Simulation of Adversarial Scenarios in OMNeT++ – Putting Adversarial Queueing Theory from Its Head to Feet

Daniel Berger
Distributed Computer Systems Lab (DISCO)
University of Kaiserslautern, Germany
berger@cs.uni-kl.de

Martin Karsten
David R. Cheriton School of Computer Science
University of Waterloo, Canada
mkarsten@uwaterloo.ca

Jens Schmitt
Distributed Computer Systems Lab (DISCO)
University of Kaiserslautern, Germany
jschmitt@cs.uni-kl.de

ABSTRACT

Adversarial models of traffic generation replace probabilistic assumptions by considering the deterministic worst-case. The framework of adversarial queueing theory (AQT) has discovered unexpected results on the stability of networks and has seen continuous research efforts over more than 15 years. So far, almost all AQT results have been derived analytically under simplifying but arguably harmless assumptions. However, as can be observed from recent work in AQT, the adversarial scenarios, in particular those that demonstrate instability, become more and more contrived and complex, thus lending themselves less and less to analytical tractability. While simulation seems like a good match for this problem, no available simulation model includes adversarial traffic generation. In this work, we introduce an OMNeT++ simulation framework for AQT as a tool to facilitate the study and development of instability examples. We validate the usefulness of AQT simulations in several use cases and, en-passant, discover some new insights into adversarial effects.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; I.6.m [**Simulation and Modeling**]: Miscellaneous

Keywords

Adversarial queueing theory, discrete-event simulation, FIFO scheduling, OMNeT++, stability, traffic generation

1. INTRODUCTION

For network performance evaluation the term *stability* has emerged to describe the desired state of a network that is able to cope satisfactorily with the load requested. For example, in queueing theory, a networking system is termed unstable if no equilibrium exists, i.e., if queues might grow indefinitely [7].

Achieving stability over long time-scales is non-trivial and has been in the scope of research ever since the beginning of telecommunication network models. When early work on queueing delay by Erlang [10] was expanded to networks of queues [13] and networks with diverse customers [14], the initial belief was shaped that instability might only happen in (locally) overloaded networks. Yet, in the early 1990s, probabilistic examples [17, 19] have shown network scenarios that are unstable despite the queues not being overloaded locally by the input traffic.

Starting from this insight, novel queueing models have emerged, among them *Adversarial Queueing Theory* (AQT) [5], which is centered around the notion of stability. While previous work usually assumes stochastic processes for packet arrivals and service times (in most cases of Markovian nature), this model introduces a radically new approach to study stability. In an attempt to dispense with specific probabilistic assumptions, these are replaced by deterministic worst-case considerations: the adversarial traffic model assumes that arrivals are controlled by a hypothetical adversary, whose goal it is to put stress on the networking system and to eventually destabilize it. Indeed, this line of thinking proves to be a rich source of fundamental insights into network stability and has delivered a number of surprising results. In the following, we restrict our brief survey to the practically very relevant case of FIFO scheduling.

Andrews et al. [2] were the first to prove that a small four-node network, known as the baseball network, can be unstable even though the arrival rate of new packets requiring any component in the network is less than each component's processing capacity, i.e., in the classical sense an underloaded system. The corresponding example is based on an inductive construction. By assuming a certain queue length and a specific pattern of traffic flows, such that, after some time, the assumptions for the same process are satisfied again, yet now at a larger queue length, this provides a recipe to destabilize the network in terms of infinitely growing buffers.

Interestingly, bursty packet arrivals are not inherently sufficient to cause instability [12] and all instability examples work with smooth rates and by exploiting the network's topology. Hence, rate and topology are central in the process of causing instability and critical to understand the exact boundaries of stable network behavior.

The lower bound on the arrival rate of unstable networks has been lowered in subsequent work [9, 15, 18] and ultimately, it has been shown that networks with FIFO scheduling can be unstable at arbitrary small $(0+\epsilon)$ arrival rates [3].

Similarly, the space of unstable topologies has been ex-

OMNeT++ 2013 March 5, Cannes, France.

plored by graph minors [1, 2, 11]. A comprehensive characterization has determined that only so-called decorated cycles are FIFO-stable topologies [23]. However, many realistic topologies are not captured by this characterization and thus their stability is not guaranteed.

In summary, interesting results have been obtained from adversarial queueing models but are currently limited to a theoretical viewpoint. A more practical approach and broader understanding is an important step to make use of these results. To this end, we propose to simulate instability effects and introduce a suitable model for the OMNeT++ discrete simulation framework [22]. Using simulations provides the flexibility to relax certain "convenient" assumptions from the analysis, evaluate the tightness of bounds, and study the robustness of adversarial effects.

