Model-Based Concurrency Analysis of Network Simulations

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

ABSTRACT

To achieve highest performance, parallel simulation of networks on modern hardware architectures depends on large numbers of independent computational tasks. However, the properties determining a network model's concurrency are still not well understood. In this paper, we propose an analytical model that enables concurrency estimations based on model knowledge and on statistics gathered from sequential simulation runs. In contrast to an automated concurrency analysis of event traces, the analytical approach enables insights into the relationship between the topology and communication patterns of the simulated network, and the resulting concurrency. We consider three fundamentally different network models as implemented in the network simulators PeerSim and ns-3: a large-scale application-layer peer-to-peer network, IP-based routing in a fixed topology, and a wireless ad-hoc network. For each model, we conduct an in-depth analysis, exposing the relationships between model characteristics and concurrency. Our analysis is validated by comparing estimated concurrency values to reference results of a trace-based analysis. The identification of key factors for concurrency forms a step towards a classification of network models according to their potential for parallelization.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*Modeling Techniques*; I.6.1 [**Computing Methodology**]: Simulation Theory—*Model Classification*; I.6.8 [**Computing Methodology**]: Types of Simulation—*Parallel, Distributed, Discrete Event*

General Terms

Performance

Keywords

concurrency, parallelism, simulation, parallel, distributed, discrete-event, network simulation, network models

SIGSIM-PADS'15, June 10–12, 2015, London, United Kingdom.

DOI: http://dx.doi.org/10.1145/2769458.2769464.

1. INTRODUCTION

The benefit of parallelization of a discrete-event network simulation model depends on properties of the considered network. Due to the complex interactions between the network topology, the communication patterns and the simulator realization, there is still a lack of guidelines to decide upon the suitability of different classes of network models for parallel simulation.

In the past years, there has been a renewed interest in parallel simulation due to the diminishing increases in single-core performance of processors. In addition to traditional approaches in CPU-based multi-core or supercomputing environments, the increasing prevalence of parallel hardware architectures such as graphics processing units and CPU-based many-core accelerators has led researchers to consider simulator designs for parallel execution on hundreds or thousands of tightly coupled cores [1, 5, 15, 18]. The novel simulator designs enable an efficient execution of network models previously considered unsuitable for parallel simulation due to their fine-grained computations and the need for frequent synchronization between cores. However, to occupy the enormous numbers of available cores, large numbers of independent computational tasks are required. Hence, the *concurrency* of simulation models, i.e., the maximum number of simulation events that can be executed in parallel while still maintaining simulation correctness, comes into focus as a key property for efficient parallel simulation. We distinguish concurrency from *parallelism*, i.e., the number of events executed in parallel in a real-world simulation run, as well as from the resulting real-world simulation *performance*, both of which are subject to the simulator realization.

In this paper, we conduct an analysis of network simulation models to expose the relationships between the models' network topologies and communication patterns, and their concurrency. We analyze three network models representing distinct classes of networks and show how the parameters required for analytical concurrency estimation can be calculated. We expect the presented results to facilitate the analysis of further network models with similar topologies and communication patterns, so that our analysis forms a step towards a classification of network models according to their concurrency, i.e., according to their aptness for parallelization. We validate the concurrency estimations by comparison with reference values gathered from event traces of sequential simulation runs. In addition to guiding parallelization decisions, the analysis enables a closer understanding of differences in concurrency between models of different classes of networks.

Our main contributions are as follows:

1. Estimation Model: We present an analytical model to estimate the concurrency of network models based on model knowledge and on statistics gathered from sequential simulation runs. The analytical approach enables an understanding of the causes of a network model's degree of concurrency. Our validation shows that reasonably accurate estimations can be achieved even when abstracting from the network model's topology to a large degree.

2. Network Model Analysis: We apply the estimation model to three fundamentally different network models as implemented in popular simulators: a large-scale peer-to-peer network, IP-based routing in a fixed topology, and a wireless ad-hoc network. The analysis uncovers the relationship between network model properties and concurrency.

The remainder of the paper is structured as follows. In Section 2, we give a brief overview of our approach to the construction of our analytical concurrency model. In Section 3, we discuss related approaches to concurrency evaluation. In Section 4, we present our methodology and our analytical model for concurrency estimation. In Section 5, we analyze three network models' event patterns, conduct a sensitivity analysis and estimate the models' concurrency. In Section 6, we evaluate the accuracy of our concurrency estimation. In Section 7, we discuss challenges of the analytical approach. Section 8 gives a summary of our results and concludes the paper.

2. OUR APPROACH

Our goal is to provide an analytical model to *estimate* a network simulation model's concurrency in a white-box fashion, i.e., while enabling insights into the causes of the network model's concurrency. A network model's *exact* concurrency can be determined using *critical path analysis*, a well-known black-box approach that will be introduced in Section 3. Critical path analysis provides reference values that our analytical model will be validated against. However, the properties of critical path analysis render it a difficult target for mathematical analysis.

Instead, we construct our analytical model by analyzing *YAWNS* (cf. Section 3), a second black-box concurrency evaluation approach that lends itself to mathematical modeling and whose results are close to critical path analysis. Our analytical model successfully approximates the results of YAWNS, and hence, the reference results gathered using critical path analysis.

Our analysis applies to simulations under *conservative* synchronization, where the synchronization algorithm ensures correctness prior to each event execution. *Optimistic* synchronization may potentially enable larger concurrency, but is out of the scope of our work.

3. RELATED WORK

In this section, we give a brief summary of two classes of methods for concurrency evaluation: trace-based approaches determine the concurrency in a network model by a programmatic analysis of event traces generated during sequential simulation runs. These approaches determine the concurrency in the model accurately under their stated assumptions, but are performed in a black-box fashion that limits insights into the sources of the identified concurrency. Analytical approaches require more manual effort and usually determine a rough estimation of concurrency, yet may allow for an understanding of concurrency potentials and limitations, as is the intention of the analytical model and results presented in this paper.

3.1 Trace-Based Concurrency Evaluation

In discrete-event network models, communication activities are modeled as timestamped *events* representing instantaneous state changes of the simulated nodes. The communication patterns in a given network model define a precedence relation governing the event execution order. For instance, subsequent message arrivals at a single node must be simulated in timestamp order to maintain the correctness of the node state. An event can safely be executed as soon as no remaining precedence relationships demand the prior execution of other events. An additional constraint is given by the *lookahead*, which defines an upper bound for the delta between the current simulation time and the timestamp of events that can be executed safely. The magnitude of valid lookahead values depends on network model properties, e.g., on the link latencies between simulated nodes. In synchronous simulation approaches, processor cores can be considered to execute safe events in lock-step. The concurrency of the network model is the average number of events executed at the same time, disregarding simulation overheads and limitations in the number of cores.

Critical path analysis [4, 9] is a method to determine the concurrency in a simulation model by traversing a dependency graph reflecting the precedence relationships between the events of a previous sequential simulation run of the considered model. In the dependency graph depicted in Figure 1, events are represented by circles. An arrow between events e_1 and e_2 reflects the precedence relationship "e_1 before e_2". There are two causes of precedence relationships: first, events cannot be processed prior to their creation in the course of the simulation. Hence, there is an edge reflecting the precedence of an event e over any new events created by e. Second, to enforce timestamp ordering of events in each node, there is an edge between an event and its direct predecessor w.r.t. simulated time pertaining to the same node. Critical path analysis can be performed in an iterative fashion: events that are safe to be executed according to the precedence relation are removed, allowing further events to be removed in the next execution. In Figure 1, dashed rectangles indicate groups of events that can be processed at the same time, i.e., concurrently. If equal processing time for all events is assumed, the total number of executions, divided by the number of events in the simulation in total, is the average concurrency of the simulation model assuming an unlimited number of processor cores, infinite lookahead, and no overheads for synchronization and communication. In Figure 1, nine events are processed in a total of six executions. Hence, the concurrency is $9/6 = 1.5$.

