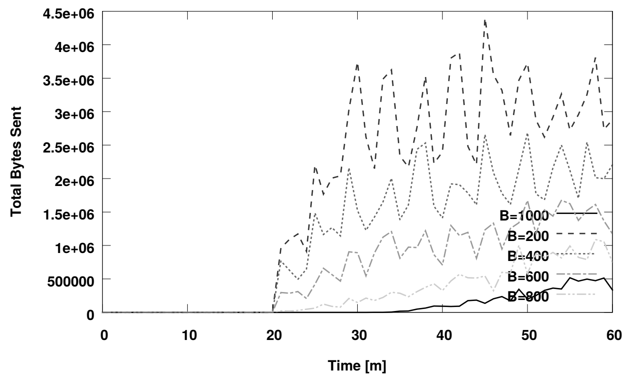
(c) E.2: Average Visited Nodes.



(c) E.2: Average Visited Nodes.



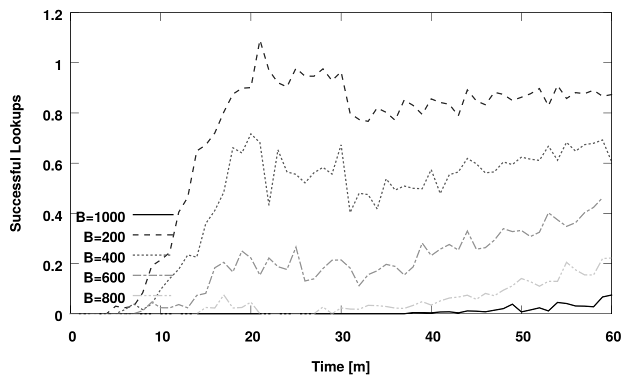
(d) E.2: Total Bytes Sent.



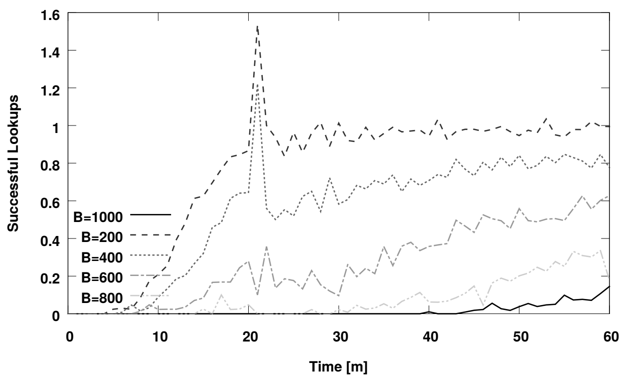
(d) E.2: Total Bytes Sent.

Fig. 36. E.2: Unmodified Flooding Overlay.

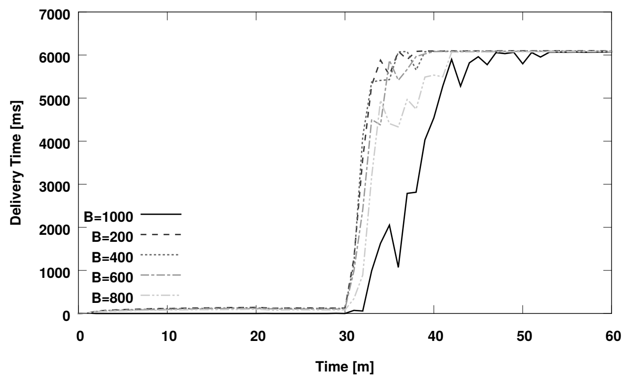
Fig. 37. E.2: Flooding Overlay in Overlay Stack.



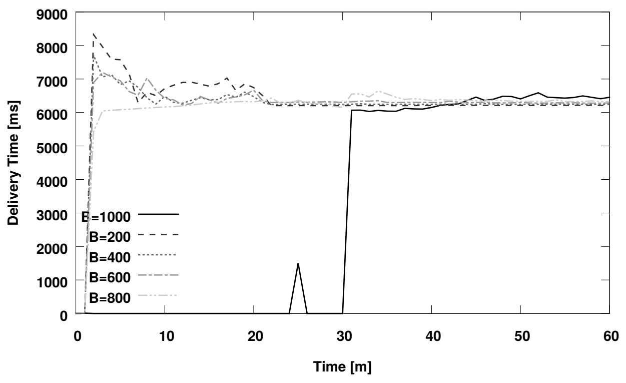
(a) E.3: Successful / Initiated Area Searches (Ratio).



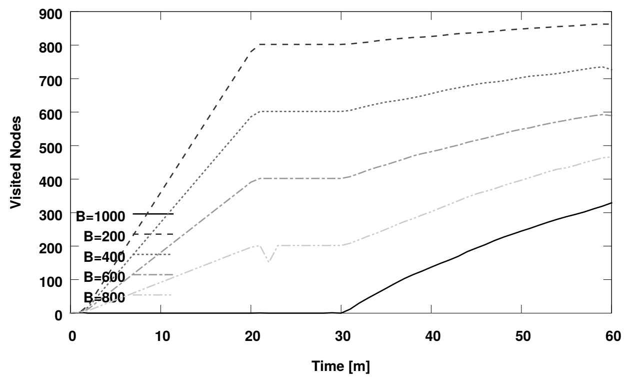
(a) E.3: Successful / Initiated Area Searches (Ratio).



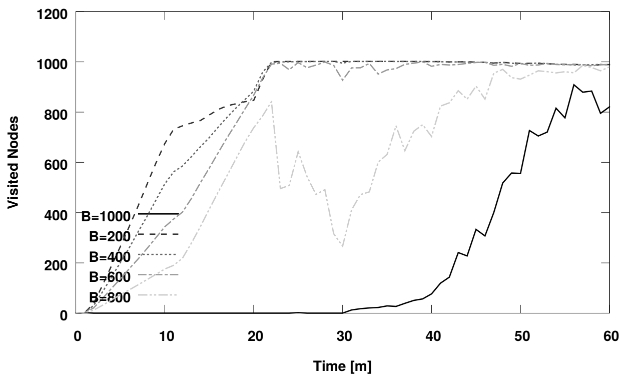
(b) E.3: Average Delivery Time.



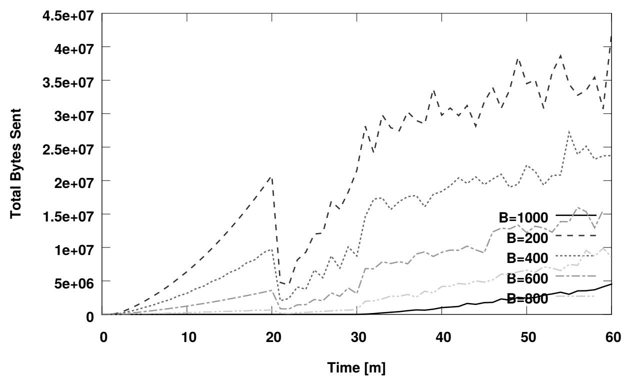
(b) E.3: Average Delivery Time.



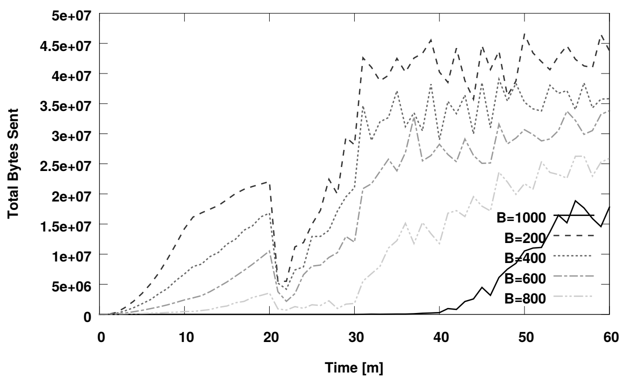
(c) E.3: Average Visited Nodes.



(c) E.3: Average Visited Nodes.



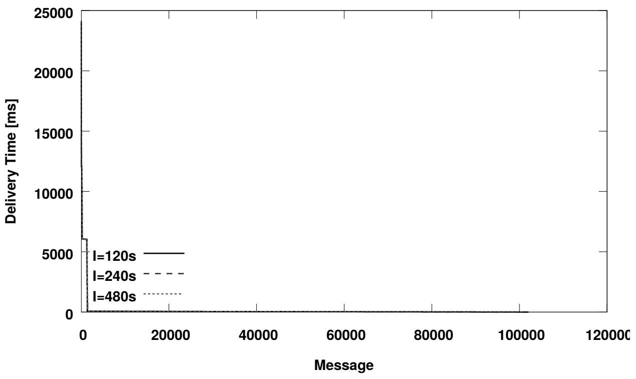
(d) E.3: Total Bytes Sent.



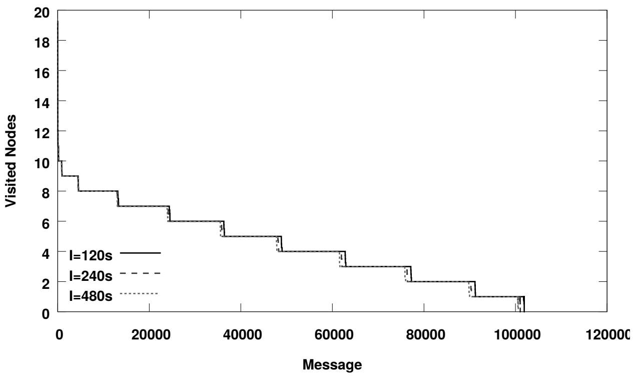
(d) E.3: Total Bytes Sent.

Fig. 38. E.3: Unmodified Geodemlia Overlay.

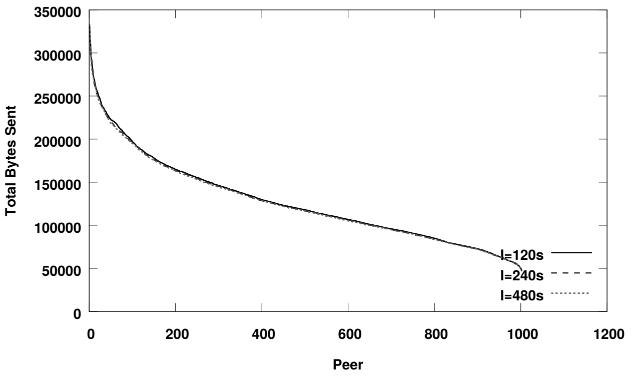
Fig. 39. E.3: Geodemlia Overlay in Overlay Stack.