1.1 Problem Statement and Use Cases

Previous work on adversarial queueing models studies stability with analytical methods. This works well, e.g., when deriving delay bounds for stable networks, or to describe small instability examples [2, 9, 15], but becomes increasingly complex for involved constructions with low adversarial arrival rates [18, 3]. For instance, for the latter two AQT scenarios only bounds of unclear tightness are known. This may be sufficient for the qualitative evaluation whether a system is stable or not. However, we are also interested in quantitative measures of instability such as the severity of an instability scenario, the time until a given measure of congestion occurs, or the amount of loss under finite queues.

To summarize, for quantitative evaluation, we observe that tight bounds on the behavior of networking systems in adversarial queueing models are hard to derive. This is a major motivation for an AQT simulator, because such a tool can be used to efficiently obtain numerical results in this domain. Additionally, we find that developing new adversary types can be supported very well with a simulation tool at hand. In particular, the possibility to visualize the system state at any given time is an important aid in designing sophisticated adversaries.

As pointed out already, many assumptions of adversarial queueing models are not accurate and are made for analytical convenience. For example, the models use discrete time steps where events occur at distant nodes in the network with precise and synchronous timing. Of course, this does not necessarily hold in real-world networks. A flexible simulation model allows an increased level of detail and realism to iteratively adapt an adversary's strategy to observations from simulation experiments.

1.2 Related Work

To our knowledge, there is little work on AQT simulation, and what exists only touches the issue while focusing on other aspects. Chroni et al. [8] use a simulation of adversarial queueing theory to determine the stability of compositions of different schedulers in a network. The scope of this approach is fundamentally different from our goal as it focuses on the exact conditions for stability. In contrast, the main focus of our work is to study and understand instability. We wish to observe the theoretically predicted unstable behavior in order to understand the corresponding implications for a real-world network. We focus on a quantitative assessment of FIFO instability under slightly more realistic assumption than the original model. The simulation in [8] focuses on a variety of different scheduling strategies and composite schedulers and evaluates only qualitatively whether instability occurs.

Also, a distinguishing feature of this work is that we validate our implementation by a comparison to previous analytical results and make the complete source code available to the public. We facilitate the implementation of new adversaries and reuse of existing ones by a modular code-excerpt structure and are able to visualize network behavior through the capabilities of OMNeT++

Another related approach is concerned with more complex inter-arrival distributions for traffic generation [16]. By implementing several stochastic process types as modules for OMNeT++, the authors are able to simulate positive correlation between packet arrivals. An adversarial traffic injection model is different as inter-arrivals are entirely deterministic. Furthermore, adversarial queueing theory is mainly concerned with circular topologies where the adversary also controls the routes taken by packets. From this perspective, our implementation goes beyond traffic generation and extends into dynamic continuous packet routing [5].

Because of the deterministic nature of AQT, the initial simulation model does not include stochastic variation. The output of the model strictly depends on the set of input parameters and behaves the same for every run. This approach can be considered as being in the realm of the general topic of deterministic simulation models, which see limited use in computer networking, but are regularly used in other science disciplines [20]. However, we use this method differently than most previous work in this domain as we take the opposite view point: the model is given and corresponding analytical predictions are studied.

1.3 Outline

We report adaptations to AQT to fit an implementation in OMNeT++ in Section 2. The implementation and design decisions are described in Section 3. Experimental results for the validation of the usefulness of AQT simulations are presented in Section 4, and we conclude in Section 5.

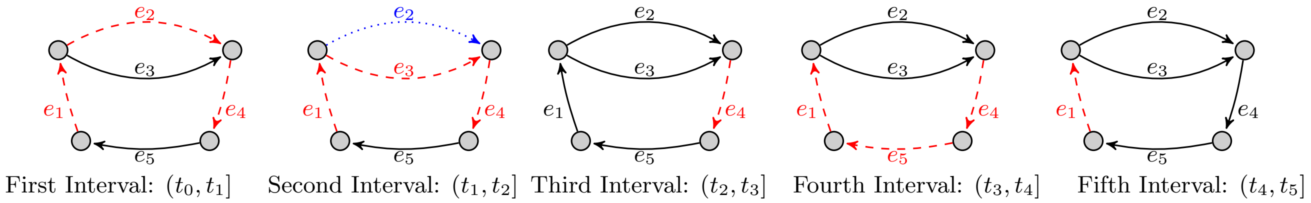
2. SIMULATION MODEL

The underlying model for the simulation presented here is very similar to the original specification of AQT [5]. We first introduce the original model, present an example, and then describe adaptations to AQT, and design decisions we took for our implementation.

2.1 Adversarial Queueing Model

A network is represented as a directed graph $\mathcal{G} = (V, E)$ where V are network nodes and E are network links. Packet routes are assumed to be simple paths. Edges comprise a queue at their origin where packets wait if they arrive at an edge which is already processing requests. At a non-empty queue the next packet to be processed is chosen by some scheduling policy which will be FIFO throughout this work. If two packets arrive at the same time an arbitrary tie-breaking rule is used.