YAWNS [13] is a well-known synchronous synchronization algorithm for parallel and distributed simulation. A simulation iteration using YAWNS is illustrated in Figure 2. In each iteration, the timestamp t_{min} of the earliest event is determined. A fixed lookahead value l determined according to model properties gives a lower bound on the timestamp delta between an event and its creation. Given t_{min}, $l \in \mathbb{N}$, all events in the *lookahead window* $[t_{min}, t_{min} + l]$ are guaranteed to create no events with timestamps below $t_{min} + l$. As the lookahead window is a closed interval, its

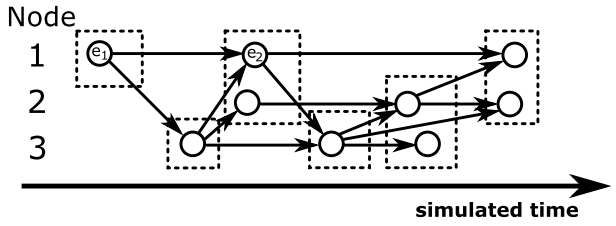


Figure 1: Critical path analysis of a dependency graph.

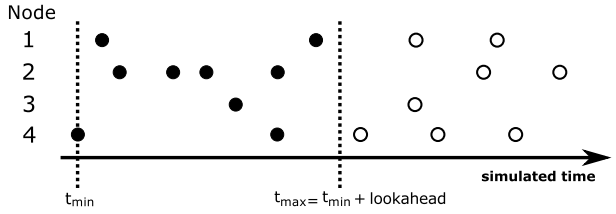


Figure 2: A simulation iteration using YAWNS: events in the interval $[t_{min}, t_{max}]$ create no events with timestamps below t_{max} and are thus safe to be executed in parallel.

width is $l + 1$. Events pertaining to different nodes within each lookahead window can be processed concurrently without allowing for violations of timestamp order per node. The number of executions required to process a lookahead window is the largest number of events pertaining to a single node. In the example, nine events in the lookahead window can be processed in four executions. Hence, the concurrency within the shown lookahead window is $9/4 = 2.25$. Due to its conceptual simplicity, YAWNS has been used as a basis for analytical concurrency estimation in previous works [13, 17]. Here, we employ YAWNS in two ways: first, we analytically estimate the expected YAWNS concurrency on the basis of key parameters of network models. Second, we show that the results between a concurrency analysis using critical path analysis and YAWNS are sufficiently close to use these approaches interchangeably when evaluating the potential of network models for parallelization.

3.2 Analytical Concurrency Estimation

Existing works proposed a multitude of approaches to performance estimation of parallel and distributed simulations. In this paragraph, we limit our brief summary of previous works to approaches that focus on allowing for insights in the causes of the concurrency available in simulation models, disregarding black-box approaches based on simulative models or event traces.

In 1993, Nicol [13] studied the concurrency of simulations based on YAWNS. Using a stochastic model, it is shown that synchronization overheads relative to event processing costs decrease with larger simulation activity. Calculated bounds are evaluated for a number of example models. Similar to our work, concurrency analysis is performed by estimating the number of events in each lookahead window. However, contrary to Nicol, we focus on specific network models implemented in popular simulators and show how concurrency estimations can be derived with relative ease based on network model properties readily available to modelers.

In 1999, Liu et al. [7] presented back-of-the-envelope calculations to estimate the runtime of parallel simulations. However, while the costs for communication in the execution environment are taken into account, load imbalances, which determine the model's concurrency, are not considered. Hence, the proposed approach is not applicable if we are interested in the concurrency of the network model.

In 2003, Varga et al. [20] proposed an efficiency criterion for conservative parallel simulation using the null-message algorithm. Their approach relates the number of events per lookahead window to the communication costs in the execution environment, allowing for rough estimations of simulation efficiency. Since differences in the workload for different groups of nodes in the simulated network are not considered, estimation errors will be large if there are substantial load imbalances. Improvements in estimation accuracy of their approach may be possible by conducting a manual network model analysis as presented in our paper, and relating the resulting estimation to the expected communication costs.

In 2013, Pienta et al. [17] analyzed the concurrency of scale-free networks. Based on an assumed communication pattern within the network, their analysis provides an estimation of the expected concurrency under YAWNS-based synchronization. To determine the stationary distribution of events among the simulated nodes, an iterative calculation is performed until a steady-state is reached. Again, our work differs in the consideration of specific network models as implemented in real-world simulators. Further, an analysis of the communication patterns in the studied network models allows us to base our estimation on expected numbers of events within a single lookahead window without the need for an iterative model.

4. ESTIMATING CONCURRENCY

In this section, we first describe the building blocks and assumptions of our concurrency estimation methodology. We then propose an analytical concurrency model as a basis for the analysis of specific network models.

4.1 Methodology

Critical path analysis allows for a trace-based calculation of the concurrency in the network model assuming infinite lookahead. However, typically, parallel simulation is performed under a fixed lookahead value that restricts the concurrency that is visible to the simulator. Thus, the results given by critical path analysis are not sufficient to determine the effective maximum concurrency.

To gather more realistic estimates, we adapt critical path analysis by limiting the lookahead during each analysis iteration. The adapted critical path analysis determines an upper bound for the average number of events that can realistically be executed in parallel given unlimited processor cores and assuming no overhead for synchronization and communication between cores. In addition, we apply the common assumption of identical computational costs for all events. In effect, the adapted critical path analysis coincides with an analysis of event traces similar to YAWNS, with the difference that a new lookahead window is calculated after each execution of events. Hence, a YAWNS-based analysis will typically determine lower concurrency values than the upper bound calculated by critical path analysis.

Ideally, we would derive an analytical model to estimate critical path analysis results directly. However, we are not aware of a method to analytically estimate critical path analysis without relying on an iterative, i.e., numerical approach. As critical path analysis allows for overlapping lookahead windows, each iteration depends on the results of the previ-

ous iteration. However, our aim is a concurrency estimation that enables insights beyond the sheer concurrency values determined by iterative approaches.

In contrast, when analyzing YAWNS, each lookahead window can be considered separately. Therefore, the analytical model that will be presented in Section 4.2 estimates concurrency according to YAWNS. For the analytical results to be meaningful, it is important that the estimations are close to the reference results from critical path analysis. In Section 6, we show that for all considered network models, the deviations between the results of critical path analysis, YAWNS and our analytical model are sufficiently low so that the methods can be used interchangeably to guide decisions on the parallelization of network models.

Further, the benefit of the analytical approach hinges on the simplicity of acquiring the necessary inputs to the analytical model from the given network model. In Section 4.2, we will detail the information required to gather estimations for three network models with fundamentally different characteristics. We argue that for the considered network models, the required inputs to the analytical model can be determined with relative ease.

4.2 Analytical Model

Our fundamental approach to concurrency estimation is to determine the characteristics of an average lookahead window. In Section 3.1, we have seen that the concurrency within a single lookahead window of a YAWNS-based simulation is the total number of events e_{total} in the lookahead window, divided by the largest number m of events pertaining to a single simulated node. Hence, given estimates of m and e_{total}, the estimated concurrency of the network model is

$$Y_{est} := \frac{e_{total}}{m}$$

While e_{total} can easily be estimated based on the communication activity in a network model, we need to derive m from an estimate of the distribution of events to simulated nodes. As we will see in Section 5 on the example of concrete network models, simulated nodes can be grouped into a small number of categories, each category being assigned a distinct number of events total. As an approximation, events assigned to each category are assumed to be distributed uniformly among the nodes in the category. The *number of nodes in each category* and the *number of events assigned to each category* are the inputs from which our analytical model derives m.

