Parallelism Potentials in Distributed Simulations of Kademlia-Based Peer-to-Peer Networks

Philipp Andelfinger, Konrad Jünemann and Hannes Hartenstein
Steinbuch Centre for Computing and Institute of Telematics
Karlsruhe Institute of Technology
76131 Karlsruhe, Germany
{philipp.andelfinger, konrad.juenemann, hannes.hartenstein}@kit.edu

ABSTRACT

The benefits of distributing a network simulation depend on characteristics of the simulated network. Performance improvements reported in the literature are comparatively low for peer-to-peer overlay networks in particular, as the logical topology of these networks can necessitate frequent synchronization between the processors executing the simulation. In this paper, we show that a speedup of up to a factor of 6.0 using 16 nodes connected using InfiniBand and close to linear reductions in memory usage are possible for simulations of Kademlia-based networks. Our distributed simulator implementation enables simulations of one of the largest peer-to-peer networks at full scale of about 10 million peers. Based on the two fundamental goals of minimizing communication between processors and minimizing synchronization overheads, we propose two strategies for assigning simulated nodes to processors. We analyze the effects of the two strategies and show that each strategy supports one of the goals, while being detrimental to the other. We propose efficiency metrics that expose how much of the potential for parallel execution is exploited by a simulator. Through detailed performance measurements and by applying the new metrics to our simulator implementation, we quantify remaining efficiency potentials.

Categories and Subject Descriptors

I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Parallel, Distributed, Discrete Event*; E.2 [**Data Storage Representations**]: [Hash-Table Representations]

General Terms

Performance

Keywords

network simulation, distributed, kademlia, dht, lookahead

1. INTRODUCTION

Simulation is commonly used for performance analysis in the design and development of network protocols. For a realistic assessment, the protocol may need to be applied to simulations of networks at the scale of an envisioned or existing real-world deployment. However, the memory demands of simulations of millions of network nodes can easily exceed the resources available to individual processors. In addition, by increasing the number of nodes in a simulated network, the simulated traffic and hence the simulation runtime is usually increased as well. Parallel and distributed network simulation [13] is a common approach to reduce per-machine memory requirements and simulation runtime by partitioning the simulated network and distributing the simulation workload to a number of inter-connected processors. Each processor involved in executing the simulation is assigned a fraction of the simulated network nodes according to a partitioning scheme and simulates these nodes' behavior. The portion of the simulation handled by one processor is referred to as a *logical process* (LP). Communication between LPs is required to reflect the interactions between nodes in the simulated network. Additionally, to ensure that all activities in the simulated network are performed in correct order, synchronization is performed between LPs. Efficient distributed simulation requires sufficiently large *lookahead*, a metric determining the amount of simulated time by which an LP can advance through the simulation without synchronization. Both the required amount of communication between LPs and the synchronization overheads depend on characteristics of the network to be simulated.

Recently, there has been an interest in classifying networks according their potential for parallel and distributed simulation (e.g., [1, 31]), as for some types of networks, substantial runtime reductions through distributed simulation are reported in the literature [14, 34], while for other types of networks, the benefits are less pronounced [16, 33]. For peer-to-peer overlay networks in particular, existing work suggests that in many cases only limited runtime reductions are possible [10, 33]. Overlay networks impose a logical topology on top of a physical network, resulting in a distinction between physical and logical proximity of nodes. When simulating physical networks, it can be possible to exploit the spatial locality of the simulated traffic by assigning physically close nodes to the same LP. However, in overlay networks, traffic patterns follow the logical topology as opposed to the physical topology. Therefore, an assignment of physically close simulated nodes to individual processors does not necessarily result in low inter-processor interaction. If an efficient assignment based on logical proximity cannot be found instead, performance gains through distributed simulation remain low. Still, as peer-to-peer overlay networks

Simutools 2014, March 17-19, Lisbon, Portugal
Copyright © 2014 ICST 978-1-63190-007-5
DOI 10.4108/icst.simutools.2014.254609

in use today comprise millions of active nodes [12, 18, 35], efficient means for simulation of these networks are needed.

In this paper, we show how networks based on the Kademlia protocol can be simulated at the scale of real-world networks. Remaining efficiency potentials are investigated using novel metrics and through measurements of our high-performance simulator implementation. Our main contributions are the following.

1. Partitioning Schemes for Kademlia-Based Peer-to-Peer Networks: we show that, contrary to simulations of physical networks, the two main optimization goals in partitioning networks for distributed simulation, minimization of inter-LP traffic and maximization of lookahead, are not necessarily coupled for peer-to-peer overlay networks. For Kademlia-based networks, we show that a partitioning scheme based on the logical network topology can minimize inter-LP traffic, while a location-based partitioning scheme can increase the available lookahead.

2. Distributed Simulator and Performance Evaluation: through measurements using up to 16 LPs, we investigate the efficiency of our distributed simulator implementation for Kademlia-based peer-to-peer networks. Our implementation is based on PeerSim[1], a peer-to-peer network simulator that in its sequential implementation scales to up to 10^6 simulated nodes [29]. We show that the added capability for distributed simulation increases the simulator's scalability so that a large Kademlia-based network, the BitTorrent Mainline DHT [23], can be simulated at full scale [18] of around 10 million peers. We make the simulator code available to the community through our web site[2].

3. Synchronization Efficiency Metrics: we propose metrics to quantify the proportion of parallelism existing in a simulation model that is exploited by the synchronization mechanism. Measurements reveal the potentials for performance increases in our distributed simulations. We discuss how optimizations can exploit the identified potentials.

The remainder of the paper is structured as follows. In Section 2, we cover the related work fundamental to our paper and put our work in the context of previous approaches. In Section 3, based on the challenges in distributed simulation of peer-to-peer networks, we propose two partitioning schemes and study their efficiency benefits. In Section 4, we introduce two novel metrics to investigate the performance of the synchronization method used and discuss how optimizations can be applied to improve the efficiency. In Section 5, we validate our simulator implementation with reference to its sequential counterpart and through measurements investigate the performance benefits and remaining efficiency potentials when distributing the simulation. Section 6 concludes the paper.

2. BACKGROUND AND RELATED WORK

In this section, we describe the existing concepts our work is based on: first, we describe the Kademlia protocol [26] and the protocol properties that determine its potential for distributed simulation. Second, we describe the synchronization mechanism used in our simulator implementation. Third, we discuss existing literature on improving and measuring the efficiency of synchronization. Finally, we discuss the results presented in existing works on parallel and distributed simulation of peer-to-peer networks.

2.1 Kademlia

Kademlia [26] is a distributed hash table (DHT) enabling storage and retrieval of key-value pairs in a peer-to-peer network. Peers as well as keys are identified by numbers taken from a 160-bit ID space. The logical distance between IDs is defined by the XOR metric $d(x, y) = x \oplus y$. All interactions between peers are performed using remote procedure calls (RPCs), each of which is comprised of a request and a subsequent response. RPCs form the basis for *lookups*. We focus on *FIND_NODE* lookups that are initiated by sending messages to a number of peers, requesting the closest peers to a target ID. Additional requests are sent to the peers received in the incoming responses to iteratively retrieve the peers in the network that are closest to the target ID.