(a) A.1: Delivery Time per Message.

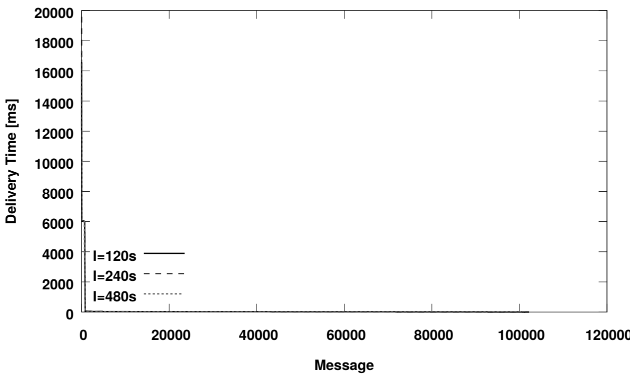


(b) A.1: Visited Nodes per Message.

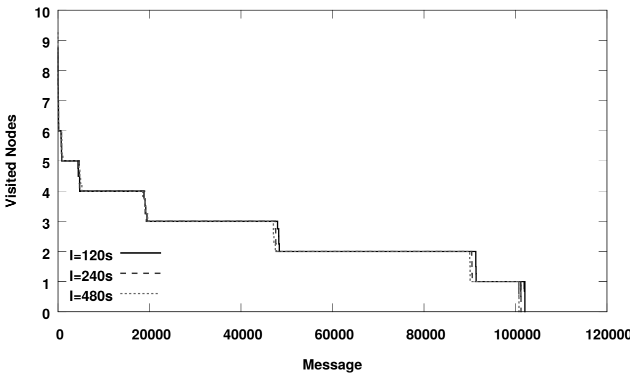


(c) A.1: Total Bytes Sent per Peer.

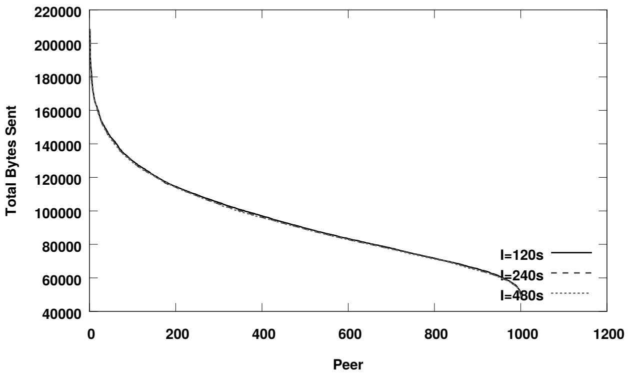
Fig. 40. A.1: Unmodified Chord.



(a) A.1: Delivery Time per Message.

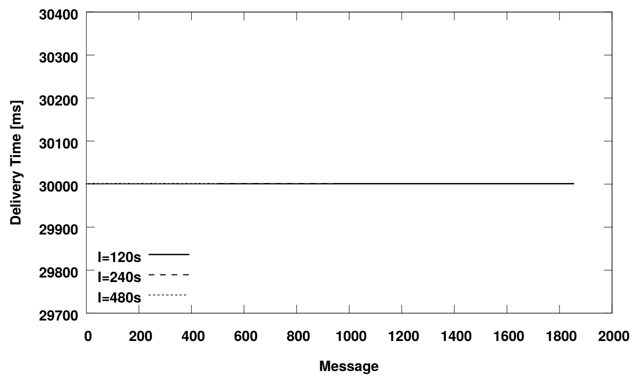


(b) A.1: Visited Nodes per Message.

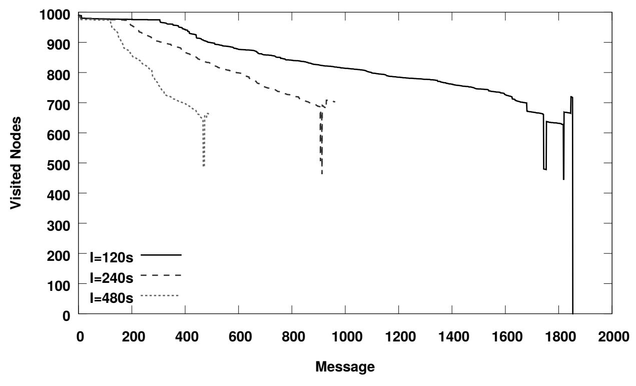


(c) A.1: Total Bytes Sent per Peer.

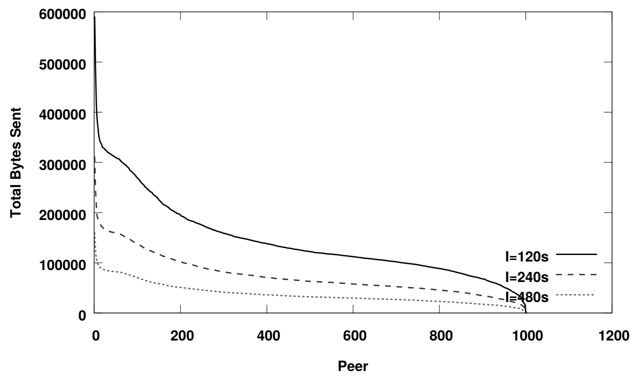
Fig. 41. A.1: Chord in Overlay Stack.



(a) A.2: Delivery Time per Message.

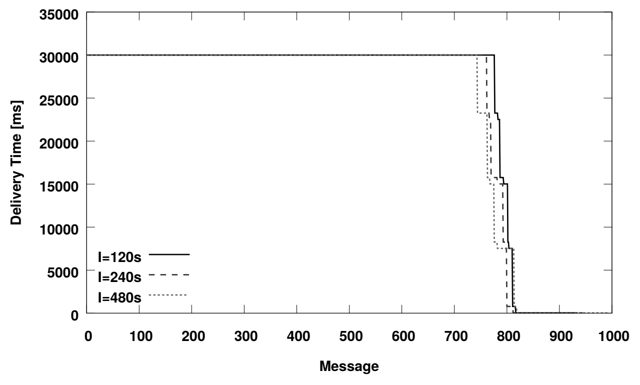


(b) A.2: Visited Nodes per Message.

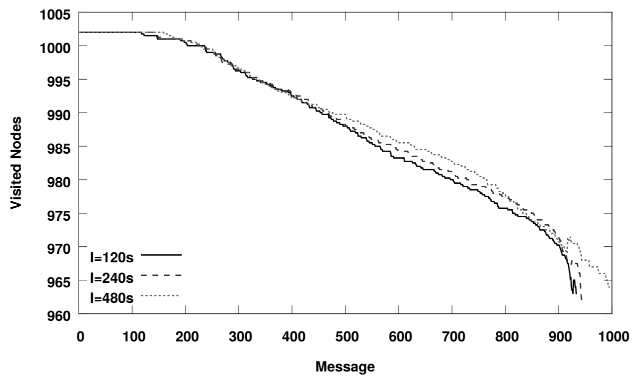


(c) A.2: Total Bytes Sent per Peer.

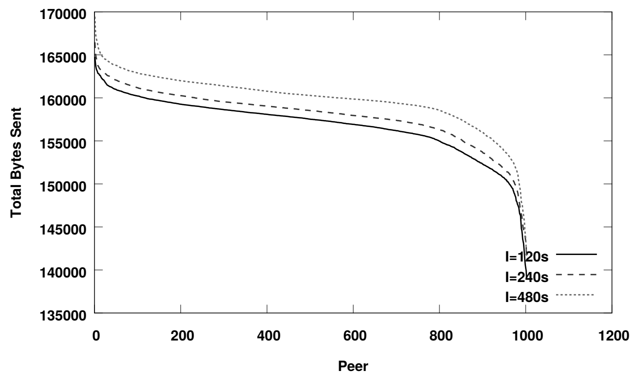
Fig. 42. A.2: Unmodified Flooding Overlay.



(a) A.2: Delivery Time per Message.

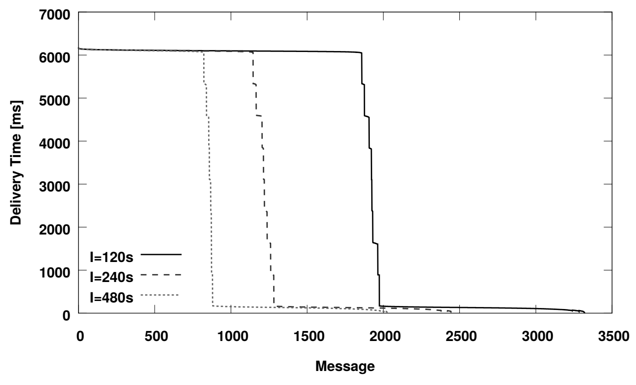


(b) A.2: Visited Nodes per Message.

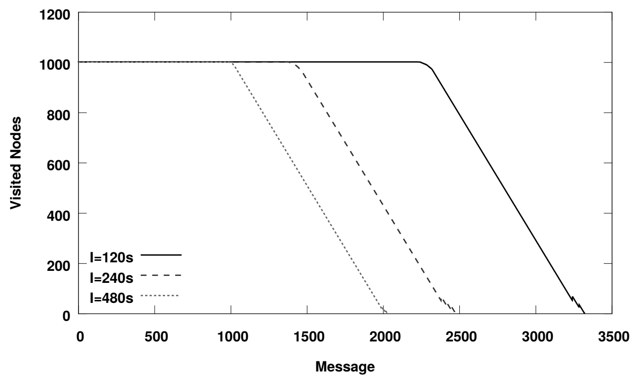


(c) A.2: Total Bytes Sent per Peer.