More formally, we divide the nodes of the simulated network into c categories so that all n_i nodes in a single category i share the same estimated number of events e_i per lookahead window. Within each category, we consider the assignment of events to nodes as a sequence of Bernoulli trials with probability $p_i = 1/n_i$ each. The probability for a *single* node of category i being assigned $\leq k$ events follows the binomial distribution:

$$F_i(k) = \sum_{j=0}^{k} \binom{e_i}{j} {p_i}^j (1-p_i)^{e_i-j}$$

The probability for *all* nodes of category i being assigned $\leq k$ events is then:

$$G_i(k) = F_i(k)^{n_i}$$

By considering all node categories, we arrive at the probability for all nodes *of all categories* being assigned $\leq k$ events:

$$G(k) = \prod_{i=1}^{c} G_i(k)$$

We are interested in the expectation of random values distributed according to G, i.e., the expected largest number of events any single node is assigned in a lookahead window [3]. The expectation is:

$$m = \sum_{k=1}^{\infty} k g(k)$$

Here, m is the expected number of parallel event executions required to process a single lookahead window. Now, given the expected total number of events in a lookahead window e_{total}, the estimated YAWNS concurrency is: $Y_{est} := e_{total}/m$

In the following, we show how the node categories and event counts can be determined for concrete network models.

5. NETWORK MODEL ANALYSIS

In this section, we study the concurrency of three network models. For each model, we first analyze the event patterns resulting from the communication patterns in the simulated network. Second, we conduct a sensitivity analysis to find the key factors determining the model's concurrency. Third, we determine the parameters required to analytically estimate the model's concurrency.

5.1 Peer-to-Peer Overlay Network

As our first example, we study a model of a Kademlia-based network. Kademlia [11] is a protocol used to form a distributed hash table (DHT) enabling the efficient storage and retrieval of key-value pairs in a peer-to-peer network. We have previously shown that models of Kademlia-based networks contain substantial concurrency and can benefit strongly from parallelization using SMP clusters or many-core architectures [1, 2]. We analyze the Kademlia model with reference to an implementation in the PeerSim network simulator[1]. As is common with peer-to-peer networks, the model abstracts from all OSI layers but the application layer, i.e., the physical topology is reflected by link latencies drawn from a random distribution. The application-layer itself is modeled accurately in accordance with the BitTorrent DHT specification [10].

5.1.1 Event Patterns

There are two sources of traffic in Kademlia: communication triggered actively by users of the DHT, and routing table maintenance. The latter comprises both operations for refreshing bucket contents as well as operations for checking the responsiveness of specific peers.

The event patterns representing the communication activities in the Kademlia model are shown in Figure 3. The building block fundamental to all communication in Kademlia is the remote procedure call (RPC), a sequence of three events representing the following interaction:

A. Peer 1 sends a request

B. Peer 2 receives the request and creates a response

C. Peer 1 receives the response.

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

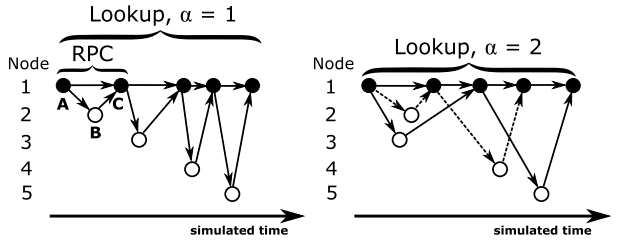


Figure 3: Event patterns in the Kademlia model: lookups are composed of α overlapping sequences of RPCs. Node 1 performs a lookup with $\alpha = 1$ (left) and $\alpha = 2$ (right).

So-called *lookups* are used to perform storage and retrieval operations. Each lookup consists of a sequence of RPCs where step C generates a new request until the lookup terminates. A parameter α specifies the number of concurrent RPCs during a lookup. Lookups with $\alpha > 1$ can be regarded as a superposition of multiple sequences of RPCs.

From the event pattern, we can easily deduce the number of events associated with a lookup. One initial event triggers the lookup, and each subsequent RPC is associated with two events: a request and its response. If the number of RPCs r per lookup is known, the total number of events per lookup is $e_{per\_lookup} = 2r + 1$, independently of α. Of these, $r + 1$ events pertain to the peer performing the lookup, and r events pertain to other peers.

The remaining traffic in the Kademlia model is created by pings triggered if the responsiveness of a peer is to be checked. If a peer's routing table is fully populated and a peer becomes aware of a new remote peer, peers of unknown responsiveness in the routing table are checked using ping RPCs. The receiver of a ping request may then recursively trigger new pings to further peers.

When creating the inputs to the analytical model, we consider the events created by lookups in detail, while treating ping events as uniformly distributed noise.

Concurrency in the Kademlia model results from two independent parameters:

1. λ independent lookups running concurrently
2. α concurrent RPCs performed during each lookup

Since events pertaining to each individual peer must be executed sequentially in timestamp order, the resulting concurrency is limited by the number n of peers in the network.

5.1.2 Sensitivity Analysis

We study the sensitivity of the Kademlia model to the number n of peers in the network, the number λ of concurrent lookups, and the number α of concurrent RPCs per lookup. To set a fixed λ accurately, we require an estimate of the average lookup duration d, which can be gathered from a brief initial simulation run. Then, the rate r at which lookups must be generated to achieve the desired number of λ concurrent lookups follows Little's law: $r = \frac{\lambda}{d}$.

For runs with $\lambda = 100$ and $\lambda = 1{,}000$, we triggered the generation of event traces after 1,000s of simulated time to allow the network to reach a steady state. Using the adapted critical path analysis according to Section 4.1, we analyzed events executed within 10s of simulated time. However, since the results differed only slightly with shorter runs, we configured the computationally expensive runs for $\lambda =$ 10,000 with only 300s of warm-up time. Since link latencies in ms are drawn from a uniform distribution on $[10, 200]$, a fixed lookahead value of 10ms was used. From the results depicted in Figures 4 and 5, we can see that, as expected, larger numbers of concurrent lookups result in larger concurrency. Furthermore, larger α provides an increase in concurrency. In both figures, we can see that concurrency is limited by the network size. The results provide an upper bound for the speedup by parallelization of the network model. Depending on the costs of communication and synchronization during simulation, many of the considered parameterizations suggest the parallel execution of the model on hundreds or thousands of processor cores. Hence, as shown on the example of a GPU-based simulator in [1], the results indicate that the Kademlia network model can benefit substantially from execution on many-core devices.

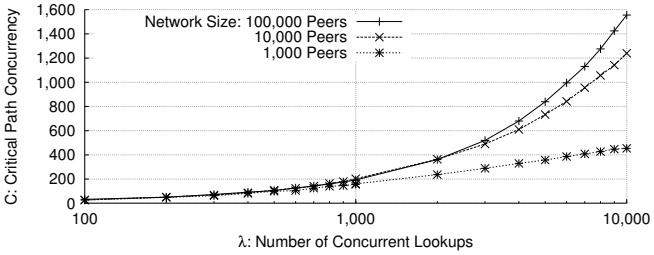


Figure 4: Results of sensitivity analysis of the Kademlia model with $\alpha = 8$, varying the number λ of concurrent lookups.

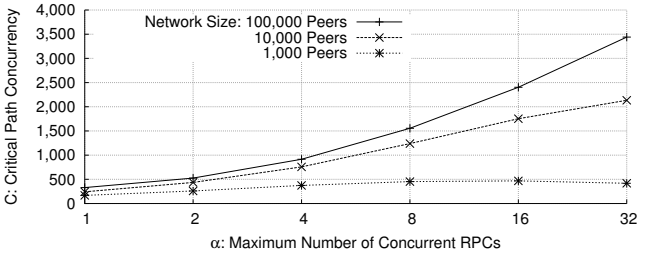


Figure 5: Results of sensitivity analysis of the Kademlia model with $\lambda = 10{,}000$, varying the number α of concurrent RPCs per lookup.