Each peer in the DHT maintains a routing table containing other peers in the network. The routing table is a binary tree of *k-buckets*, each holding at maximum k, usually 8, peers. Each k-bucket holds peers in a subsegment of the 160-bit ID space so that the set of all k-buckets in a peer's routing table covers the full ID space without overlap. When a peer A becomes aware of another peer B in the network, an attempt is made to insert B in the k-bucket covering the ID range corresponding with B's ID. If the k-bucket holds less than k peers, B is added to the k-bucket. If the k-bucket is full of alive peers, one of two steps is performed:

1. If the k-bucket covers the ID of peer A itself, the corresponding k-bucket is split in two, each new k-bucket handling half of the original k-bucket's ID range. Peer B is added to the new bucket corresponding to its ID.

2. If the k-bucket does *not* cover the ID of peer A itself, peer B is discarded.

Due to the splitting mechanism, a peer's routing table tends to contain more peers close to its own ID than peers with large XOR-distance.

Our implementation of the Kademlia protocol is modeled after the Mainline BitTorrent DHT as specified in [23]. The sources for traffic induced by the protocol are as follows.

1. Bootstrapping: when entering the DHT, each peer is bootstrapped by executing a lookup targeting its own ID to populate its routing table.

2. Routing Table Maintenance: if a peer attempts to add a new peer to a k-bucket that is full, requests are sent to peers in the k-bucket that have not sent a message in the past 15 minutes. If one of the probed peers does not respond within a timeout interval, it is replaced by the new peer. Additionally, if the contents of a k-bucket do not change within 15 minutes, the k-bucket is refreshed by performing a lookup for a random ID in the k-bucket's range.

3. User-initiated lookups: when a user requests the value associated with a given key, a lookup is triggered.

As each peer's routing table is biased towards peers with IDs close to its own, bootstrapping and routing table maintenance induce traffic concentrated in XOR-proximity of the peer. The traffic resulting from user-initiated lookups converges against the lookup's target ID, which in the simulation is drawn from a uniform distribution on the ID space.

2.2 Conservative Synchronization

In distributed simulations, synchronization between LPs is required to ensure that simulation results correspond to those produced by a sequential simulation. We focus on *discrete-event* simulations, where state changes in the simulated system are modeled as events occurring at discrete

[1] `http://peersim.sourceforge.net/`

[2] `http://dsn.tm.kit.edu/`

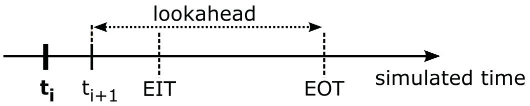

Figure 1: An LP determines its EOT by considering the timestamps of the earliest possible incoming remote event and of the next locally scheduled event. If the earliest of these events will trigger the creation of a remote event, the lowest possible timestamp of the new event is $min(EIT, t_{i+1}) + lookahead = EOT$.

points in simulated time. There are two main classes of synchronization algorithms for distributed simulations: conservative algorithms guarantee correctness of the simulation at all times by deciding in advance whether events are safe to be executed. An event is *safe* iff no event with lower timestamp can be received from any remote LP. If LPs only execute safe events, all events are executed in timestamp order by each LP. In contrast, optimistic algorithms do not ensure correctness in advance and, when detecting an incorrect event execution order, restore a previous correct simulation state. In our implementation, we apply conservative synchronization based on the common Chandy-Misra-Bryant (CMB) algorithm [6, 7]. Safe events are determined according to the metrics used by Bagrodia et al. [3]:

- *Lookahead*: the lowest possible delta between the timestamp of the event to be executed next by an LP and the timestamp of any locally created event to be executed on a remote LP. In network simulations, a fixed lookahead value is given by the minimum link latency between peers in the simulated network. Dynamic lookahead calculation will be discussed in Section 2.3.
- *Earliest Input Time (EIT)*: for each LP, the EIT is the earliest possible timestamp of an event that can be received from any remote LP.
- *Earliest Output Time (EOT)*: for each LP, the EOT is the earliest possible timestamp of the next locally created event to be executed on a remote LP. In the CMB algorithm, *null messages* containing the EOT are exchanged between LPs. If t_{i+1} is the timestamp of the next locally scheduled event, the EOT can be calculated as $EOT = min(EIT, t_{i+1}) + lookahead$ (cf. Figure 1). The EIT can be determined from the minimum of all other LPs' EOT.

Synchronization is achieved by the following steps.

1. When checking whether the next event in the local queue is safe to be executed, all LPs determine their local EOT and, if it has increased from the previous iteration, communicate it to all other LPs.
2. A local event is safe to execute if its timestamp is lower than the EIT, i.e., no event that should have been executed prior to the local event can be received from a remote LP. If no safe events exist, the LP is idle.

LPs alternate between two states: execution of safe events and waiting for events to become safe. In our implementation, every time an LP has finished executing all available safe events and there is a change in the local EOT, null messages are broadcast to all remote LPs. An alternative approach is to send null messages only on request by idle LPs [4]. However, as we will see in Section 5.2, null message traffic accounts for only a marginal amount of simulation runtime, while LPs are idle frequently. Hence, it is desirable to notify LPs about changes in EOT early.

2.3 Lookahead Extension

Several methods have been proposed in the literature to increase the lookahead extracted from a given simulation model, aiming to determine tight lower bounds on the timestamp of the earliest next event to be created for a remote LP. If model-specific knowledge is supplied to the distributed simulator, the lookahead can be calculated depending on the current state of the simulated system [8, 22, 27, 30, 36]. A more abstract view is given by control flow graphs modeling state transitions between the components of the simulation model [9, 37]. The lookahead can then be determined by following the path of the control flow through a control flow graph created statically or dynamically. Similarly, Meyer et al. propose a data flow view of the simulation [27] to dynamically calculate the lookahead.

The improvements achieved by lookahead extension approaches have in most cases been evaluated by studying the effects on simulation speedup, the number of null messages sent and the null message ratio, which is the proportion of null messages in relation to all messages exchanged between LPs. If there is insufficient lookahead, large numbers of null messages are required for the simulation to advance.

Some authors [28, 30] additionally evaluated their approaches using the Ideal Simulation Protocol (ISP) [17], which can predict the highest speedup achievable assuming optimal synchronization through distributed runs based on traces generated in previous simulation runs. Comparing the runtime of a distributed simulation under a proposed synchronization method with the runtime using ISP gives a clear view on the overall efficiency of the synchronization mechanism. To inspect in detail how much of the available lookahead is extracted from a model and to guide lookahead optimizations, efficiency metrics such as the lookahead ratio introduced by Fujimoto [15] and its derivate proposed by Preiss et al. [32], the null message inverse lookahead ratio, are required. Both metrics are calculated from the perspective of the LP *sending* future events. The metrics proposed in our paper, in addition to considering the lookahead calculation itself, take the perspective of the *receiver* of future messages and are sampled over wall-clock time. The receiver perspective allows us to determine how efficiently the lookahead is communicated among the LPs, i.e., to what extent the parallelism extracted from a simulation model by a given LP can actually be utilized by the other LPs.