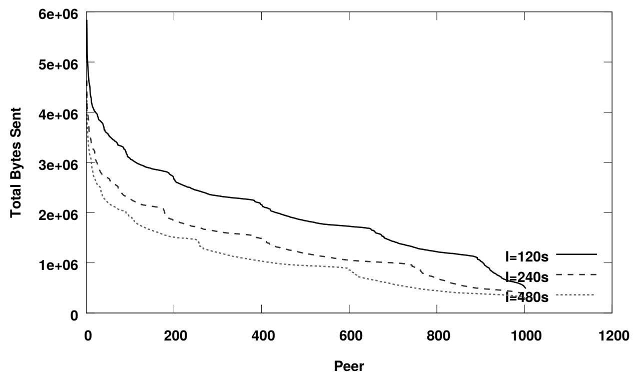
Fig. 43. A.2: Flooding Overlay in Overlay Stack.



(a) A.3: Delivery Time per Message.

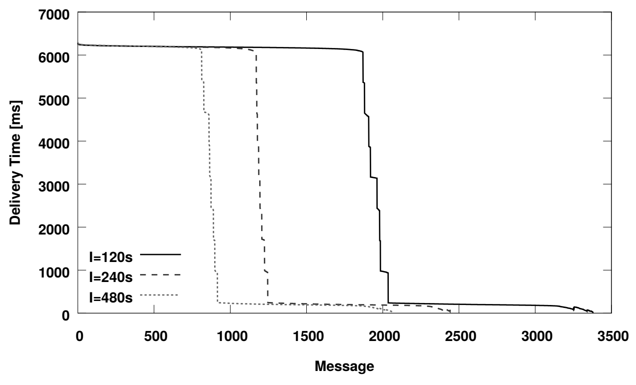


(b) A.3: Visited Nodes per Message.

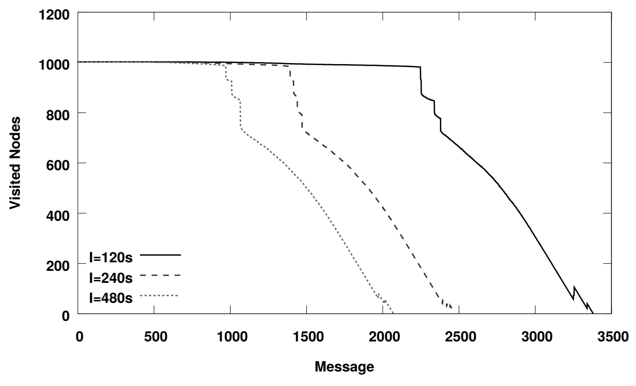


(c) A.3: Total Bytes Sent per Peer.

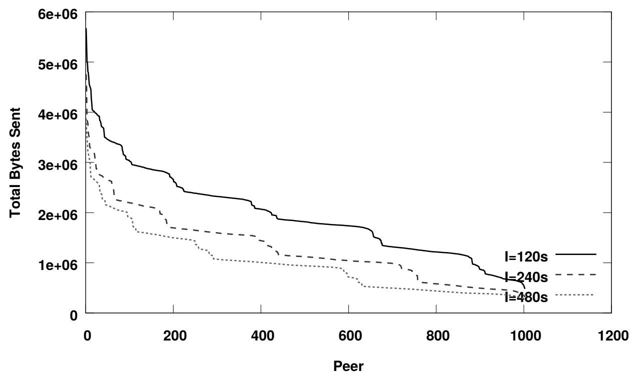
Fig. 44. A.3: Unmodified Geodemlia Overlay.



(a) A.3: Delivery Time per Message.

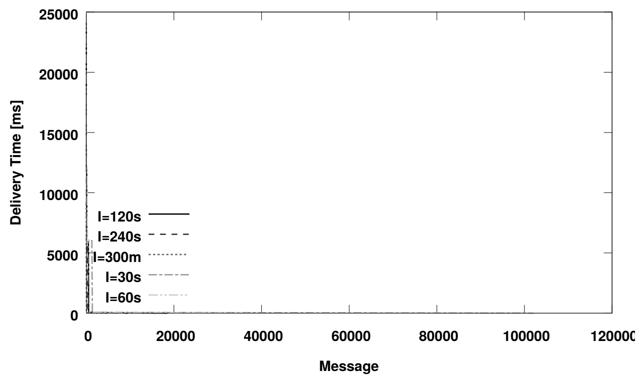


(b) A.3: Visited Nodes per Message.

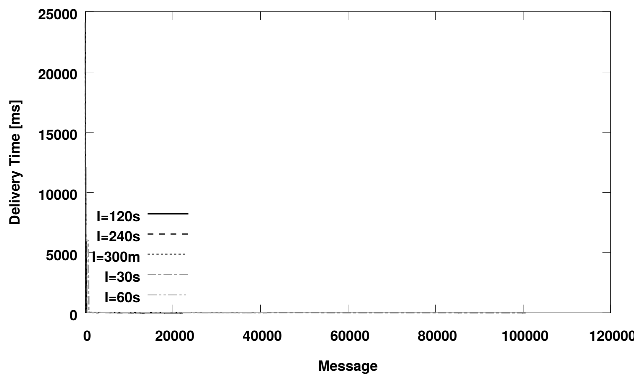


(c) A.3: Total Bytes Sent per Peer.

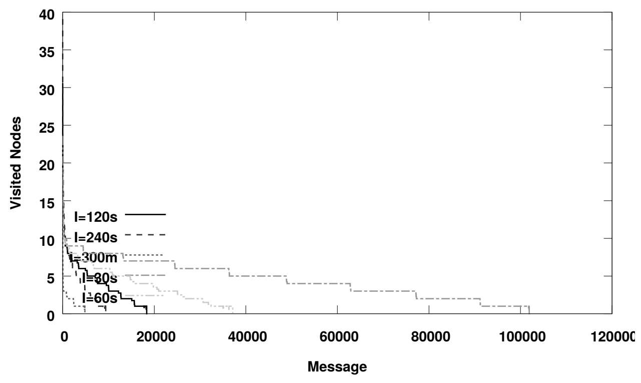
Fig. 45. A.3: Geodemlia Overlay in Overlay Stack.



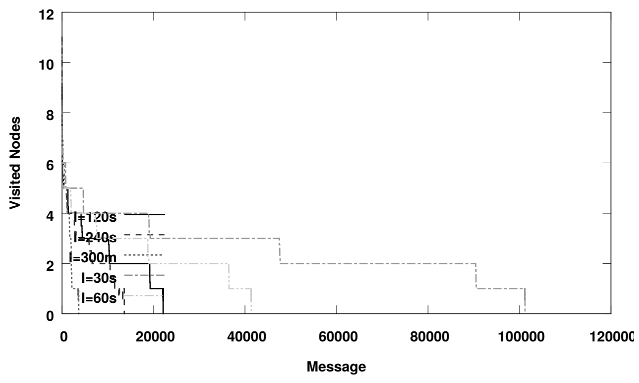
(a) B.1: Delivery Time per Message.



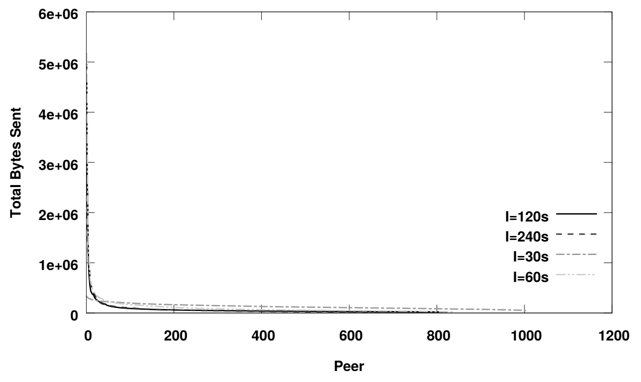
(a) B.1: Delivery Time per Message.



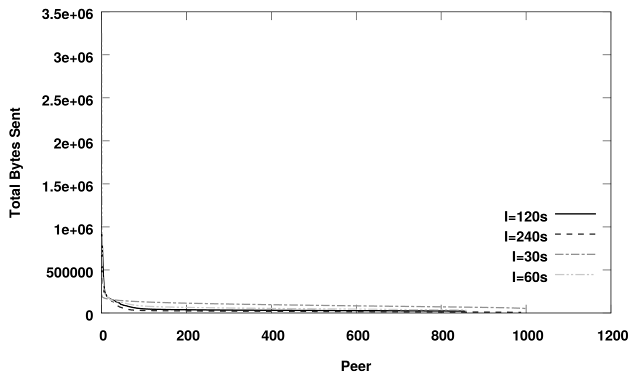
(b) B.1: Visited Nodes per Message.



(b) B.1: Visited Nodes per Message.



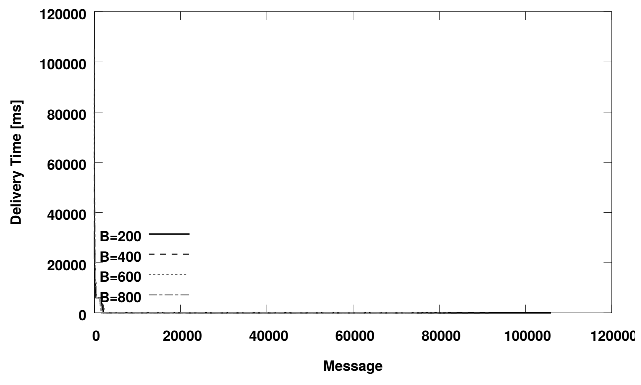
(c) B.1: Total Bytes Sent per Peer.



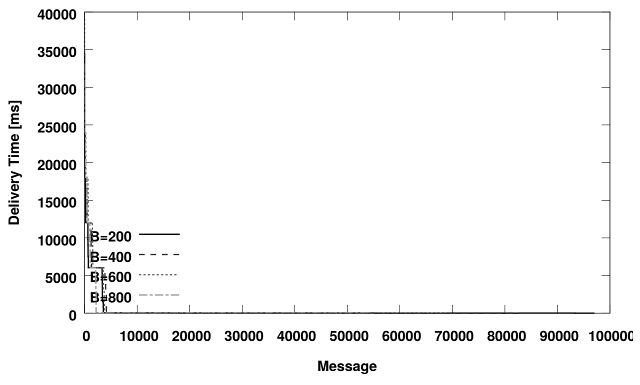
(c) B.1: Total Bytes Sent per Peer.

Fig. 46. B.1: Unmodified Chord.