5.1.3 Analytical Concurrency Estimation

In the following, we describe how, based on key metrics of the Kademlia model, we determine the inputs required for the analytical concurrency estimation. The inputs to the analytical model are printed in boldface.

We differentiate between two categories of peers: active *peers that are currently executing a lookup, and* passive *peers that respond to incoming requests only.* We assume that the number λ of concurrent lookups is a scenario parameter, and that the initiations of lookups are distributed uniformly over the peers in the simulated network. The average number λ_{rt} of additional concurrent lookups created for routing table maintenance can be gathered from a sequential simulation run of the given configuration and subsequently be included in our consideration:

$$\lambda' = \lambda + \lambda_{rt}$$

Then, the ratio of active peers of all peers is:

$$r_{active} = 1 - (1 - \frac{1}{n})^{\lambda'}$$

The absolute numbers of active and passive peers are thus:

$$n_{active} = n \times r_{active}$$
$$n_{passive} = n - n_{active}$$

Given the width of the lookahead window $l+1$, the average number r of RPCs per lookup, and the average lookup duration d, the total number of events generated for all lookups per lookahead window is:

$$e_{lookups} = (l+1) \times \lambda' \times \frac{2r+1}{d}$$

Finally, we consider the number p of ping RPCs per unit of simulated time generated for checking the online status of peers, each generating two events, to obtain the total number of events per lookahead window:

$$e_{total} = (l+1) \times (\lambda' \times \frac{2r+1}{d} + 2p)$$

Now, we consider active and passive peers separately. In each lookup, active peers generate one initial event and one event for each RPC. The number of these events for all active peers is:

$$e_{active,lookup} = (l+1) \times \lambda' \times \frac{r+1}{d}$$

Active peers also receive some of the requests generated in lookups of other peers. The number of request events for all active peers is:

$$e_{active,request} = (l+1) \times \lambda' \times r_{active} \times \frac{r}{d}$$

Finally, a proportion of ping events targets active peers:

$$e_{active,ping} = (l+1) \times p \times r_{active}$$

Now, the total number of events expected to be generated per lookahead window for all active peers is:

$$e_{active} = e_{active,lookup} + e_{active,request} + e_{active,ping}$$

The remaining events pertain to passive peers:

$$e_{passive} = e_{total} - e_{active}$$

Using the estimated number of events for the two categories of active and passive peers, we can now determine the expected largest number m of events per lookahead window to be processed by a single peer in the simulation according to the calculations described in Section 4.2. The estimated concurrency is then e_{total}/m.

Discussion: In the simulated network, each lookup creates a sequence of RPCs targeting a sequence of peers according to the dynamic contents of the routing tables of peers on the path to the target of the lookup. Nevertheless, in the analysis, we consider the events pertaining to each peer category as uniformly distributed among the peers in the respective category, ignoring the network topology created by the Kademlia protocol completely. In Section 6 we will see that our estimations are reasonably accurate. Hence, we can conclude that the impact of the exact topology of the considered Kademlia-based network on the concurrency of the network model is relatively low. Instead, the concurrency is dominated by the raw message counts per peer category as well as by the overall network size.

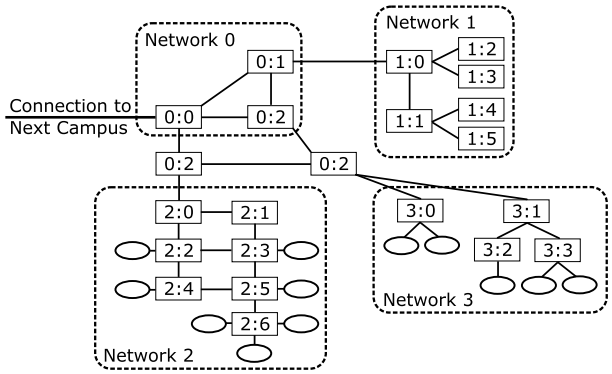


Figure 6: A single campus network of the topology from the NMS program (Fig. adapted from [14]).

5.2 TCP/IP in a Fixed Topology

Our second example is a network model created in the context of the DARPA "Network Modeling and Simulation" (NMS) program. The model is commonly used to benchmark parallel simulators [6, 14, 19] and was selected for its strong impact of the network topology on concurrency.

The basic building block of the topology is the campus network depicted in Figure 6. On each campus, there are three subnetworks. In Network 0 and Network 1 and between Network 0, 1 and 3, there are nodes representing servers and routers interconnected by a 1 Gbps link with 5ms latency. In Network 2 and 3, there are local area networks containing a configurable number of nodes representing user workstations connected to a switch using 100 Mbps links with 1ms latency. A configurable number of campus networks is connected in a ring using links with a latency of 200 ms. For each of the LAN nodes, a TCP stream with a constant data rate of 500 kbps is transmitted by one of the nodes 1:2, 1:3, 1:4 or 1:5 of the neighboring campus network.

Since all messages pass through the nodes connecting individual campus networks, we study the effects of varying the bandwidth between these nodes between 1 Mbps and 1,000 Mbps. In the following, we refer to the nodes connecting the individual campus networks as *hubs*. In addition, we differentiate between two types of bottlenecks: *network bottlenecks* are nodes that due to their position in the network and their limited bandwidth restrict the overall throughput in the network. *Simulation bottlenecks* are nodes for which disproportionally large numbers of events must be processed per unit of simulated time, so that these nodes limit the concurrency of the simulation model.

An implementation of the network model in the popular network simulator ns-3[2] uses an accurate representation of the network and transport layer, whereas the lower layers are modeled by the fixed link latencies specified above. In our analysis, we apply the common approach of using a fixed lookahead value of 1ms that is applicable to all nodes in the network. It may be possible to extract larger concurrency with a dedicated lookahead value for each link at the cost of higher complexity of the synchronization scheme (e.g., [12]).

5.2.1 Event Patterns

Since it is not always possible to transmit messages created by a simulated application instantaneously, in ns-3,

[2] `http://www.nsnam.org/`

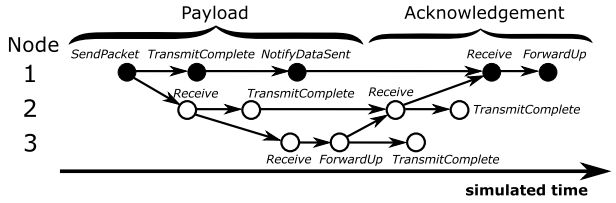


Figure 7: Event patterns in the TCP/IP model: a single packet is transmitted from node 1 to 3 via node 2. Node 3 replies with an acknowledgement.

creation of messages and their transmission is modeled separately. Hence, the transmission of a single message holding a payload via a linear sequence of nodes is reflected by the events and precedence relationships depicted on the left hand side of Figure 7: the sender generates one event for the message's creation (`SendPacket`), one for the message's successful transmission on the link layer (`TransmitComplete`), and one notifying the transport layer that a packet was sent (`NotifyDataSent`). Each node on the path to the receiver generates two events for reception (`Receive`) and successful forwarding (`TransmitComplete`) of the message. Finally, the receiving node generates two events representing reception: one for reception on the link layer (`Receive`), and one for forwarding the message to the upper layers of the network stack (`ForwardUp`).

Additional messages are created by TCP on the receiver side. We use the New Reno implementation of TCP, wherein by default, for every second message, an acknowledgement is transmitted from the receiver to the sender. As depicted on the right hand side of Figure 7, each acknowledgement generates one event for the receiver of the payload (`TransmitComplete`), two events for each hop on the path to the sender (`Receive`, `TransmitComplete`) and two events for the original sender (`Receive`, `ForwardUp`).