2.4 Parallel and Distributed Simulation of Peer-to-Peer Networks

A previous effort of extending PeerSim for distributed simulation was presented by Dinh et al. [10] in 2008 for networks based on the Chord protocol. As in our work, synchronization is achieved using the CMB algorithm. While memory usage per LP is reduced substantially, the authors report simulation slowdown factors of 83 and more compared with a sequential implementation. In the same year, the authors presented performance measurements of the same simulator for the Chord and Pastry networks [11], reporting a superlinear speedup factor of more than 100 using 64 LPs. A partial explanation for the large speedup can be gathered from the enormous computational load incurred by their network model: a runtime of over two weeks is reported for a *sequential* simulation of a static network of 524,288 peers generating a fixed amount of traffic. While the performance of simulations of Chord and Kademlia cannot be

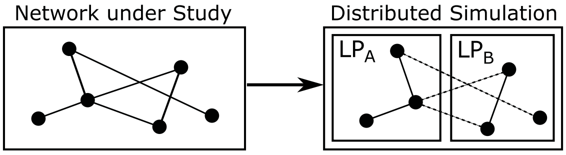

Figure 2: Assignment of peers of a studied network to LPs for distributed simulation. Edges denote communication between peers. In the distributed simulation, communication between peers simulated in different LPs (dashes lines) requires the exchange of physical messages between LPs.

compared directly, an indication of the computational intensity of their model is given by the runtime of 399s for identical parameters in our own sequential implementation. If the computational load of a simulation is very large, overheads for communication and synchronization incurred by distributing the simulation have only marginal impact, even though the absolute runtime remains very high.

Lin et al. presented a simulator engine for peer-to-peer networks that uses a synchronous master-worker synchronization scheme [20]. As limited scaling was observed under strict synchronization, the authors relax the synchronization requirements and ensure that simulated results are not affected substantially by determining bounds within which event timestamps may be altered during simulation. Simulations of networks based on the XRing protocol of up to 16,384 peers and speedup factors of up to 5.4 using 32 workers are reported by the authors. A similar architecture was proposed by Quinson et al. for simulations of the Chord protocol, achieving a speedup factor of up to about 1.45 using 24 threads [33]. The authors identify the low computational granularity and the difficulty of partitioning networks exhibiting the small-world property, i.e., low hop counts separating peers, as particular challenges in distributed simulation of peer-to-peer networks. Arguing that the resulting overheads cannot be amortized using traditional synchronization approaches, the authors propose a synchronous master-worker architecture for multicore systems. In contrast to this argument, for Kademlia-based peer-to-peer networks, our results show that distributing the simulation under the traditional CMB algorithm can substantially reduce simulation time. Furthermore, we point out efficiency potentials that can be exploited with future optimizations.

3. PARTITIONING SCHEMES FOR KADEMLIA-BASED NETWORKS

To distribute a simulated network of peers to a number of LPs, a decision must be made on the partitioning scheme, i.e., the strategy used in the assignment of simulated peers to LPs. Two sources of overheads must be considered that can critically affect the simulation performance: first, *physical exchange of messages between LPs*. Second, *waiting times required for synchronization.*

Simulation of the communication between peers assigned to different LPs requires a physical exchange of messages between the LPs (cf. Figure 2). Therefore, an efficient partitioning scheme aims to minimize the communication between peers simulated in different LPs. Similarly, the choice of partitioning scheme can also affect the available lookahead: if peers are assigned to LPs so that simulated message exchanges across LP boundaries are associated with high latency, a large lookahead value can be used.

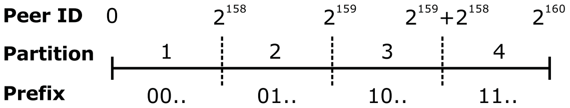

Figure 3: Example for ID-based partitioning of a simulated network into 4 partitions. Each partition contains peers with IDs sharing a common prefix.

In the following, from the perspective of a given peer, we refer to peers simulated on different LPs as *remote peers* and events targeting remote peers as *remote events.* Accordingly, peers simulated on the same LP are termed *local peers* and events targeting local peers are termed *local events.*

When simulating physical networks, a suitable partitioning can usually be found on the basis of the physical proximity of the simulated peers by assigning spatially close peers to the same LP. If closely located simulated peers are connected through high-throughput links and interact frequently (e.g., in a LAN), while distant peers interact less frequently over a low-throughput connection (e.g., through a WAN), remote events are infrequent and the overhead for exchanging messages between LPs is low. In addition, as link latency tends to increase with spatial distance [2], in a simulation using a location-based partitioning scheme, the minimum link latency of simulated messages sent across LP boundaries tends to be larger than the minimum latency of messages simulated within an LP, allowing for a large fixed lookahead value. Therefore, for simulations of physical networks, location-based partitioning can jointly reduce remote events and synchronization overheads.

In contrast, peer-to-peer overlay networks superimpose an application-level logical topology onto the underlying physical network. Finding a suitable partitioning for simulations of overlay networks is complicated by the fact that the logical topology of the overlay network does not necessarily reflect the physical proximity relationships between peers. Hence, contrary to simulations of physical networks, there is a tradeoff between minimization of the number of remote events through a partitioning based on the logical topology of the network, and maximization of latencies, and hence lookahead, associated with remote events through location-based partitioning.

3.1 ID-Based Partitioning

First, we focus on reducing the physical exchange of messages between LPs. To this end, we need to be aware of the communication patterns arising from the topology of the simulated network. The traffic induced by two of the sources of traffic in the Kademlia network, bootstrapping and routing table maintenance, is concentrated around the initiating peer's ID (cf. Section 2.1). We can exploit the resulting locality by partitioning the ID-space into segments of equal size and assigning one partition to each LP (cf. Figure 3).

We show that ID-based partitioning results in low amounts of overhead for communication between LPs: *For each doubling of the number of LPs, only a maximum of k additional peers in a peer's routing table come to reside on a remote LP, where k is the maximum number of peer IDs each bucket in a peer's routing table can hold, usually 8.*

Each peer's routing table can be viewed as a binary tree [26] where leafs are k-buckets and edges are annotated with the ID prefix handled by the leaves of the corresponding subtree

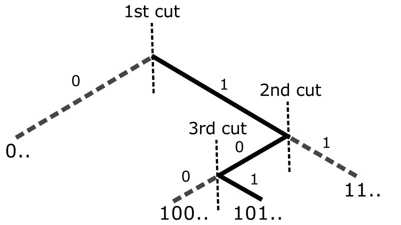

Figure 4: Binary tree structure of the Kademlia routing table of a peer with ID prefix 101. Dashed lines denote edges leading to leaf nodes not sharing the peer's prefix. Each doubling of LPs leads to a cut that displaces the peers in a single leaf node of the routing table to a remote LP.