We consider discrete time, where each edge may process one packet each time step. Processing means that the packet disappears from the queue and appears in the queue of the next edge in the packet's path if it has not yet reached its final destination. If it arrives at its final destination, it is assumed to be absorbed, thus leaving the system.



e_2
e_3
e_1
e_4
e_5
First Interval: $(t_0, t_1]$
Second Interval: $(t_1, t_2]$
Third Interval: $(t_2, t_3]$
Fourth Interval: $(t_3, t_4]$
Fifth Interval: $(t_4, t_5]$

Figure 1: An example of a small graph taken from the class $\mathcal{A}^+$ introduced in [23]. This class is one of three instability graph minors which are the foundation of more complex examples. A phase is divided into five intervals; in each interval injection's paths are marked by dotted or dashed lines.

Besides the network representation, the model comprises an adversary $\mathcal{A}$, giving name to the model. The adversary injects packets that follow predefined routes chosen by the adversary such that the stress on the network is maximized. All injections are subject to a load condition which ensures that the network is not trivially overloaded by the requests upon injection.

Different but similar load constraints have been used in previous work. We find the *bounded adversary* model [2] more intuitive than the original form [5]. As both adversary models exhibit the same power [21], we adhere to the former. Fix a natural number $b \geq 1$ and $0 < r < 1$; an adversary is said to be (b, r) bounded if, for any interval $[s, t)$, the adversary's injections containing any particular edge in the graph do not exceed $r(t-s)+b$ packets. This implements a simple leaky bucket constraint on the adversary's injections where b allows for some burstiness, and r is called the injection rate. Note, that the injection rate is strictly less than one, which in traditional settings would mean a (local) underload.

2.2 Example for the Original AQT Model

Proving a network to be unstable works in an inductive fashion: assuming a number of packets s somewhere in the network and the adversary applying some injections, it is shown that the network satisfies the initial assumption again, but the number of packets has increased to $s' > s$.

As an example we consider the graph in Figure 1. This graph and its adversary are examples for the class $\mathcal{A}^+$ introduced in [23] and the mechanism is representative of many other examples as it is one of three fundamental instability graph minors[1]. We will use the $\mathcal{A}^+$ graph minor as a running example in subsequent sections. The adversary is described in textual form in [23], but given in a condensed form here, with a visualization of injections in Figure 1.

ADVERSARY 1.
$\boldsymbol{t_0}$: *Assume an initial set S_0 of size s is queued in e_1.*
$\boldsymbol{(t_0, t_1 = t_0 + s]}$: *Inject the set S_1 of size rs into e_1 with path (e_1, e_2, e_4).*
$\boldsymbol{(t_1, t_2 = t_1 + rs]}$: *Inject the set S_2 of size $r^2 s$ into e_1 with path (e_1, e_3, e_4); inject the set C of size $\frac{r^2}{1+r}$ into e_2.*
$\boldsymbol{(t_2, t_3 = t_2 + r^2 s]}$: *Inject the set S_3 of size $r^3 s$ into e_4.*
$\boldsymbol{(t_3, t_4 = t_3 + r^2 s]}$: *Inject the set S_4 of size $r^4 s$ into e_4 with path (e_4, e_5, e_1).*
$\boldsymbol{(t_4, t_5 = t_4 + r^2 s]}$: *Inject the set S_5 of size $r^5 s$ into e_1.*

Observe that in $(t_0, t_1]$ S_1 stays queued behind the initial set S_0. In $(t_1, t_2]$ S_1 starts traversing its path, but is slowed down by the injections of C which compete with S_1-packets to traverse e_2. Meanwhile S_2 is still queued behind the remaining packets of S_1. In $(t_2, t_3]$ the adversary waits for the simultaneous arrival of S_1 and S_2 at e_4 and uses this time to further fill the queue of e_4 with packet injections. It remains to ensure that the initial assumption of S_0 is satisfied for the next phase. Appendix B.1 in [23] gives a proof showing that at the end of some phase, $s' = sr^6 \frac{2+r}{1+r}$ packets satisfy the assumption of S_0 for the next phase. Then, another phase may follow with the same injection and packet flow patterns. For $r > 0.933$ it holds that $s' > s$ and by induction the number of packets in the network grows without bounds.

2.3 Adaptation to the Simulation Model

We represent a network as an undirected graph as this holds for the majority of communication networks in practice. This has been done before in adversarial queueing models [2], but most instability examples use directed graphs. Instead of an explicit list of edges packets have to visit, we adapt a loose source routing approach, i.e., a list of nodes to visit and static shortest-path routing so that the node list does not need to include the full path. This means a change from a link addressing scheme to node addressing, which in turn changes the requirement of simple paths to requiring paths without cycles. We choose this adaption because routing in real-world networks is also based on node addresses instead of link addresses.