5.2.2 Sensitivity Analysis

In the analysis of the sensitivity of the model's concurrency to model parameters, we used 60s of warm-up time. Since the results were virtually independent of the considered span of simulated time, it was sufficient to analyze events executed within 1s of simulated time. The results in Figure 8 show the sensitivity of the model's concurrency to the number of campus network and the hub bandwidth for a fixed number of 16 LAN nodes. Since campus networks communicate only with their direct neighbors, larger numbers of campus networks do not increase the amount of traffic handled by individual hubs. Hence, irrespective of the hub bandwidth, there is a linear relationship between the number of campus networks and the critical path concurrency. A comparison between the curves for different hub bandwidths shows initially surprising results: the concurrency does not simply increase with larger hub bandwidth: even though a hub bandwidth of 1,000 Mbps allows for far fewer messages transmitted per unit of simulated time than a bandwidth of 10 Mbps, the larger number of messages crossing the hubs limits the concurrency of the simulation.

Figure 9 shows the concurrency when varying the number of LAN nodes and the hub bandwidth for a fixed number of 16 campus networks. With 2 LAN nodes, only 2,000 kbps of traffic crosses each hub, i.e., there is a network bottleneck in the run with 1 Mbps only. Accordingly, the results with

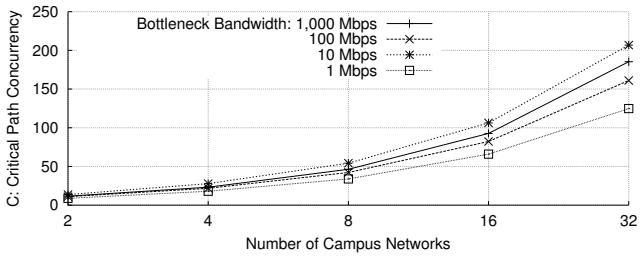


Figure 8: Results of sensitivity of the TCP/IP model varying the number of campus networks.

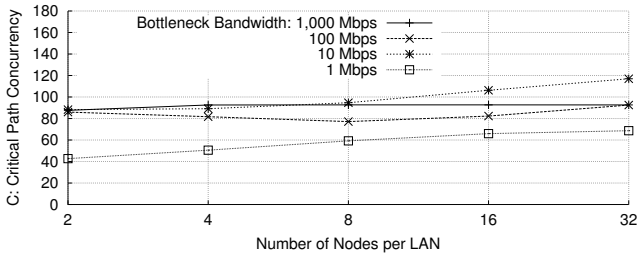


Figure 9: Results of sensitivity analysis of the TCP/IP model varying the number of receiving nodes per LAN.

hub bandwidths of 10 Mbps and above are nearly identical. For 4 and more LAN nodes, the magnitudes of the results do not simply follow the hub bandwidth. Instead, the resulting concurrency depends on three factors: the rate of message generation by the senders, the rate at which the messages pass through the network as dictated by the hub bandwidth, and the total number of message flows. With 1,000 Mbps, the concurrency is nearly independent of the LAN node count. The reason is that, since there are no network bottlenecks, each doubling of the LAN node count doubles the total number of messages per unit of time, but at the same time doubles the number of messages at each hub, i.e., double the original number of events is processed in double the number of executions.

When disregarding the costs of synchronization and communication during a simulation run, the concurrency with 32 campus networks suggests the use of up to about 200 cores for execution of the model. As the concurrency depends strongly on the number of campus networks, efficient parallel execution on many-core devices requires a large number of campus networks in relation to the number of cores.

5.2.3 Analytical Concurrency Estimation

For the analysis, we first assume that there are no network bottlenecks. Then, the total number e_{total} of events per TCP flow and simulated second in the steady-state can be estimated as follows:

$$m_{app} = r_{app}/s_m$$

$$e_{payload} = m_{app}(3 + 2n_{fw} + 2)$$

$$e_{ack} = \frac{m_{app}}{2}(1 + 2n_{fw} + 2)$$

$$e_{total} = e_{payload} + e_{ack}$$

Here, r_{app} is the desired bitrate of each application that generates a TCP flow. s_m is the size of each message including headers. In our example, TCP and IP add 20 bytes

of header data each to a payload of 512 bytes. m_{app} is the message rate per flow. n_{fw} is the average number of forwarding nodes between a sender and a receiver. Using these values, the total event rate is given by the sum of the payload and acknowledgement event rates. This calculation can be repeated for each TCP flow to determine the total event rate of the simulation.

Up to this point, we did not consider network bottlenecks. We model these by considering two message rates: as above, m_{app} is the target message rate of the application. m_{tm} is the number of messages actually transmitted per second when considering network bottlenecks. Of course, network bottlenecks must be identified first. For complex topologies, an approximation can be calculated using common flow algorithms. In the topology considered here, hubs with low bandwidth are obvious network bottlenecks. All other forwarding nodes handle substantially smaller numbers of events.

When considering network bottlenecks, the event rate estimation must be based on the rate of actually transmitted messages m_{tm} per flow according to the maximum message rate m_{hubs} of the hubs resulting from their bandwidth:

$$m_{app} = r_{app}/s_m$$

$$m_{tm} = min(m_{hubs}, m_{app})$$

$$e_{payload} = m_{tm}(2 + 2n_{fw} + 2) + m_{app}$$

$$e_{ack} = \frac{m_{tm}}{2}(1 + 2n_{fw} + 2)$$

$$e_{total} = e_{payload} + e_{ack}$$

The estimation makes the simplifying assumption that the combination of all TCP flows fully saturates the capacity of network bottlenecks. Since TCP only approximates the channel capacity, the average number of messages in flight will be somewhat lower than our estimation. In Section 6, we evaluate how strongly network bottlenecks affect the accuracy of the concurrency estimation.

We now explicitly consider the event rates of two categories of nodes: hubs and senders. Each of the n_{cns} campus networks holds a single hub and four senders:

$$n_{hubs} = n_{cns}$$

$$n_{senders} = 4n_{cns}$$

Since each campus network contains both senders and receivers, the number of TCP flows crossing each hub is $2 \times n_{lan\_nodes} \times n_{cns}$.

The total event rate for the hubs is thus:

$$e_{hubs} = 2n_{hubs} \times (m_{tm} + \frac{m_{tm}}{2})$$

Again, each forwarded message generates one event for reception and one for transmission and there is one acknowledgement for every other message.

The total event rate for the senders is:

$$e_{senders} = n_{flows} \times (m_{app} + 2m_{tm} + 2 \times \frac{m_{tm}}{2})$$

As before, for the concurrency estimation, we assume that e_{hubs} events are placed in the lookahead window with the same per-event probability for each hub. The corresponding assumption is made for the senders. Using our analytical model, we can now estimate the largest expected number of events in each lookahead window for a single node in either the hub or the sender group. The result m is the number of parallel event executions required to process a single lookahead window. The estimated concurrency is then e_{total}/m.

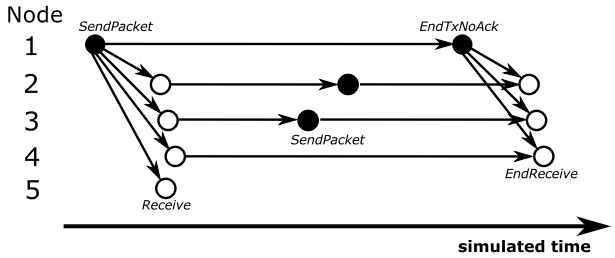


Figure 10: Event patterns in the wireless model.