(cf. Figure 4). The depth of the tree is only increased by applying the splitting mechanism described in Section 2.1. We use α to denote the ID of the peer owning the routing table. Consider the leaf node pertaining to α at depth i of the binary tree. The leaf node corresponds to a k-bucket holding peers with a common prefix of length i. The splitting mechanism replaces a leaf node containing α with a new subtree consisting of two edges: an edge e_s with a leaf node corresponding to IDs with a common prefix of length $i+1$ shared by α, and an edge e_n with a leaf node for a prefix of the same length *not* shared by α. In consequence, when following the edges pertaining to α's prefix, on level i of the tree, there is either a leaf node containing α, or there are two edges: one edge leading to an arbitrary number of nodes pertaining to IDs with prefix length i shared by α, and one edge leading to only a single node pertaining to IDs with prefix length i *not* shared by α.

Doubling the number of LPs from 2^i to 2^{i+1} mirrors the splitting mechanism and can be regarded as dividing two halves of the subtree at depth i between two LPs. For $2^0 = 1$ LP, the k-buckets pertaining to all nodes of the tree are handled by the local LP. When doubling the number of LPs, there are two cases: if the subtree at depth i is a leaf node, peers in one half of the corresponding k-bucket's ID range are assigned to a remote LP, while peers in the other half remain local. If the subtree at depth i has two edges, the peers of the single leaf node below e_n are assigned to a remote LP, while all other nodes in the subtree remain local. All nodes below e_s remain on the local LP. Hence, as each k-bucket holds a maximum of k peers, only a maximum of k peers become remote in each doubling of the LP count.

In Section 5.2, we show through measurements that the inter-LP communication increases only by a roughly constant amount when doubling the number of LPs.

Besides reducing inter-LP message exchanges, ID-based partitioning also enables reductions in the computational effort required within each LP. In simulations strictly adhering to the discrete-event paradigm, all state changes are modeled as events. However, by exploiting the fact that frequently, all system state pertaining to a single simulated communication is contained in a single LP, some events can be omitted in an effort to improve simulation performance.

Consider the following example: the simulated peer A is instructed to send a request to peer B, expecting a corresponding response. The reception of the request is delayed by a random link latency δ. However, peer B is offline (or behind a firewall). Therefore, after a fixed delay γ, a timeout will occur at peer A. After an initial *SendRequest* event has been scheduled, the process is modeled as follows.

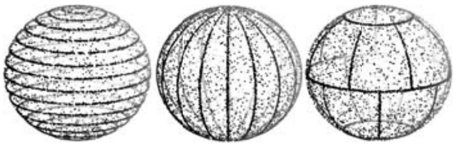

Figure 5: In the location-based partitioning scheme, peers are assigned to ranges of latitudes (left) or longitudes (middle), or to regions of small diameter and equal size (right) on the earth's surface.

1. t_0: Execute the *SendRequest* event at peer A.
2. t_0: Schedule a *ReceiveRequest* event for peer B.
3. $t_0 + \delta$: Execute the *ReceiveRequest* event. Peer B's offline status is detected.
4. $t_0 + \delta$: Schedule a *Timeout* event for peer A.
5. $t_0 + \gamma$: Execute the *Timeout* event.

If peer B's state is in local memory, we can optimize the process by probing the online status of peer B in step 1. Steps 2 and 3 can be avoided completely. However, peers A and B may be simulated by different LPs. In this case, peer B's status cannot be accessed directly, requiring the exchange of a minimum of two messages between the LPs simulating A and B. Nevertheless, for all peers simulated on a single LP, the optimization can be applied. As the only effect of steps 2 and 3 is the creation of the *Timeout* event, simulation results are not affected by this optimization. Hence, in addition to reducing the number of physical inter-LP message exchanges, some computational effort can be avoided by allocating communicating peers on the same LP.

3.2 Location-Based Partitioning

We will now focus on increasing the maximum lookahead value available in the simulation. A location-based partitioning can increase the average spatial distance between remote peers compared with local peers. As there is a strong relationship between physical distance and link latency [2], an increase in distance between communicating remote peers will be reflected by an increase in link latencies. Hence, with dynamic lookahead calculation, more local events will be safe to execute on average, potentially reducing idle times.

In our location-based partitioning scheme, peers are assigned to LPs according to the peers' spatial position. We compare three strategies: assignment based on ranges of latitudes or longitudes, and an assignment to regions with small diameter and equal area (cf. Figure 5). To find appropriate regions on the earth's surface, we used the MATLAB implementation of the algorithm proposed by Leopardi [19].

For a network model that follows the real-world distribution of peers across the earth's surface [18], the partitioning scheme would need to consider the given distribution both to achieve load balance between LPs and to maximize distances between remote peers. Here, we follow simplifying assumptions to be able to demonstrate the fundamental effects of the partitioning scheme. Based on the assumptions of a perfectly spherical earth and peers being distributed uniformly on the earth's surface we numerically examine the average spatial distance between peers exchanging messages across LPs. For each chosen number of partitions we calculate the average distance between points residing in different partitions, picked at random on the surface of a sphere.

Table 1: Average distance between remote peers under location-based partitioning.

	Average Distance [km]		
#Part.	By Latitude	By Longitude	By Regions
2	11895.2 ± 2.4	11895.7 ± 2.4	11895.4 ± 2.4
4	10685.5 ± 2.5	11263.4 ± 2.4	10954.7 ± 2.5
8	10341.4 ± 2.6	10653.3 ± 2.5	10808.4 ± 2.5
16	10166.9 ± 2.6	10325.1 ± 2.6	10465.0 ± 2.5
32	10084.2 ± 2.7	10158.8 ± 2.7	10254.6 ± 2.6
64	10040.3 ± 2.7	10078.1 ± 2.7	10132.5 ± 2.6

For picking points on the surface of a sphere, we use a method by Marsaglia [25]: we generate V_1 and V_2, both uniformly distributed on $(-1, 1)$ and reject all pairs where $S = V_1^2 + V_2^2 \geq 1$. Using the remaining pairs, the cartesian coordinates of points distributed uniformly on the unit sphere are given by $(2V_1\sqrt{1-S}, 2V_2\sqrt{1-S}, 1-2S)$. Given two such points, and after conversion to spherical coordinates ϕ_1, λ_1 and ϕ_2, λ_2, the distance on a sphere of radius r is given by $d = r\psi$, with $\psi = cos^{-1}(cos\phi_1 cos\phi_2 cos(\lambda_1 - \lambda_2) + sin\phi_1 sin\phi_2)$ (e.g., [5]).

The numerical results are listed in Table 1 with 95% confidence intervals. The average distance between points for a single partition is 10,000 km, corresponding to the expected distance between points on the surface of a sphere with 40,000 km circumference. The same mean distance is achieved by partitioning schemes not considering peer locations, regardless of the number of partitions. The largest benefit is achieved for two LPs: remote link latencies are increased by about 19%. When increasing the number of partitions, each partition becomes smaller and the results asymptotically approach those for a single partition. For large numbers of LPs, partitioning the earth into regions of small diameter gives the largest benefit of the three schemes.