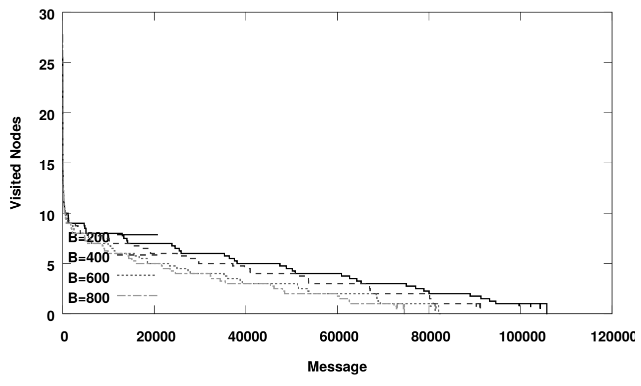
Fig. 47. B.1: Chord in Overlay Stack.



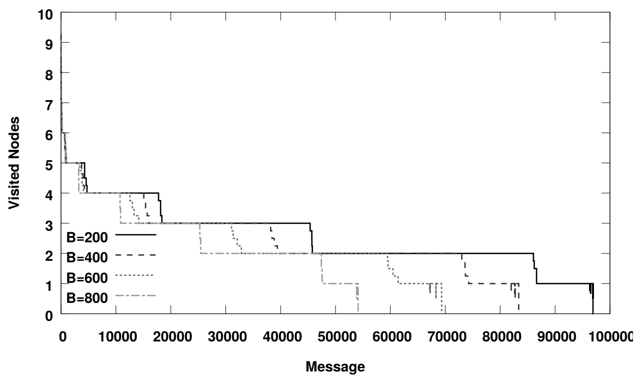
(a) C.1: Delivery Time per Message.



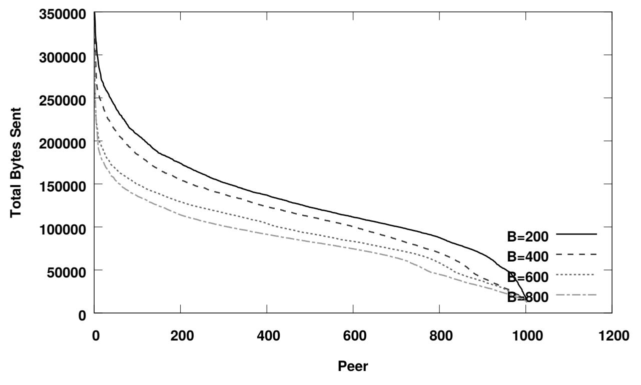
(a) C.1: Delivery Time per Message.



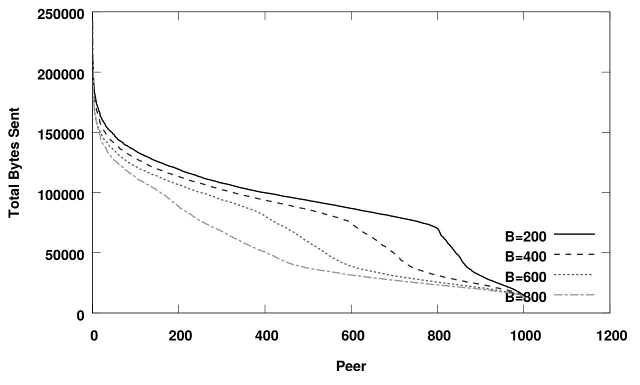
(b) C.1: Visited Nodes per Message.



(b) C.1: Visited Nodes per Message.



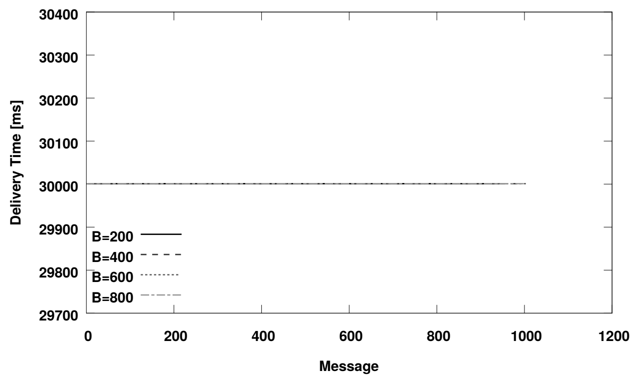
(c) C.1: Total Bytes Sent per Peer.



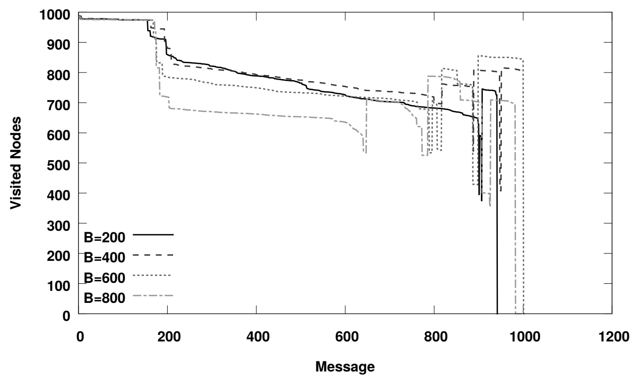
(c) C.1: Total Bytes Sent per Peer.

Fig. 48. C: Unmodified Chord.

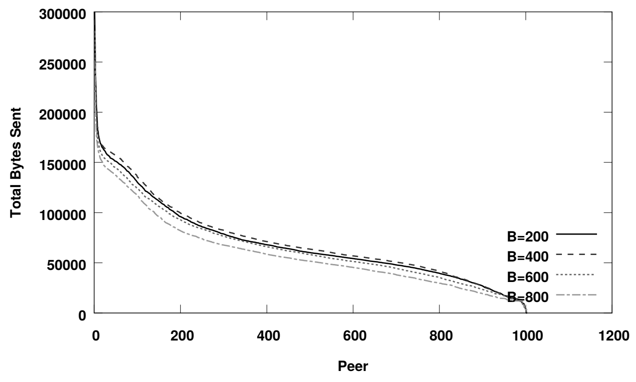
Fig. 49. C.1: Chord in Overlay Stack.



(a) C.2: Delivery Time per Message.

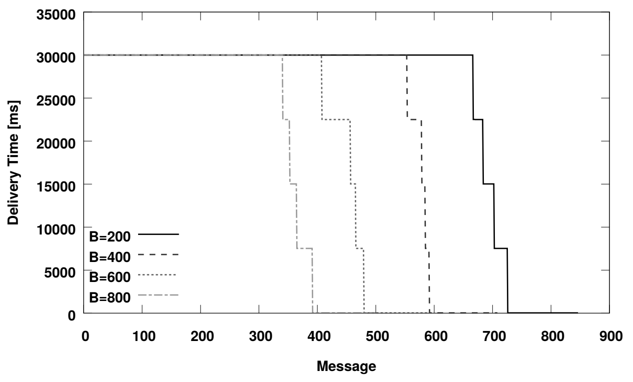


(b) C.2: Visited Nodes per Message.

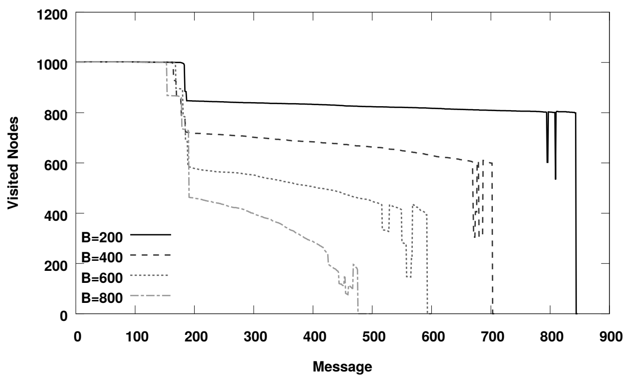


(c) C.2: Total Bytes Sent per Peer.

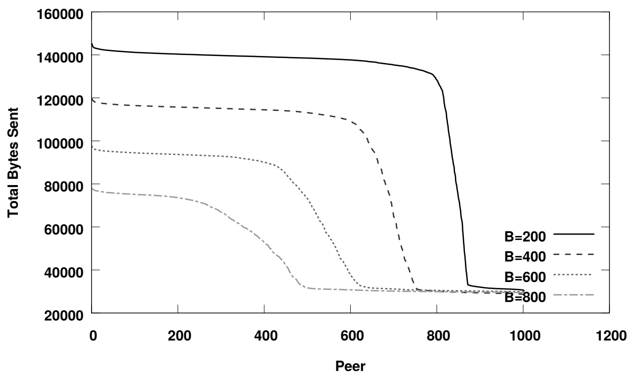
Fig. 50. C.2: Unmodified Flooding Overlay.



(a) C.2: Delivery Time per Message.

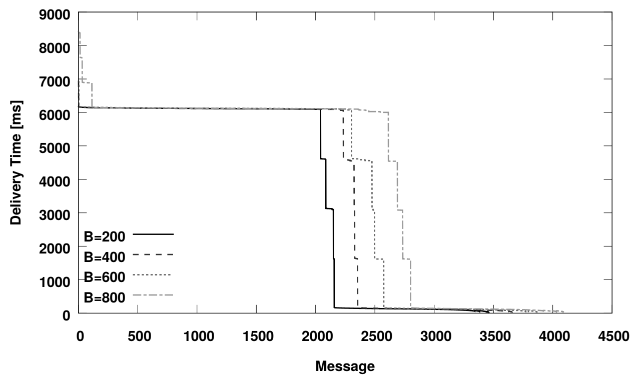


(b) C.2: Visited Nodes per Message.

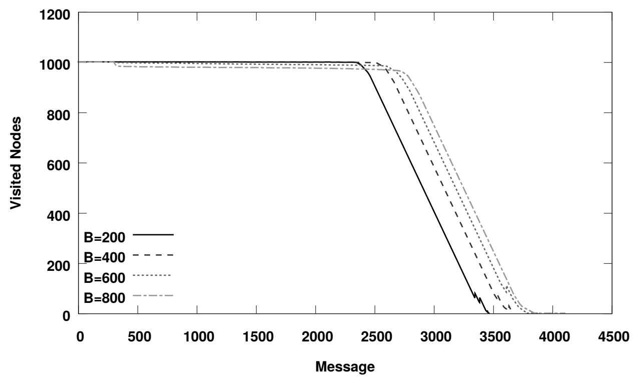


(c) C.2: Total Bytes Sent per Peer.