Discussion: From the analysis, we can gather relationships between properties of the considered network model and the model's concurrency: first, since parallel simulation progress is determined by the simulation bottlenecks, a large number of events for non-bottleneck nodes is beneficial for high concurrency. Hence, given the fact that each hop forwarding a message generates two events, longer path lengths increase concurrency. Second, m_{tm} decreases if there are network bottlenecks, whereas m_{app} is independent of network bottlenecks. Therefore, it is possible that the total event rate is dominated by events generated at senders, even though all traffic passes through the hubs. Because of this, there is an inverse relationship between network and simulation bottlenecks: *hubs that* do not *limit the message rate are simulation bottlenecks, but no network bottlenecks, whereas hubs that* do *limit the message rate are network bottlenecks, but no simulation bottlenecks.*

5.3 Wireless Ad-Hoc Communication

As a third example, we study the concurrency of a model of a wireless ad-hoc network. The model's analysis forms a counter-example to the approach used to determine the concurrency of the previous two models: due to the broadcast nature of the wireless medium and the avoidance of message collisions, we can express the concurrency directly based on an analysis of individual transmissions, without reliance on the statistical approach presented in Section 4.2.

Ad-hoc networks are frequently studied in the context of car-to-car communication, where cars establish a mutual awareness by periodic transmission of *beacon* messages holding, e.g., the sender's current position. The mutual awareness can be leveraged by applications to increase road safety and efficiency. A frequent focus of simulation studies in this area lies in investigating the channel load resulting from the beacon traffic.

In the scenario considered here, a configurable number of nodes are positioned randomly on a linear 100m road segment. The nodes broadcast beacons at a configurable rate, each beacon comprising 400 bytes of data including headers. Communication is performed with a data rate of 6 Mbps over a wireless channel using a CSMA-based MAC layer, i.e., nodes check for activity on the channel and delay their transmissions if necessary.

5.3.1 Event Patterns

We study the event patterns resulting from the described model by reference to ns-3. A single transmission comprises the following sequence of events (cf. Figure 10): given no

ongoing transmission on the channel, a `SendPacket` event of the transmitting node represents the start of a transmission and creates a `Receive` event for every remaining node as well as a single `EndTxNoAck` event reflecting the completion of the transmission. For each receiver that successfully detects the beacon, the `Receive` event creates a corresponding `EndReceive`. In total, each successful transmission is reflected by a minimum of $1+(n-1)+1 = n+1$ and a maximum of $1+(n-1)+1+(n-1) = 2n$ events.

A CSMA-based MAC layer aims to reduce the probability of collisions. In case the channel is busy, the initial `SendPacket` event creates a single `AccessTimeout` event that takes the role of a `SendPacket` at a later point in simulated time. In the following, we refer to `SendPacket` events only, since `AccessTimeout` events are handled identically during simulation. We refer to `SendPacket` events by which a busy channel is detected as `Probe` events.

There are two situations in which interactions between multiple transmission attempts affect concurrency. First, collisions occur in the case of two nodes starting to send at the same time. Second, a `SendPacket` event will detect a busy channel and delay the corresponding transmission, so that no `Receive` events are created until the next attempt. The occurrence probabilities of both situations depend on the channel load. Our concurrency estimation disregards overlapping transmissions, but does consider `Probe` events.

5.3.2 Sensitivity Analysis

The sensitivity of the network model's concurrency to the beacon rate and the number of nodes was analyzed using event traces covering 10s of simulated time after a warm-up time of 30s. Figure 11 shows that the concurrency increases close to linearly with the number of nodes in the network. For extremely large channel loads, collisions increase the concurrency substantially. Further, slight differences in concurrency for lower beacon rates are caused by varying numbers of *Probe* events.

In the considered parameter combinations, we measured concurrency values below 100 even for large node densities. Due to the limited spatial extent of 100m of the network, larger node counts lead to unrealistically large channel load. Parallel execution on many-core devices should hence be considered when studying scenarios with larger spatial extent that support larger numbers of nodes under realistic channel loads.

5.3.3 Analytical Concurrency Estimation

To estimate the concurrency in the model analytically, we need to be aware of the lookahead that will be available in a simulation run. Simulations of wireless networks are well-known to exhibit only small amounts of fixed lookahead. Due to the broadcast nature and limited spatial extent of wireless networks, transmissions pertain to all nodes in proximity of the sender, and the time delta between transmission and reception is quite small. Hence, a fixed lookahead value considering the minimum latency between any two nodes of the network can be insufficient for high concurrency. The literature proposes the use of model knowledge regarding OSI layers 2 and above to enable larger lookahead values [8, 16]. If it is known at simulation runtime that according to the current state of, e.g., the MAC or application layer of the nodes, new events up to a certain point in time can be ruled out, the lookahead can be extended to this point.

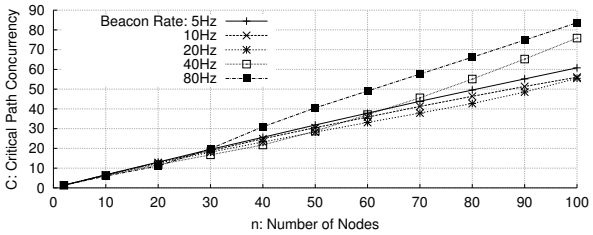


Figure 11: Results of sensitivity analysis of the wireless network model, varying the number of nodes and the beacon rate.

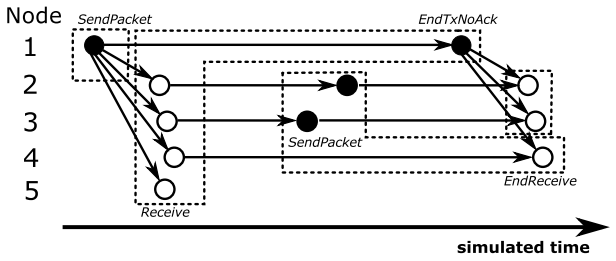


Figure 12: Concurrency in a single transmission in the wireless model. Sets of events surrounded by dashed lines can can be executed concurrently.

For the analysis, we consider the case where model knowledge provides sufficient lookahead to cover all events that have no pending precedence relationships. Figure 12 depicts the event patterns in the model, grouping concurrent events. The initial `SendPacket` event creates $n-1$ `Receive` events as well as a single `EndTxNoAck` event. Since execution of the `SendPacket` event triggers the creation of all other events, it cannot be executed in parallel with any further events. Now, all `Receive` events can be executed in parallel together with the `EndTxNoAck`, n events in total. Next, all remaining `SendPacket` events are executed concurrently with `EndReceive` events of nodes that execute no `Probe` events and receive the current packet successfully.

We now consider the number of parallel event executions required to process the `Probe` events. Since events for each node must be executed in timestamp order, up to n events can be executed at the same time. For a given simulation run of T transmissions with f_t `Probe` events during transmission t, the average number of event executions required to process all `Probe` events is $r_p = \frac{1}{T}\sum_{t=1}^{T}\lceil\frac{p_t}{n}\rceil$, the value of which can be determined from a sequential simulation run.

Now, we estimate the model's concurrency by dividing the number of events per transmission by the number of executions required. The estimated concurrency is

$$C_{est} = \frac{1+(n-1)+1+s(n-1)+p}{3+r_p} = \frac{n+1+s(n-1)+p}{3+r_p}$$

Discussion: The simplicity of the analysis reflects the simplicity of the event precedence relation in the wireless model: the available concurrency results from the independent reception events evenly distributed among all receivers. Hence, in contrast to the previous two network models, we can estimate the critical path concurrency directly without estimating YAWNS concurrency first.

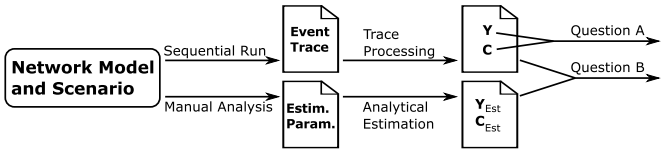


Figure 13: Validation of analytical concurrency estimations with reference to trace-based results.