In all cases, for 2 and more LPs, the minimum latency is given for communication between peers at a shared border of two partitions and hence remains constant. Therefore, the lookahead must be calculated dynamically to benefit from the location-based optimization scheme. The overall effect on simulation runtime depends on the communication costs between LPs: as peer IDs are chosen independently from locations, location-based partitioning does not follow the simulated network's logical topology. Consequentially, the number of messages exchanged between LPs must be expected to be substantially larger than with ID-based partitioning. Therefore, we will focus on the ID-based partitioning scheme in our performance evaluation.

4. PARALLELISM POTENTIALS

A given partitioning scheme exposes a certain proportion of the parallelism inherent in the simulated network. Ideally, LPs spend most of the distributed simulation runtime executing events instead of waiting for local events to become safe to execute. There are two possible reasons for frequent waiting: either there is insufficient parallelism in the simulated system itself, or the existing parallelism is not exploited to its full extent. In this section, we first introduce metrics to expose how much of the parallelism of a simulation model under a given partitioning scheme is exploited and apply the metrics to our distributed simulator. Second, we discuss how existing lookahead optimization techniques can be applied to distributed simulations of peer-to-peer networks in order to improve the efficiency of synchronization.

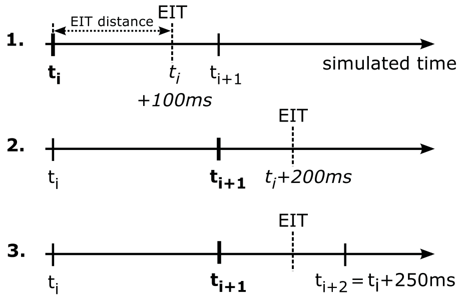

Figure 6: Chronological sequence of simulation schedules for LP_A as an example for waiting times due to synchronization. LP_A waits for its EIT to advance (1.) before executing further events (2.). At $t_i + 250ms$, a remote event arrives from LP_B (3.).

4.1 Assessment

Considering the timeline of a logical process LP_A depicted in Figure 6 in a distributed simulation using two LPs. The *EIT distance* is the delta between the current EIT and the LP's current point in simulated time. LP_A is currently idle, waiting for its EIT of $t_0 + 100ms$ to advance beyond any of the locally scheduled events so they become safe to execute (1.). Now, LP_B updates its EOT to $t_0 + 200ms$ and LP_A can start executing local events (2.). Finally, LP_A receives a remote event from LP_B with a timestamp of $t_0 + 250ms$ (3.). As we can see, at 1. it would have been possible for LP_A to start executing local events right away without violating timestamp order. Recall that the EOT is a lower bound on the timestamp of any event which may be created for a remote LP. An obvious question is then: *how tight is this lower bound?* We can consider the unnecessarily large waiting time of LP_A an effect of the insufficient *quality* of the EOT calculated by B. We introduce the term *EOT quality* and define it intuitively as follows: **the EOT quality is the average proportion of simulated time until an actual remote event is received that is covered by a previously received EOT**. An EOT quality of 100% corresponds to perfect synchronization between LPs, i.e., LPs are able to exactly predict the timestamp of the next incoming remote event and can execute all prior safe events immediately, whereas an EOT quality of 0% will not allow the simulation to progress at all. As we are interested in the average quality of the EOT over the course of a simulation run, we sample the EOT distance periodically during simulation runtime by storing remote EOT distances received in the most recent null messages. When the next remote event by each remote LP is received, the stored EOT distance is divided by the distance of the remote event's timestamp from the reference point in simulated time. In our example, the quality of EOT_B at 1. is $100ms/250ms = 40\%$. The EOT quality indicates how efficiently the lookahead available in the simulation model under a given partitioning scheme is determined *and communicated* to other LPs. A related metric is the lookahead ratio introduced by Fujimoto [15] and its derivative by Preiss et al. [32], the null message inverse lookahead ratio (NILAR). The lookahead ratio relates the average time increment between an LP's events to the lookahead, without considering null messages. The NILAR applies the same idea to the lookahead used in null messages. Both metrics are determined from the perspective of the LP *sending* events and null messages. However, our aim is to study the reasons for idle times in the distributed simulation. As an LP is idle whenever there are no safe events according to the EOTs received from other LPs, the EOT quality is calculated from the perspective of an LP *receiving* an EOT and sampled over wall-clock time. This way, in addition to considering the lookahead calculation itself, the EOT quality takes into account the efficiency of the null message sending strategy.

We will now give a more formal definition of the EOT and EIT quality metrics. Given the current wall-clock time τ and the current position t_i in simulated time, we define the *EOT distance* as the delta between the last EOT received from the remote logical process LP_r and the current position in simulated time: $d_{EOT}(\tau, LP_r) = EOT(\tau, LP_r) - t_i(\tau)$. Sampling the EOT quality at τ in wall-clock time in a logical process LP_l is comprised of the following steps.

1. LP_l's current position in simulated time $t_i(\tau)$ is stored together with all current EOT distances $d_{EOT}(\tau, LP_r)$.

2. For *each* remote logical process LP_r, when the next remote event with timestamp t_{r,LP_r} is received, the corresponding EOT quality is calculated as

$$Q_{EOT} = \frac{d_{EOT}(\tau, LP_r)}{t_{r,LP_r} - t_i(\tau)}$$

In simulations with more than two LPs, another metric becomes useful: EIT quality is the proportion of simulated time until a remote event is received that is covered by a previous EIT. The EIT quality shows the effect of aggregating the remote LPs' EOTs. Sampling is performed as follows.

1. LP_l's current position in simulated time $t_i(\tau)$ is stored together with the current EIT distance $d_{EIT}(\tau) = EIT(\tau) - t_i$.

2. When the next remote event with timestamp t_r is received from *any* of the remote LPs, the EIT quality is calculated as

$$Q_{EIT} = \frac{d_{EIT}(\tau)}{t_r - t_i(\tau)}$$

We will study the EOT and EIT quality measured in runs of our distributed simulator implementation in Section 5.2.

4.2 Lookahead Extension Options

Given metrics for the quality of the calculated EOT and EIT, the question arises: *how are these measures affected when applying optimizations to the distributed simulator?*

In the EOT and EIT quality measurements, we have seen that only a small portion of the available parallelism is exploited given a fixed lookahead. To achieve more efficient synchronization, the lookahead can be calculated by evaluating each LP's system state dynamically to determine a lower bound on the timestamp of the next remote event that may be created locally. Previously, we determined the EOT using the timestamp of the next locally scheduled event and a fixed lookahead: $EOT = min(EIT, t_{i+1}) + lookahead$. In this section, we focus on events created locally and hence consider the case where $EIT > t_{i+1}$, and consequentially $EOT = t_{i+1} + lookahead$. The EOT calculation using fixed lookahead makes three worst case assumptions, each limiting the EOT and EIT quality: *First*, we assume that the next remote event will have the minimum possible link latency. In non-pathological examples, a substantial proportion of link latencies will be larger than the minimum. *Second*, it is assumed that the next local event will in fact create a remote event.*Third*, we calculate the EOT under the assumption that any new event can affect any of the remote LPs. However, ultimately, each event created will affect at most one of the remote LPs. How can we relax these worst case assumptions and obtain tighter bounds on the distance of the next remote event based on the existing methods discussed in Section 2.3? To address the first assumption, we revisit the example of Section 4.1, now focusing on LP_B (cf. Figure 7). At t_j in simulated time, we are aware of the timestamp t_{j+1} of the next local event. If the next event will indeed create a remote event, we are able to predict the new event's timestamp exactly given knowledge of the next random latency. An optimization proposed in the literature [3, 21, 24] is based on the fact that random number generation is a deterministic process. Instead of generating the next random latency at the time of creation of each event, we can precalculate link latencies, allowing us to exactly determine the timestamp of a remote event potentially created next.