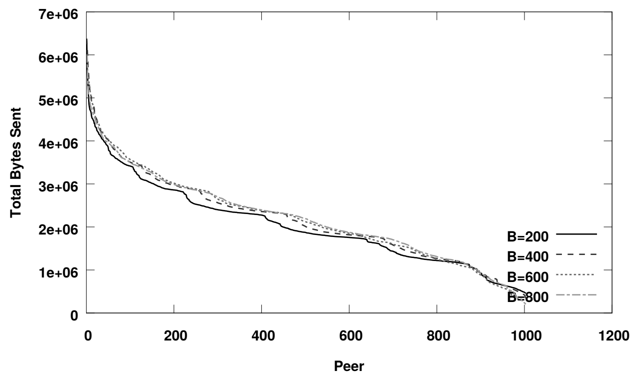
Fig. 51. C.2: Flooding Overlay in Overlay Stack.



(a) C.3: Delivery Time per Message.

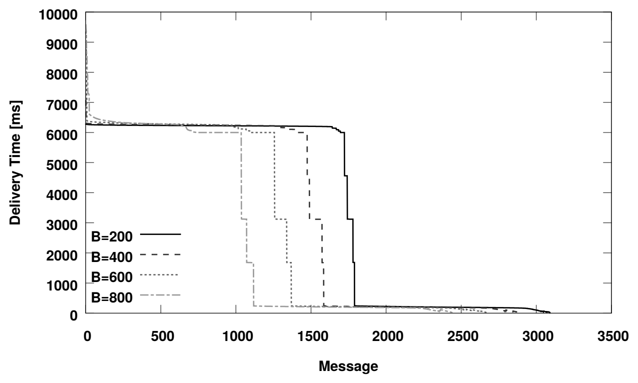


(b) C.3: Visited Nodes per Message.

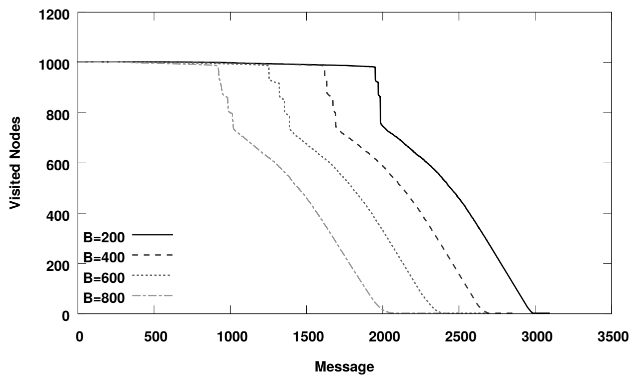


(c) C.3: Total Bytes Sent per Peer.

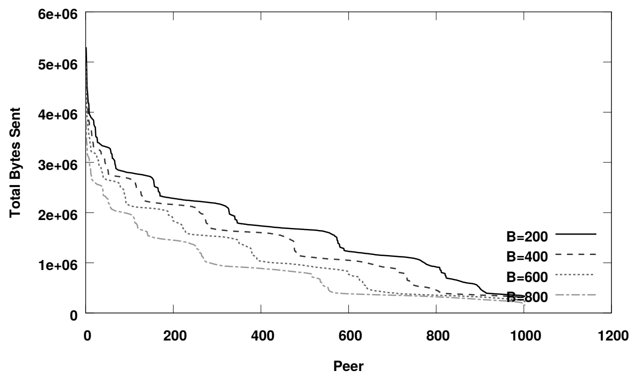
Fig. 52. C.3: Unmodified Geodemlia Overlay.



(a) C.3: Delivery Time per Message.

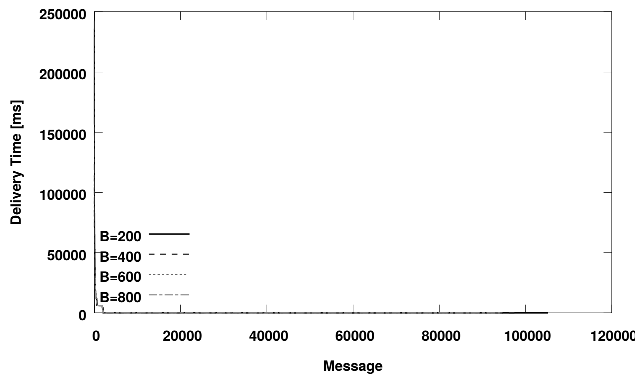


(b) C.3: Visited Nodes per Message.

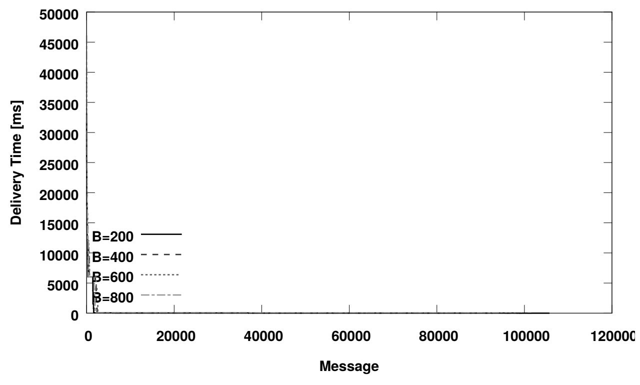


(c) C.3: Total Bytes Sent per Peer.

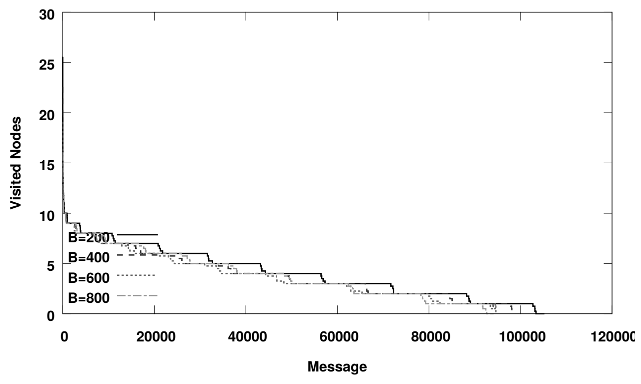
Fig. 53. C.3: Geodemlia Overlay in Overlay Stack.



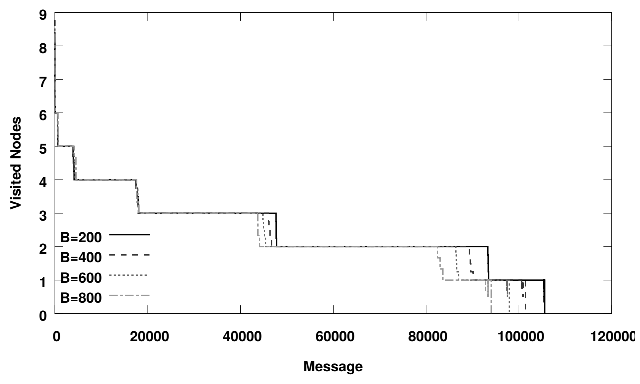
(a) D.1: Delivery Time per Message.



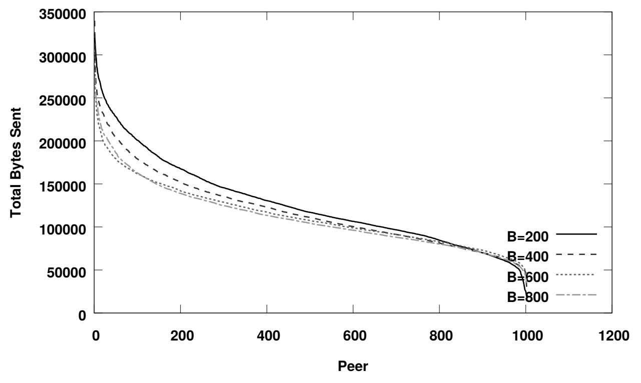
(a) D.1: Delivery Time per Message.



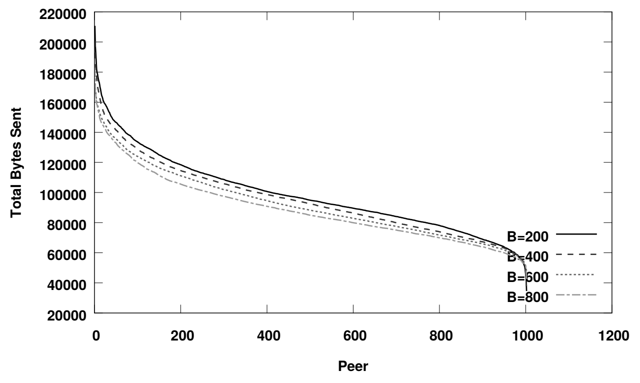
(b) D.1: Visited Nodes per Message.



(b) D.1: Visited Nodes per Message.



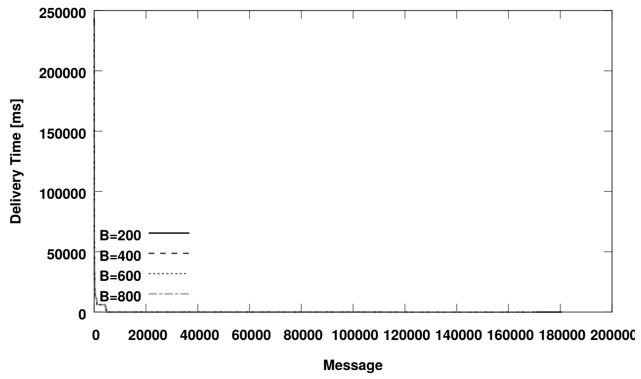
(c) D.1: Total Bytes Sent per Peer.



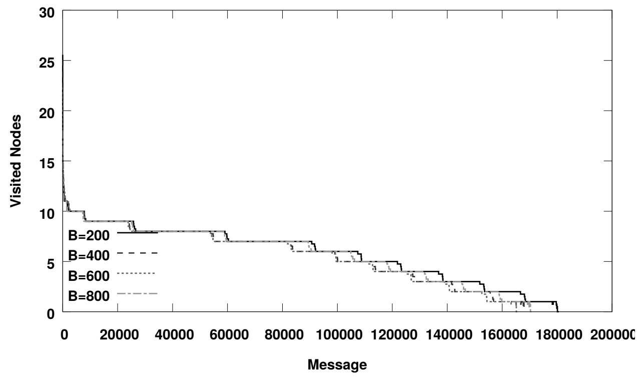
(c) D.1: Total Bytes Sent per Peer.