6. EVALUATION

In this section, two questions are addressed:

Question A: Are the concurrency values determined by YAWNS-based analysis and critical path analysis sufficiently close to use these methods interchangeably? Our analytical model estimates concurrency according to YAWNS. However, since critical path analysis determines the largest possible concurrency, it must be considered the reference for concurrency estimation.

Question B: Does our analytical model estimate critical path concurrency of the considered network models with sufficient accuracy? A correspondence between the estimation and the critical path analysis results is an indication that our network model analysis captured the key influencing factors for the models' concurrency, clarifying the relationship between properties of the simulated networks and the resulting concurrency.

Figure 13 depicts the general work flow for validation of the concurrency estimation results for the parameterizations of each network model. Based on knowledge of the network model and, if necessary, on a sequential simulation run, the inputs required by the generic analytical model are determined. Event traces generated during the sequential run are processed programmatically using YAWNS-based analysis and critical path analysis. To answer question A, the results of the YAWNS-based analysis and critical path analysis are compared. To answer question B, the analytical estimate is compared to the critical path analysis result.

We first consider question A and focus on the results for the Kademlia and TCP/IP models, as the concurrency of the wireless network model was estimated directly with reference to critical path analysis. The parameters of the Kademlia model were varied as follows:

- $n \in \{1{,}000; 10{,}000; 100{,}000\}$
- $\lambda \in \{100; 1{,}000; 10{,}000\}$
- $\alpha \in \{1; 2; 4; 8; 16; 32\}$

In addition, we configured the probability of packet loss as 0%, 25%, 50% and 75%.

The TCP/IP model was configured as follows:

- Number of CNs $\in \{2; 4; 8; 16; 32\}$
- Number of nodes per LAN $\in \{2; 4; 8; 16; 32\}$
- Bottleneck bandwidth $\in \{1; 10; 100; 1{,}000\}$ Mbps

Figures 14 and 15 compare the results of YAWNS-based analysis to critical path analysis results. In both cases, YAWNS-based analysis determines lower concurrency values than critical path analysis. The deviation increases slightly with larger concurrency. However, even for very large concurrency values, YAWNS determines concurrency values between a factor of 0.6 and 1 of critical path analysis. We consider the correspondence sufficiently close to evaluate the parallelization potential of network models.

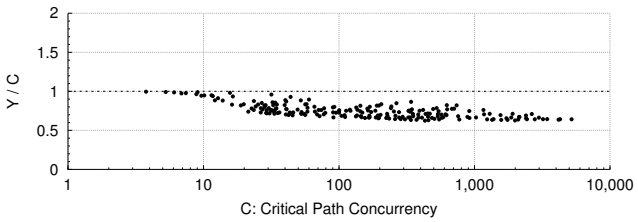


Figure 14: Comparison of YAWNS concurrency (Y) with critical path concurrency (C) of the Kademlia model.

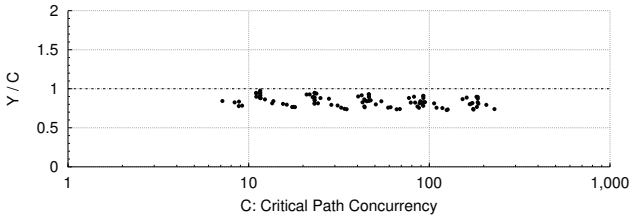


Figure 15: Comparison of YAWNS (Y) with critical path concurrency (C) of the TCP/IP model.

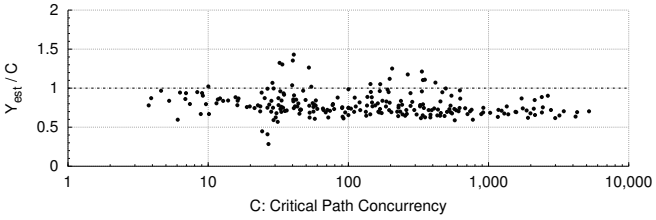


Figure 16: Comparison of our analytical estimate (Y_{est}) with critical path concurrency (C) of the Kademlia model.

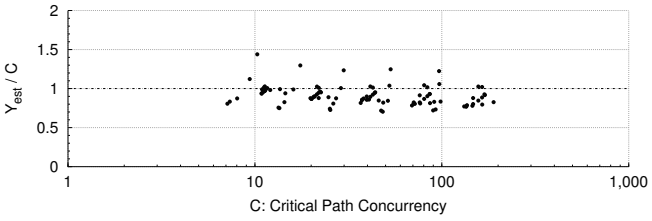
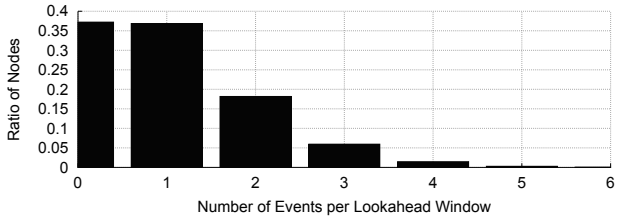


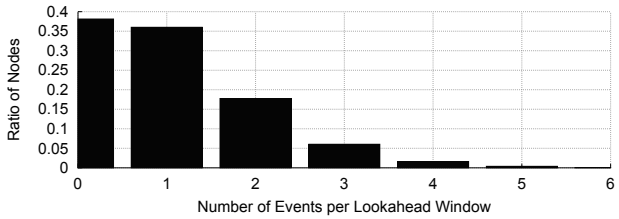
Figure 17: Comparison of our analytical estimate (Y_{est}) with critical path concurrency (C) of the TCP/IP model.

Now, we address question B and compare the analytical estimate with the critical path analysis results. Figure 16 shows the results for the Kademlia network model. As in the comparison of YAWNS and critical path analysis, a slight underestimation between the analytical model and critical path analysis can be observed in many cases. However, the model captures critical path analysis sufficiently so that, apart from few outliers, the model determines a factor between 0.5 and 1.5 of the reference value over a vast range of model parameters and resulting concurrency values.

Similarly, the results for the TCP/IP model depicted in Figure 17 show a close correspondence between the analyt-

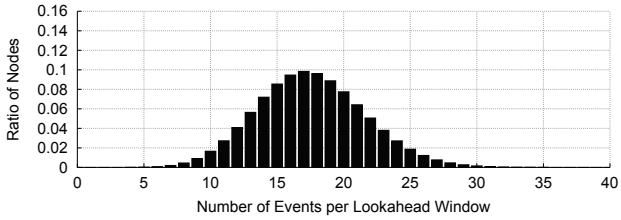


(a) Binomial distribution B(296, 1/300)

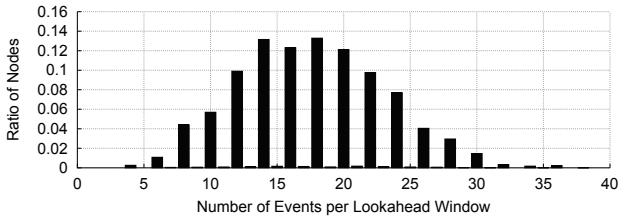


(b) Results from simulation run.

Figure 18: Expected and observed distribution of the number of events per node in the *active* category in each lookahead window for a run of the Kademlia model with $n = 1,000$, $\lambda = 300$, $\alpha = 8$, and 0% packet loss.



(a) Binomial distribution B(281, 1/16)



(b) Results from simulation run.

Figure 19: Expected and observed distribution of the number of events per node of the *hub* category in each lookahead window for a run of the TCP/IP model with 16 campus networks, 1 node per LAN, and 1Gbps of hub bandwidth.

ical estimate and critical path analysis results. Here, a repeating pattern emerges in the plotted results: our network model analysis assumed a full utilization of the channel capacity in the simulated network. With decreasing hub bandwidth, the simulated network deviates increasingly from full utilization, leading to an overestimation of concurrency.