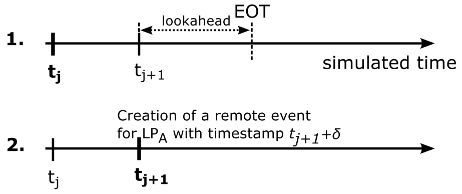

Figure 7: LP_B's schedule during calculation of its EOT based on fixed lookahead (1.). The event at t_{j+1} creates a remote event to be executed by LP_A after a delay based on a random link latency δ (2.).

To address the second assumption, we can examine the type of the next local event. If events of the given type cannot create new events, or can only create local events, events of the given type can be skipped in our EOT calculation. However, if the event is of a type that can create remote events, the peer state must be examined to make guarantees with respect to remote event creation. For instance, if a peer at a given point in simulated time is not aware of any peer on a remote LP, an event corresponding to this peer will not affect any remote peers. Gathering this knowledge requires examining the peer's routing table. The same way, we can address the third assumption by determining from a peer's state which of the remote LPs can be affected by newly created events. A dedicated EOT value can then be calculated for each remote LP.

Being able to skip events with only local effects in the EOT calculation has the notable side-effect of creating a link between the two optimization goals identified in Section 3: if a partitioning scheme minimizes the number of remote events, more events can be skipped during EOT calculation and synchronization overheads are reduced as well.

However, relaxing the worst-case assumptions during EOT calculation in an LP incurs additional computational costs. In Section 5.2, we will see that in our measurements, a substantial portion of simulation time is spent waiting for local events to become safe to execute and could be invested in dynamic lookahead calculation instead. Hence, we are hopeful that the parallelism unlocked by the additional computations can result in a reduction in overall simulation runtime.

In our future work, we will investigate the tradeoff between the computational effort required to increase EOT quality and the gains in overall simulation performance.

5. SIMULATOR EVALUATION

In this section, we evaluate the effects of the partitioning schemes introduced in Section 3 through performance measurements of our distributed simulator implementation. We will first demonstrate that our distributed simulator implementation produces results that correspond to those of the sequential simulator. A statistical validation shows deviations of 1.0% or less in the results. The simulator performance is studied for simulations of networks of 1 and 10 million peers. Simulation runs were performed on up to 16 machines equipped with 16 Intel Xeon E5-2670 cores each and connected using InfiniBand 4x QDR. To be able to fully exploit each machine's memory resources, each LP uses all 16 cores of one machine. Each LP uses one core each for simulation and communication. The remaining cores are available to the Java runtime environment to perform garbage collection. The sequential simulator used as a reference for speedup calculation utilizes 16 cores in the same fashion. The sequential implementation is highly optimized and simulates a network of one million peers for one simulated hour in about 1.5h of wall-clock time. Results are stated as averages of three runs with 95% confidence intervals.

5.1 Validation

The correctness of a distributed simulator is determined with reference to its sequential counterpart. If both implementations produce identical results, the distributed simulator is considered to be correct. Here, we perform a statistical validation and show that distributing the simulation has only an insignificant effect on the results, as in, e.g., [14].

We compare the results of the sequential and distributed runs with respect to lookup performance metrics, as DHTs such as Kademlia are used for efficient storage and retrieval of key-value pairs. The correctness of the simulator hinges on correct modeling of these procedures. Table 2 lists the number of requests, lookup duration and number of timeouts for runs of one simulated hour with a network size of 1 million peers for simulations using up to 16 LPs. Three runs with different seeds were performed for each configuration. We can see that there is a close match in the results, the highest observed deviation from the sequential runs being 1.0% for the average lookup duration, 6552.00ms compared to 6484.67ms, for 16 LPs, which we argue considering the associated confidence intervals can be regarded as marginal in the context of real-world simulation studies.

5.2 Performance

We study the performance of the simulator for two different partitioning schemes. Our focus is the ID-based partitioning scheme that promises high performance by considering the simulated network's topology. We contrast the results with a random partitioning scheme that does not consider peer IDs and must therefore be expected to incur the same amount of overheads as the location-based partitioning scheme under fixed lookahead.

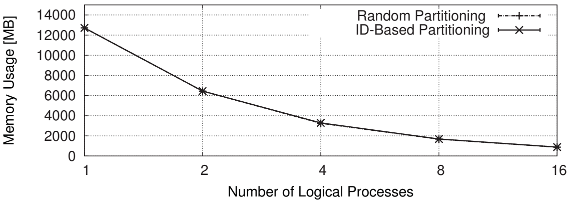

Figure 8: Memory usage per LP for a network size of 1 million peers, varying the number of logical processes and the partitioning scheme.

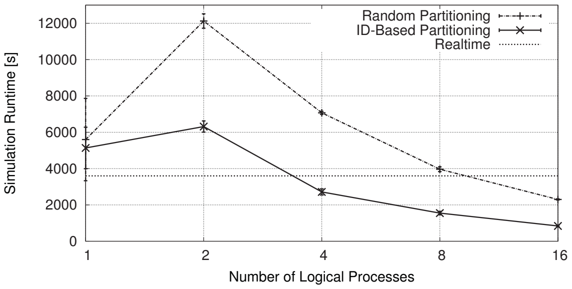

Figure 9: Simulation runtime for a network size of 1 million peers, varying the number of logical processes and the partitioning scheme.

Table 3: Percentage of messages to local peers [%] depending on partitioning schemes.

LPs	Random	ID-Based
1	100	100
2	45.67 ± 0.04	91.11 ± 0.01
4	22.24 ± 0.14	83.00 ± 0.03
8	11.11 ± 0.09	75.27 ± 0.03
16	5.82 ± 0.06	67.69 ± 0.06

Figure 8 shows the memory usage per LP for simulation runs using ID-based and random partitioning, for networks of 1 million peers over the course of one simulated hour. Memory usage was reduced close to linearly with the number of LPs, from 12713 MB to 883 MB when moving from 1 to 16 LPs, a factor of 14.4. The choice of partitioning scheme had no marked impact on memory usage.