Fig. 54. D.1: Unmodified Chord without Ring Reunion Algorithm.

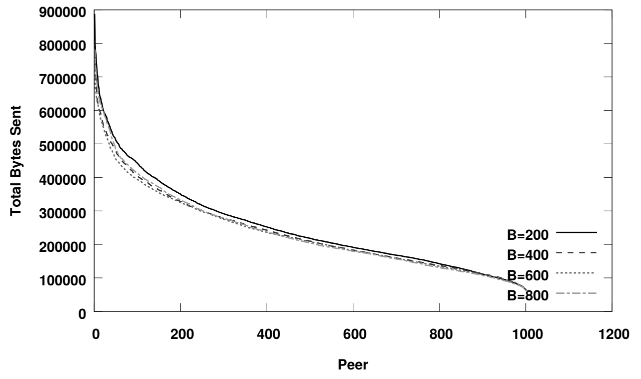
Fig. 55. D.1: Chord in Overlay Stack without Ring Reunion Algorithm.



(a) D.2: Delivery Time per Message.

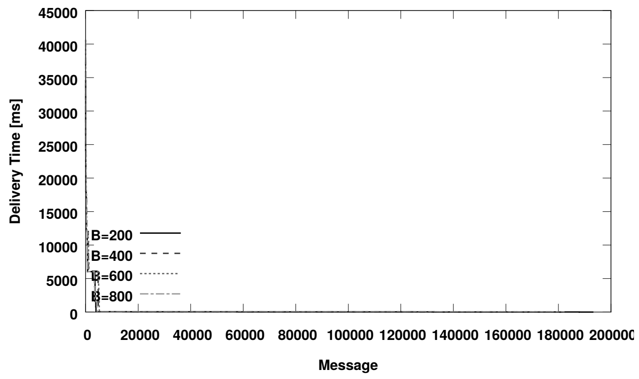


(b) D.2: Visited Nodes per Message.

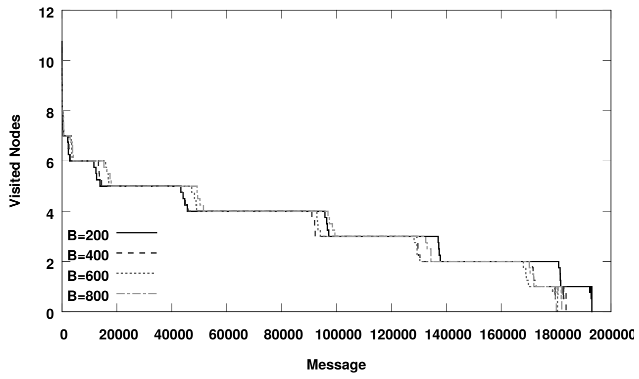


(c) D.2: Total Bytes Sent per Peer.

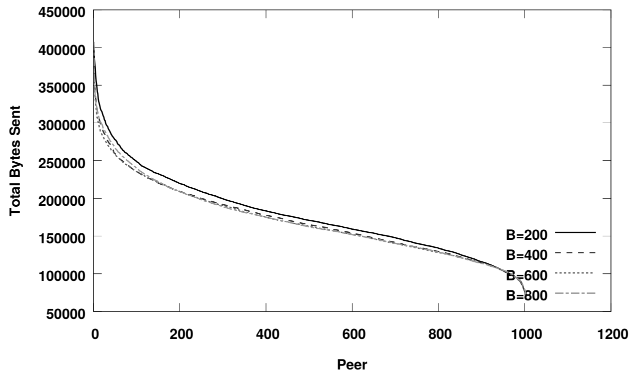
Fig. 56. D.2: Unmodified Chord with Ring Reunion Algorithm.



(a) D.2: Delivery Time per Message.

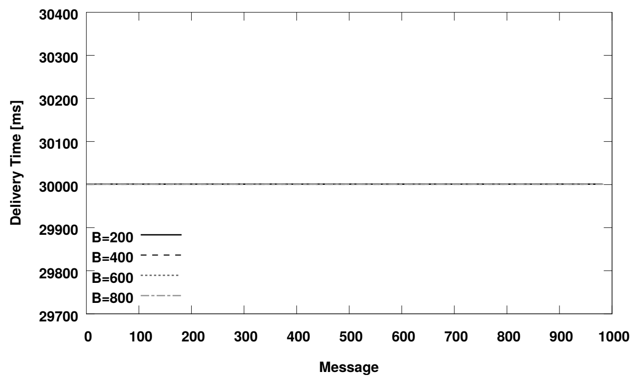


(b) D.2: Visited Nodes per Message.

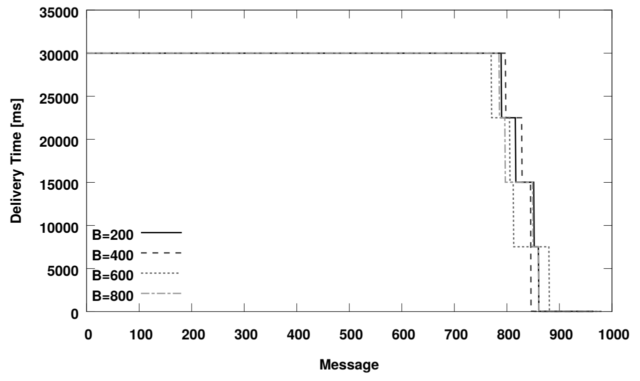


(c) D.2: Total Bytes Sent per Peer.

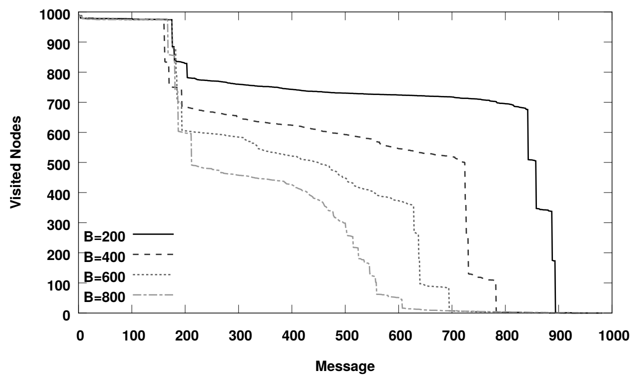
Fig. 57. D.2: Chord in Overlay Stack with Ring Reunion Algorithm.



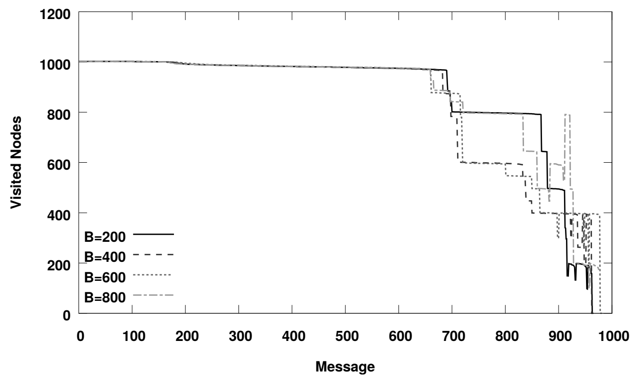
(a) D.3: Delivery Time per Message.



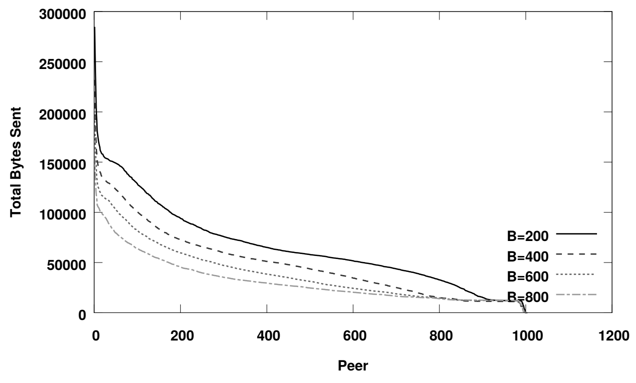
(a) D.3: Delivery Time per Message.



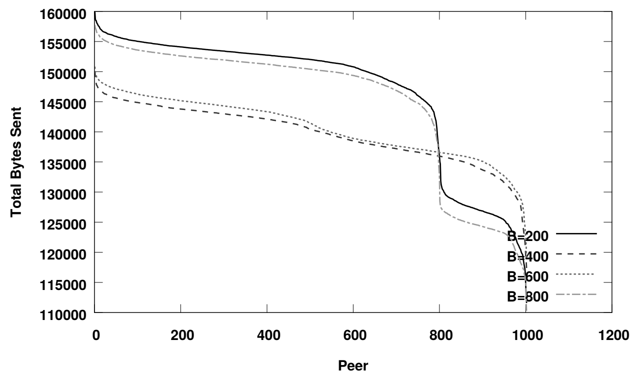
(b) D.3: Visited Nodes per Message.



(b) D.3: Visited Nodes per Message.



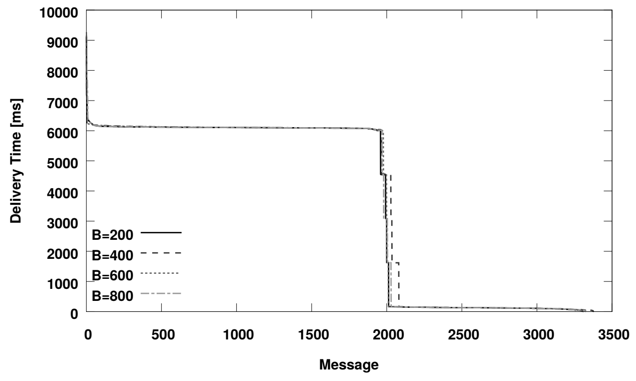
(c) D.3: Total Bytes Sent per Peer.