The analytical approach used to estimate the concurrency of the Kademlia and TCP/IP network models makes the assumption of a uniform distribution of events within each lookahead window over the nodes of each of the identified categories. If the assumption of a uniform distribution of events to nodes holds, we expect a binomial distribution of the number of events assigned to each node in each lookahead window. Since this section already shows the validity of the concurrency estimations of our analytical model, we limit our illustration of the validity of our assumption of uniform distribution to two example scenarios. We determined the appropriate parameters for the binomial distribution according to the observed number of events per lookahead window, and the number of nodes in the considered node category. Figure 18 compares the expected binomial distribution with the number of events per node of the *active* category (cf. Section 5.1) actually observed in an exemplary simulation run of the Kademlia network model. We can see that in the considered scenario, the simulation results are matched closely by the binomial distribution. Figure 19 compares the expected binomial distribution with the number of events per node of the *hub* category (cf. Section 5.2) in an example run of the TCP/IP network model. Here, a deviation in the distributions is caused by the fact that in the network model, groups of two events each are scheduled with only a small delta in simulated time. Hence, in almost all cases, an even number of events is assigned to an individual node in each lookahead window.

For validation of the estimations for the wireless network model, we varied the number of nodes in the network between 2 and 100. Figure 20 relates the estimated concurrency C_{est} to the results from critical path analysis of event

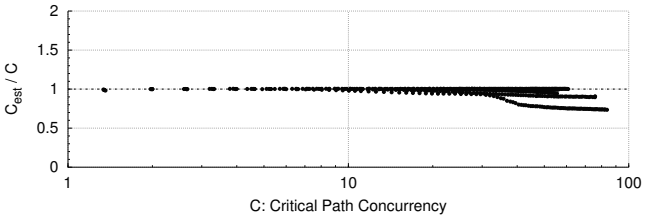


Figure 20: Comparison of our analytical estimate (C_{est}) with critical path concurrency (C) of the wireless model.

traces. For small networks, the estimation is nearly identical to the critical path results. The estimation becomes too pessimistic only in cases of extreme channel load, where collisions, which are not considered by the analytical estimate, are frequent. The largest deviation was measured in a scenario with 100 nodes and a beacon rate of 80Hz, where the estimation amounts to 73.5% of the concurrency determined using critical path analysis.

7. DISCUSSION

Our estimation approach requires the identification of simulation bottlenecks and a classification of nodes according to the number of assigned events. These requirements expose a fundamental challenge in performance estimation of simulations: simulation is typically applied when studying systems whose behavior cannot be easily modeled in a static form, such as by mathematical equations. If it is possible to model the runtime behavior of the model a priori, simulation is unnecessary. Hence, in case the aspects of the model's runtime behavior that are to be determined using simulation are also critical to the performance of the simulation, performance estimation is non-trivial.

Depending on the network model under study, it is possible to approximate the required statistics by performing brief sequential simulation runs. For instance, if the lookup duration is the desired metric when studying the Kademlia

model, yet the lookup duration is critical to the model's concurrency, a brief sequential simulation run can be performed to approximate the average lookup duration. In the cases considered here, such an estimation was sufficient to achieve a reasonable level of estimation accuracy.

8. CONCLUSION

We presented an analytical model to estimate and understand the concurrency of network simulation models. A sensitivity analysis and investigation of event patterns showed the factors determining the concurrency of three network models and the differences in their potential for parallelization. Given a modest amount of knowledge of the network model and information from sequential simulation runs, the analytical approach estimates concurrency with high accuracy over a large range of scenario parameter settings.

The key relationships between the properties of the considered network models and their concurrency are as follows: the concurrency of the considered peer-to-peer network model is strongly dependent on the number of nodes actively initiating traffic and limited by the total number of nodes, whereas the exact network topology has only little impact on concurrency. The peer-to-peer network model suggests the use of hundreds of cores for simulation. In the considered model of TCP/IP in a fixed topology, concurrency increases with larger numbers of hops between senders and receivers and, to a moderate degree, with larger bandwidth of bottleneck routers. The concurrency of a wireless network model was shown to scale in proportion to the number of nodes in the network, but is limited by the spatial extent of the scenario.

In future work, we aim to move towards a more general classification of network models by identifying recurring event patterns in models with similar topologies and communication patterns.

9. REFERENCES

[1] P. Andelfinger and H. Hartenstein. Exploiting the Parallelism of Large-Scale Application-Layer Networks by Adaptive GPU-based Simulation. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 3471–3482. IEEE, 2014.

[2] P. Andelfinger, K. Jünemann, and H. Hartenstein. Parallelism Potentials in Distributed Simulations of Kademlia-based Peer-to-Peer Networks. In *Proceedings of the Conference on Simulation Tools and Techniques (SIMUTools)*, pages 41–50. ICST, 2014.

[3] B. C. Arnold, N. Balakrishnan, and H. N. Nagaraja. *A First Course in Order Statistics*, volume 54. Siam, 1992.

[4] O. Berry and D. Jefferson. Critical Path Analysis of Distributed Simulation. In *Proceedings of the 1985 SCS Multiconference on Distributed Simulation*, pages 57–60. SCS, 1985.

[5] G. Kunz, D. Schemmel, J. Gross, and K. Wehrle. Multi-Level Parallelism for Time- and Cost-Efficient Parallel Discrete Event Simulation on GPUs. In *Proceedings of the 26th Workshop on Principles of Advanced and Distributed Simulation*, pages 23–32. IEEE Computer Society, 2012.

[6] J. Liu, Y. Li, and Y. He. A Large-Scale Real-Time Network Simulation Study Using PRIME. In *Proceedings of the 2009 Winter Simulation Conference (WSC)*, pages 797–806. IEEE Press, Dec 2009.

[7] J. Liu, D. Nicol, B. J. Premore, and A. L. Poplawski. Performance Prediction of a Parallel Simulator. In *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS)*, pages 156–164. IEEE Computer Society, 1999.

[8] 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.

[9] M. Livny. A Study of Parallelism in Distributed Simulation. In *Proceedings of the 1985 SCS Multiconference on Distributed Simulation*, pages 94–98. SCS, 1985.

[10] A. Loewenstern. BitTorrent Enhancement Proposal 5: DHT Protocol. `http://www.bittorrent.org/beps/bep_0005.html`, 2008. [Online; accessed 05-05-2015].

[11] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[12] 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.

[13] D. M. Nicol. The Cost of Conservative Synchronization in Parallel Discrete Event Simulations. *Journal of the ACM (JACM)*, 40(2):304–333, 1993.

[14] J. Pelkey and G. Riley. Distributed Simulation with MPI in ns-3. In *Proceedings of the Conference on Simulation Tools and Techniques (SIMUTools)*, pages 410–414. ICST, 2011.

[15] K. S. Perumalla. Discrete-Event Execution Alternatives on General Purpose Graphical Processing Units (GPGPUs). In *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, pages 74–81. IEEE Computer Society, 2006.

[16] P. Peschlow, A. Voss, and P. Martini. Good News for Parallel Wireless Network Simulations. In *Proceedings of the 12th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 134–142. ACM, 2009.

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

[18] C. Roth, S. Reder, G. Erdogan, O. Sander, G. Almeida, H. Bucher, and J. Becker. Asynchronous Parallel MPSoC Simulation on the Single-Chip Cloud Computer. In *2012 International Symposium on System on Chip (SoC)*, pages 1–8. IEEE, Oct 2012.

[19] B. P. Swenson, J. S. Ivey, and G. F. Riley. Performance of Conservative Synchronization Methods for Complex Interconnected Campus Networks in ns-3. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 3096–3106. IEEE, 2014.

[20] A. Varga, Y. A. Sekercioglu, and G. K. Egan. A Practical Efficiency Criterion for the Null Message Algorithm. In *ESS 2003: 15th European Simulation Symposium*. SCS, 2003.