Simulation runtime (cf. Figure 9) was reduced substantially as well. For two LPs, the overheads for synchronization and physical message exchanges were not amortized by the distributed computation. This is an effect of null messages being sent by each LP only after executing all available safe events, as LPs frequently have to wait for the next null message from their single counterpart before computation may proceed. Starting with 4 LPs, simulations under ID-based partitioning proceeded faster than realtime. Highest performance was achieved using 16 LPs with ID-based partitioning, reducing simulation runtime by a factor of 6.01 compared with sequential runs. The simulation runtime was 843s, compared with 2301s for random partitioning.

Table 2: Comparison of sequential and distributed simulation results.

	1 LP	2 LPs	4 LPs	8 LPs	16 LPs
Number of Requests (10^6)	694 ± 0.51	694 ± 1.63	692 ± 3.07	691 ± 3.34	691 ± 2.62
Avg. Lookup Time [ms]	6485.67 ± 19.30	6514.67 ± 23.87	6521.67 ± 43.62	6536.33 ± 30.36	6552.00 ± 37.51
Timeouts [%]	42.54 ± 0.08	42.54 ± 0.09	42.46 ± 0.18	42.42 ± 0.13	42.39 ± 0.16

Table 4: Percentage of time spent in the different execution states during simulation runtime.

	1 LP	2 LPs	4 LPs	8 LPs	16 LPs
Execute Event	98.19 ± 0.47	56.32 ± 4.65	55.82 ± 2.53	50.68 ± 2.58	47.38 ± 1.85
Forward Event	N/A	3.00 ± 0.66	6.13 ± 1.70	7.24 ± 1.64	8.32 ± 1.55
Handle Message	N/A	15.31 ± 1.17	22.28 ± 1.86	19.56 ± 2.72	17.53 ± 2.73
Send Null Message	N/A	0.01 ± 0.00	0.06 ± 0.01	0.22 ± 0.02	0.80 ± 0.08
Idle (incl. Overhead)	1.81 ± 0.47	25.37 ± 3.83	15.72 ± 6.03	22.30 ± 4.41	25.97 ± 5.14

Table 5: EOT and EIT quality varying the LP count.

LPs	Q_{EOT} [%]	Q_{EIT} [%]
2	32.30 ± 2.20	32.30 ± 2.20
4	39.43 ± 0.45	26.94 ± 4.61
8	31.33 ± 3.38	17.89 ± 1.79
16	23.96 ± 5.26	9.92 ± 0.56

In order to demonstrate the simulator's scalability, we performed an additional simulation run for a network with the size of the BitTorrent Mainline DHT [18] of 10 million peers over the course of one simulated hour using ID-based partitioning on 16 LPs. The simulator required 14966s (about 4.2h) to simulate a total of about 8.24×10^9 requests. Each of the 16 LPs used about 9450 MB of memory.

To explore the basis of the benefit of ID-based partitioning in the simulations for 1 million peers, Table 3 lists the percentage of simulated messages that were exchanged between local peers and thus did not require physical communication between LPs. For random partitioning, the percentage of local messages was roughly halved when doubling the number of LPs. For ID-based partitioning, the percentage of local messages was reduced by a roughly constant amount of about 8% for each doubling of LPs, supporting our analysis in Section 3.1. Location-based partitioning (cf. Section 3.2) does not consider peer IDs and must hence be expected to create as many remote events as random partitioning.

We studied the distributed simulation performance more closely by instrumenting the simulator to measure the proportion of runtime spent in the following simulation states: *Execute Event*: a safe event is being executed; *Forward Event*: an event is sent to a remote LP; *Handle Message*: an incoming message is parsed, and if the message contains a remote event, it is added to the local queue; *Send Null Message*: a null message is sent to a remote LP; *Idle*: the LP waits for local events to become safe to execute. The *Idle* state includes the overheads incurred by the time measurements. Table 4 lists the proportion of time after initialization that was spent in the different states for simulations with ID-based partitioning. We can see that with increasing numbers of LPs, the time spent executing events decreased. As expected, the time spent on exchanging events between LPs increased only moderately with larger LP count. Null message sending overhead increased super-linearly, yet only accounted for a small amount of simulation runtime. In all distributed cases, a large amount of time was spent in the idle state. For 1 LP, the idle state was comprised completely of time measurement overheads, which accounted for less than 2% of the simulation runtime, indicating that in the distributed runs, the time spent in the idle state was indeed dominated by waiting for local events to become safe.

To further investigate the cause for idle times in the simulation, we sampled the EOT and EIT quality every second as described in Section 4.1. The measurement results are listed in Table 5. On average, a maximum of 39.43% of the simulated time up to the next remote event was covered by a received EOT. As the EIT is calculated as the minimum of all received EOT, even less time was covered by the EIT, with a decrease in quality for larger numbers of LPs. For 16 LPs, the measured EIT covered only 9.92% of the simulated time until the next remote event was received. There are two possible causes for low EOT and EIT quality: either the LPs do not communicate their EOT frequently enough, or the lookahead calculation does not exploit the parallelism in the simulation model sufficiently. To determine which of the explanations applies, we maximized the null message sending frequency by sending null messages on each change of the EOT, instead of only when there were no local safe events. With more frequent null messages, we achieved an EOT and EIT quality of 46.88% for simulations using two LPs. Simulation runtime decreased from 6315s to 5340s. Idle time decreased from 25.37% to 18.48%. However, for all simulations using more than two LPs, the overheads of sending more frequent null messages increased the simulation runtime compared to the less eager strategy, as null messages were broadcast to all LPs on each EOT change.

In addition to varying the null message sending frequency, the EOT and EIT quality can also be increased by improving the lookahead calculation. In our simulation model implementation, link latencies in ms are drawn from a uniform distribution on the interval $[10, 200]$. We therefore use 10 ms as the fixed lookahead, which covers only a small proportion of the available maximum lookahead in the model. We expect that applying the methods for dynamic lookahead calculation discussed in Section 4.2 to our simulator implementation will further improve simulation performance.

6. CONCLUSION

We studied the potentials for efficient conservative distributed simulation of peer-to-peer networks of Kademlia-based networks. A partitioning scheme mirroring the construction of the simulated nodes' routing tables substantially reduces the communication required between processors executing the simulation, while a partitioning scheme based on location can reduce synchronization overheads. Using the partitioning scheme based on the simulated nodes' routing tables, our distributed simulator implementation enables close to linear reductions in memory usage per processor and runtime reductions up to a factor of 6.0 with 16 LPs. The simulator code is available to the community and can simulate a network of 10 million peers for one simulated hour in about 4 hours of wall-clock time. We introduced novel metrics that can be used to identify the potential for efficiency improvements in the synchronization scheme. Performance measurements show that substantial efficiency potentials remain that can be exploited by dynamically assessing the simulation state during synchronization. In our future work, we will study the impact of optimizations to the synchronization mechanism under the proposed efficiency metrics and with respect to overall performance.

7. REFERENCES

[1] P. Andelfinger and H. Hartenstein. Towards Performance Evaluation of Conservative Distributed Discrete-Event Network Simulations Using Second-Order Simulation. In *Proc. of the 2013 ACM SIGSIM Conf. on Principles of Advanced Discrete Simulation*, pages 221–230, 2013.