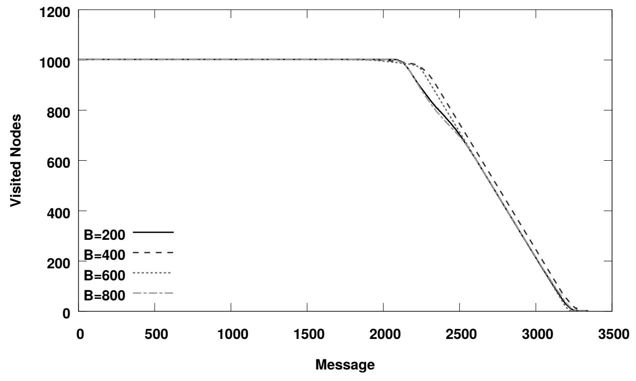
(c) D.3: Total Bytes Sent per Peer.

Fig. 58. D.3: Unmodified Flooding Overlay without Ring Reunion Algorithm.

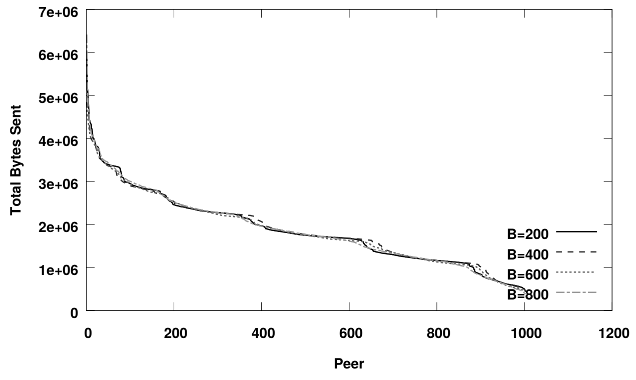
Fig. 59. D.3: Flooding Overlay in Overlay Stack with Ring Reunion Algorithm.



(a) D.4: Delivery Time per Message.

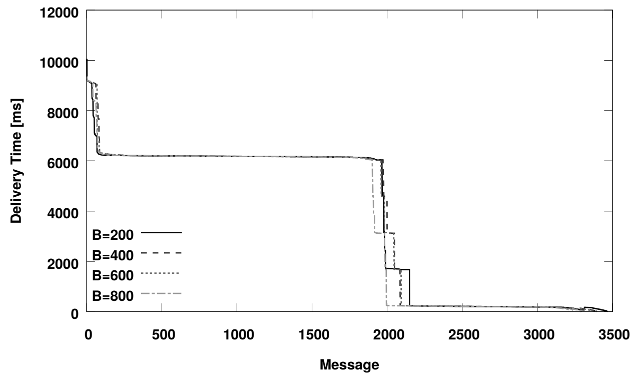


(b) D.4: Visited Nodes per Message.

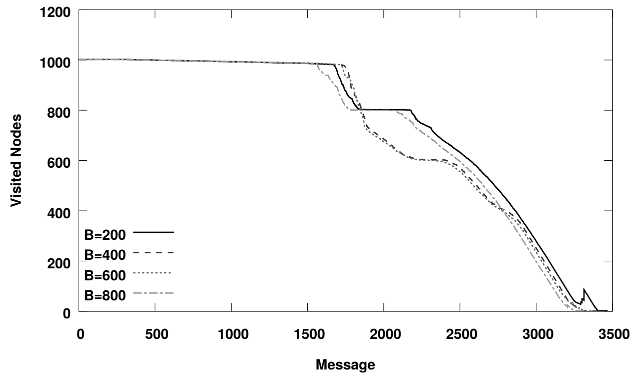


(c) D.4: Total Bytes Sent per Peer.

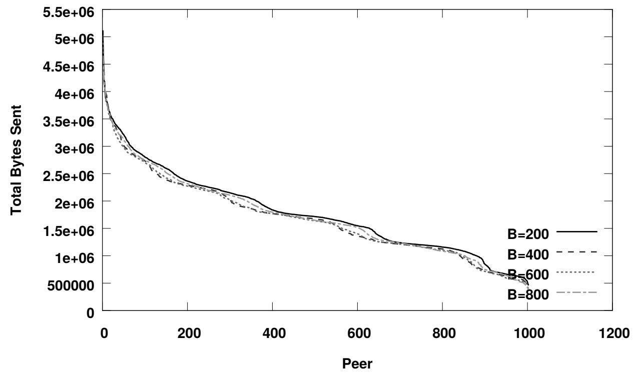
Fig. 60. D.4: Unmodified Geodemlia Overlay without Ring Reunion Algorithm.



(a) D.4: Delivery Time per Message.

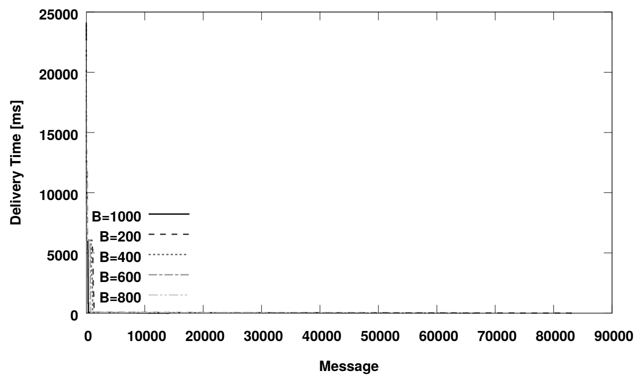


(b) D.4: Visited Nodes per Message.

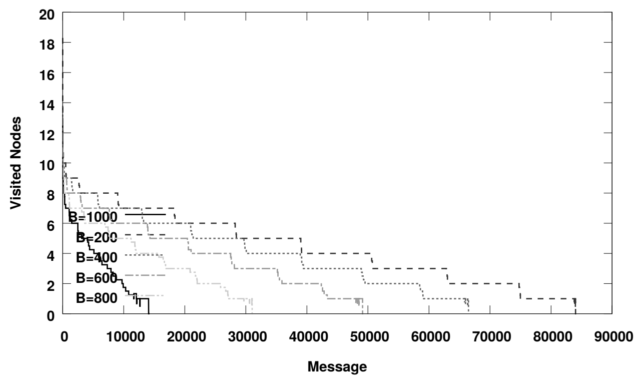


(c) D.4: Total Bytes Sent per Peer.

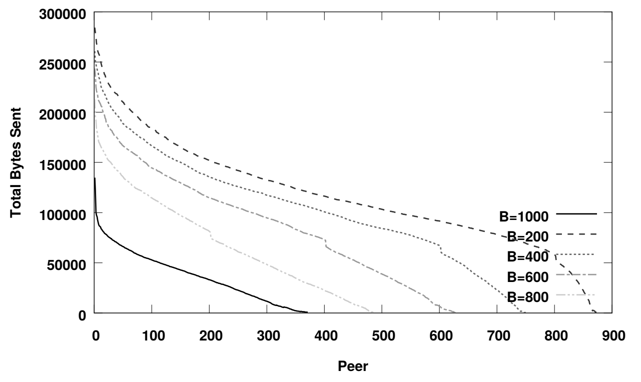
Fig. 61. D.4: Geodemlia Overlay in Overlay Stack with Ring Reunion Algorithm.



(a) E.1: Delivery Time per Message.

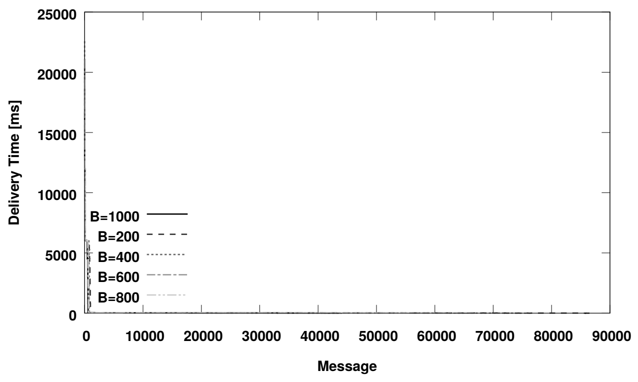


(b) E.1: Visited Nodes per Message.

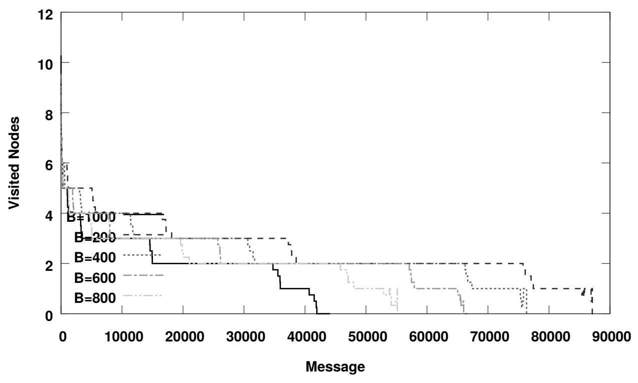


(c) E.1: Total Bytes Sent per Peer.

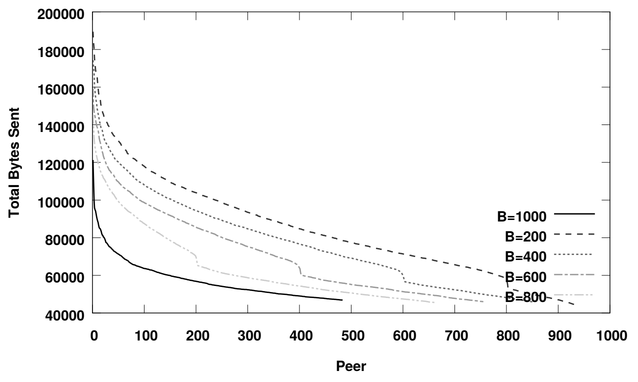
Fig. 62. E.1: Unmodified Chord.



(a) E.1: Delivery Time per Message.

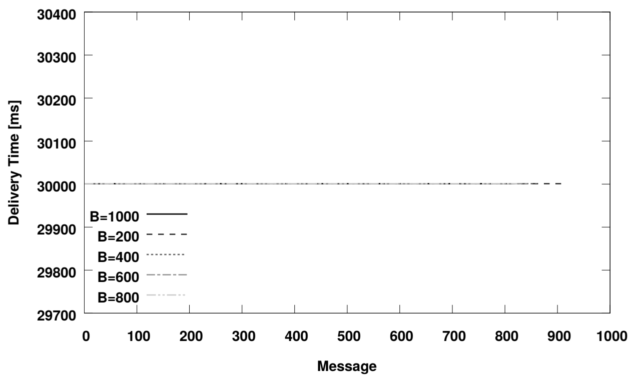


(b) E.1: Visited Nodes per Message.