[2] M. J. Arif, S. Karunasekera, and S. Kulkarni. GeoWeight: Internet Host Geolocation Based on a Probability Model for Latency Measurements. In *Proceedings of the 33rd Australasian Conference on Computer Science*, ACSC '10, pages 89–98. Australian Computer Society, Inc., 2010.

[3] R. Bagrodia and M. Takai. Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages. *IEEE Transactions on Parallel and Distributed Systems*, pages 395–411, 2000.

[4] W. L. Bain and D. S. Scott. An Algorithm for Time Synchronization in Distributed Discrete Event Simulation. In *Proc. of the SCS Multiconference on Distributed Simulation*, pages 30–33, 1988.

[5] G. Bottoni and R. Barzaghi. Fast Collocation. *Bulletin Géodésique*, 67(2):119–126, 1993.

[6] R. E. Bryant. Simulation of Packet Communication Architecture Computer Systems. Technical report, 1977.

[7] K. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, 1979.

[8] M.-K. Chung and C.-M. Kyung. Improving Lookahead in Parallel Multiprocessor Simulation Using Dynamic Execution Path Prediction. In *Proc. of the 20th Workshop on Principles of Advanced and Distributed Simulation*, pages 11–18. IEEE Computer Society, 2006.

[9] B. Cota and R. Sargent. Automatic Lookahead Computation for Conservative Distributed Simulation. *Techn. Report*, (8916), 1989.

[10] T. T. A. Dinh, M. Lees, G. Theodoropoulos, and R. Minson. Large Scale Distributed Simulation of P2P Networks. In *16th Euromicro Conf. on Parallel, Distributed and Network-Based Processing*, pages 499–507, 2008.

[11] T. T. A. Dinh, G. Theodoropoulos, and R. Minson. Evaluating Large Scale Distributed Simulation of P2P Networks. In *12th IEEE/ACM Int'l Symp. on Distributed Simulation and Real-Time Applications*, pages 51–58, 2008.

[12] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. Profiling a Million User DHT. In *Proc. of the 7th ACM SIGCOMM Conf. on Internet Measurement*, pages 129–134. ACM, 2007.

[13] R. Fujimoto. Parallel and Distributed Simulation. In *Proceedings of the Winter Simulation Conference*, volume 1, pages 122–131 vol.1, 1999.

[14] R. Fujimoto, K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley. Large-Scale Network Simulation: How Big? How Fast? In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems*, pages 116–123, 2003.

[15] R. M. Fujimoto. Lookahead in Parallel Discrete Event Simulation. *Proceedings of the 1988 Int'l Conference on Parallel Processing, Vol. 3*, pages 34–41, August 1988.

[16] Q. He, M. Ammar, G. Riley, H. Raj, and R. Fujimoto. Mapping Peer Behavior to Packet-Level Details: a Framework for Packet-Level Simulation of Peer-to-Peer Systems. In *11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems*, pages 71–78, 2003.

[17] V. Jha and R. Bagrodia. A Performance Evaluation Methodology for Parallel Simulation Protocols. *ACM SIGSIM Simulation Digest*, 26(1):180–185, 1996.

[18] K. Jünemann, P. Andelfinger, and H. Hartenstein. Towards a Basic DHT Service: Analyzing Network Characteristics of a Widely Deployed DHT. In *Proc. of the 20th Int'l Conf. on Computer Communications and Networks*, pages 1–7, 2011.

[19] P. Leopardi. A Partition of the Unit Sphere into Regions of Equal Area and Small Diameter. *Electronic Transactions on Numerical Analysis*, 25(12):309–327, 2006.

[20] S. Lin, A. Pan, R. Guo, and Z. Zhang. Simulating Large-Scale P2P Systems with the WiDS Toolkit. In *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 415–424. IEEE, 2005.

[21] Y.-B. Lin and E. Lazowska. Exploiting Lookahead in Parallel Simulation. *Parallel and Distributed Systems, IEEE Transactions on*, 1(4):457–469, 1990.

[22] J. Liu and D. M. Nicol. Lookahead Revisited in Wireless Network Simulations. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*, pages 79–88. IEEE Computer Society, 2002.

[23] A. Loewenstern. BitTorrent Enhancement Proposal 5: DHT Protocol. `http://www.bittorrent.org/beps/bep_0005.html`, 2008.

[24] M. L. Loper and R. M. Fujimoto. Pre-Sampling as an Approach for Exploiting Temporal Uncertainty. In *Proc. of the 14th Workshop on Parallel and Distr. Simulation*, PADS '00, pages 157–164. IEEE Computer Society, 2000.

[25] G. Marsaglia. Choosing a Point from the Surface of a Sphere. *The Annals of Mathematical Statistics*, 43(2):645–646, 1972.

[26] P. Maymounkov and D. Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 53–65. 2002.

[27] R. Meyer and R. Bagrodia. Improving Lookahead in Parallel Wireless Network Simulation. In *6th Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 262–267, 1998.

[28] R. A. Meyer and R. L. Bagrodia. Path Lookahead: a Data Flow View of PDES Models. In *13th Workshop on Parallel and Distributed Simulation*, pages 12–19. IEEE, 1999.

[29] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman, and D. Chalmers. The State of Peer-to-Peer Simulators and Simulations. *SIGCOMM Computer Communication Review*, 37(2):95–98, 2007.

[30] P. Peschlow, A. Voss, and P. Martini. Good News for Parallel Wireless Network Simulations. In *Proc. of the 12th ACM Int'l Conf. on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 134–142. ACM, 2009.

[31] R. S. Pienta and R. M. Fujimoto. On the Parallel Simulation of Scale-Free Networks. In *Proc. of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 179–188. ACM, 2013.

[32] B. R. Preiss and W. M. Loucks. The Impact of Lookahead on the Performance of Conservative Distributed Simulation. In *Modelling and Simulation, Proc. of the European Simulation Multiconference*, pages 204–209. Citeseer, 1990.

[33] M. Quinson, C. Rosa, and C. Thiery. Parallel Simulation of Peer-to-Peer Systems. In *CCGrid 2012 – The 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 668–675, 2012.

[34] G. F. Riley, M. H. Ammar, R. M. Fujimoto, A. Park, K. Perumalla, and D. Xu. A Federated Approach to Distributed Network Simulation. *ACM Transactions on Modeling and Computer Simulation*, pages 116–148, 2004.

[35] M. Steiner, T. En-Najjary, and E. W. Biersack. Long Term Study of Peer Behavior in the KAD DHT. *IEEE/ACM Trans. Netw.*, 17(5):1371–1384, Oct. 2009.

[36] J. Wang, Z. Dong, S. Yalamanchili, and G. Riley. Optimizing Parallel Simulation of Multicore Systems Using Domain-Specific Knowledge. In *Proceedings of the 2013 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 127–136. ACM, 2013.

[37] B. Zarei and M. Pidd. Performance Analysis of Automatic Lookahead Generation by Control Flow Graph: Some Experiments. *Simulation Practice and Theory*, 8(8):511–527, 2001.