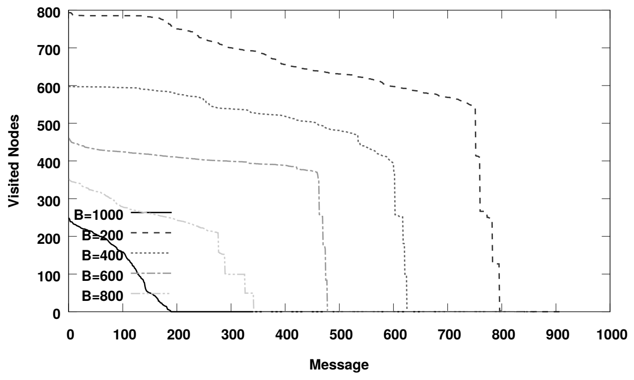


(c) E.1: Total Bytes Sent per Peer.

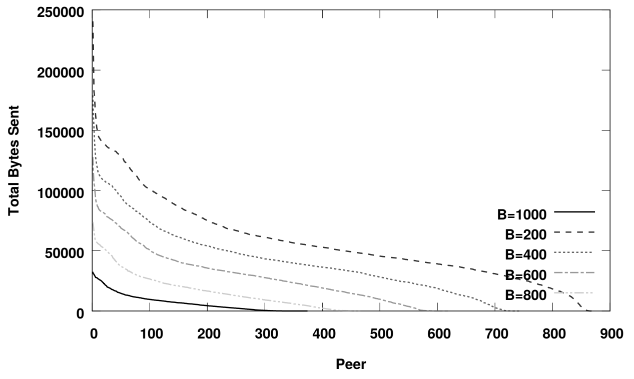
Fig. 63. E.1: Chord in Overlay Stack.



(a) E.2: Delivery Time per Message.

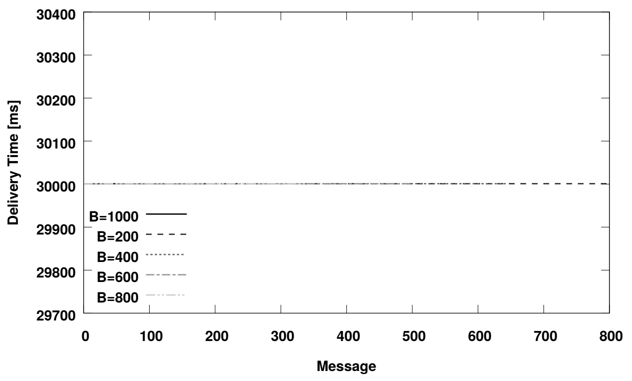


(b) E.2: Visited Nodes per Message.

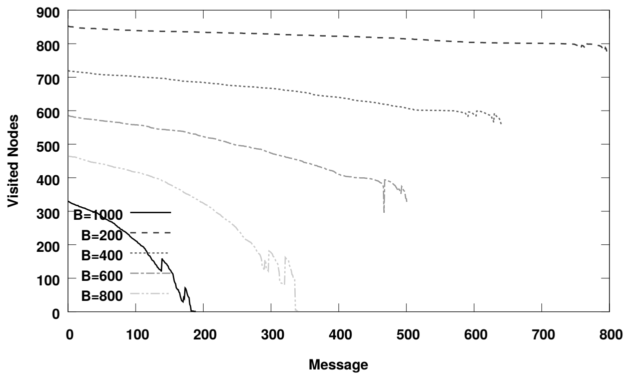


(c) E.2: Total Bytes Sent per Peer.

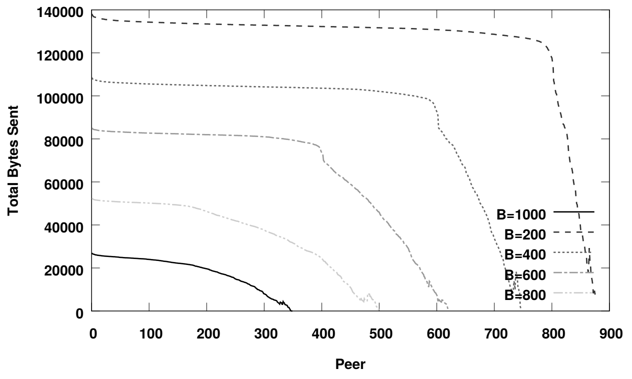
Fig. 64. E.2: Unmodified Flooding Overlay.



(a) E.2: Delivery Time per Message.

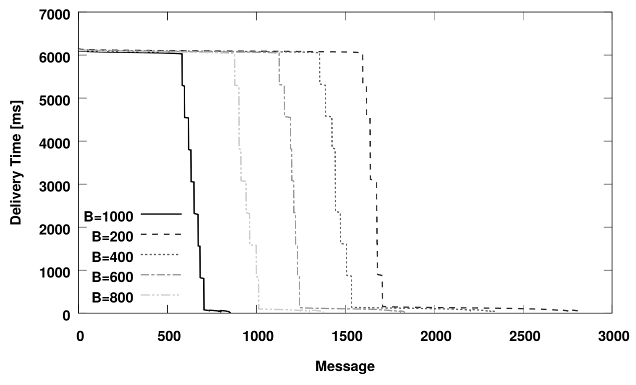


(b) E.2: Visited Nodes per Message.

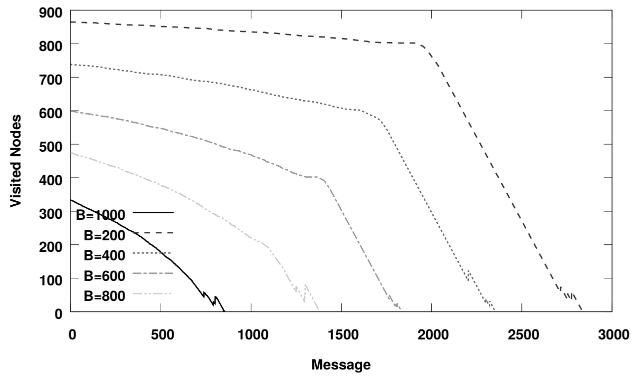


(c) E.2: Total Bytes Sent per Peer.

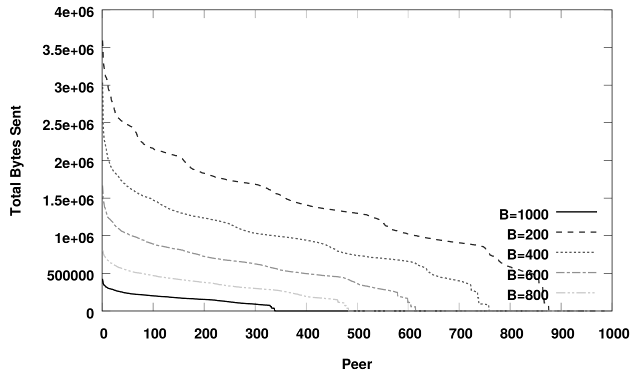
Fig. 65. E.2: Flooding Overlay in Overlay Stack.



(a) E.3: Delivery Time per Message.

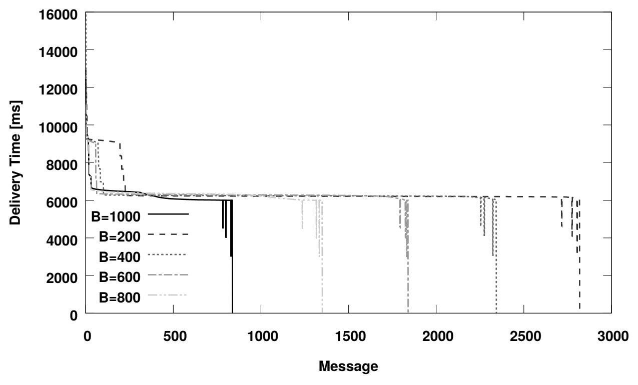


(b) E.3: Visited Nodes per Message.

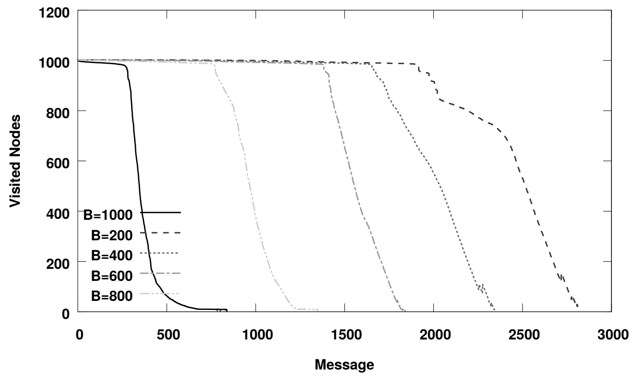


(c) E.3: Total Bytes Sent per Peer.

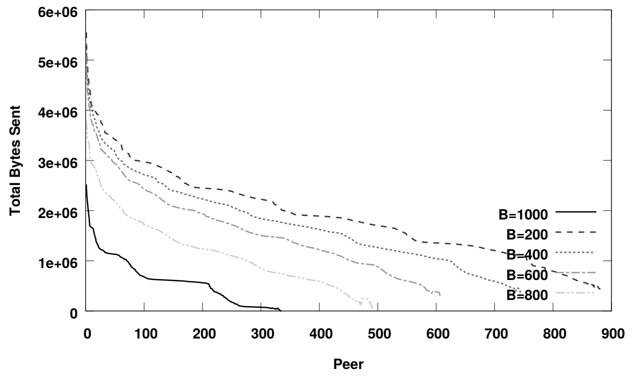
Fig. 66. E.3: Unmodified Geodemlia Overlay.



(a) E.3: Delivery Time per Message.



(b) E.3: Visited Nodes per Message.



(c) E.3: Total Bytes Sent per Peer.

Fig. 67. E.3: Geodemlia Overlay in Overlay Stack.