Because smooth arrivals are sufficient to cause instability, we drop the adversary parameter b as a parameter in our simulation. Instead we add the initial configuration of the network at time $t_0 = 0$ as parameter. An initial distribution of packets in the system is necessary for all previously published adversary types. There are proposals to transform adversaries with initial sets into adversaries with empty configurations [2, 4]; however, we are interested in the effect of initial set sizes on unmodified adversaries and topologies.

In previous work it has been consensus to "keep the definitions slightly informal for the sake of readability" [2]. Obviously, this does not work when implementing these definitions into a program. Hence some interpretation to formalize informal descriptions of adversaries in programming language statements is needed. A first step is the notation used in Adversary 1 and in Figure 1. Next, we formalize the notation by the use of OMNeT++ and C++ data structures.

2.4 Adapted Example for Simulation Model

In this section, we review the adaptations of AQT for the instability graph minor $\mathcal{A}^+$ as introduced above.

Representing the graph as an undirected graph with node-level addressing makes it impossible to route packets either

[1] A graph is FIFO-stable, iff none of the classes $\mathcal{A}^+$, $\mathcal{B}^+$, $\mathcal{C}^+$ introduced in [23] appears in it as a graph minor.

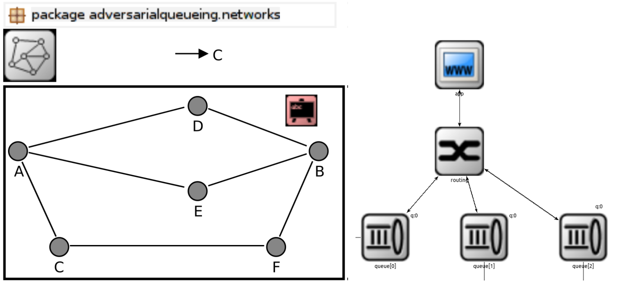


Figure 2: Left: Implementation of the $\mathcal{A}^+$ graph minor as an OMNeT++ network. Nodes are used for addressing. Right: Node A from the same network is an instance of the class SourceRoutingNode. It comprises one application, one routing module, and a queue for each link to the neighbors C, D, and E.

over e_2 or e_3 as in the original graph Figure 1. For this reason we introduce an additional helper node whenever switching from edge addressing to node addressing causes ambiguity. We show the result of adapting the $\mathcal{A}^+$ graph minor in Figure 2. D and E are examples of artificial helper nodes.

To formalize the description of Adversary 1 we need to distinguish between two different notions of time: simulation time (SimTime) in OMNeT++, and discrete time steps in AQT. We use an overlay over SimTime and represent time steps in SimTime by multiplying with the configuration parameter “timeSlots”. Injection intervals are described in AQT time steps by setting an interval start and the interval length. Corresponding injections are executed for a specified number of time steps at the injection rate r. These intervals are always rounded to integer time steps. In contrast, the injection rate r is represented by a waiting time of $1/r$ between injections – this is not rounded but uses SimTime precision. This can also be seen in Listing 1 where we consider the second interval $[t_1, t_2]$ of Adversary 1. Interval start and length are defined in the first two lines. This interval starts directly after $[t_0, t_1]$ is over, hence we add its length. The length of $[t_1, t_2]$ is the injection rate r multiplied with the previous interval's length. Injections are scheduled with an inter-packet delay of $1/r$ as specified in line 5 and the total number of injected packets is given in line 6. The rounding methods presented here are conservative as we choose a perspective on instability [5].

Injection paths are given to the simulator in an explicit order. Similar to the OMNeT++ routing example, nodes are labeled by character strings but addressing is done on integer addresses. This can be seen in Listing 1 in Lines 10-14 where a path of length 4 is specified. Node address 1 is labeled A, 22 is D, 3 is B, and 4 is F in Figure 2.

3. IMPLEMENTATION IN OMNET++

We have used OMNeT++ 4.2 to implement our AQT simulation model, and have used code and concepts from the routing example. The simulator's project page including the source code of the model is available online[2]. On the highest abstraction level, we implement topologies proposed in the AQT literature as NED files comprising a corresponding adversary class, and node modules which allow for source routing and may be controlled by the adversary. Nodes are linked by either standard data rate or delay channels, or by a channel with variable delay which we use to test the robustness of adversarial effects (see Section 4). Apart from the latter channel type, everything behaves in a deterministic way. Because AQT requires discrete time steps, the implementation uses an abstract time step structure on top of the virtual continuous time which is offered by OMNeT's SimTime Class. The basic 64-bit Integer type cannot be directly used to represent time steps, because variable channels may introduce continuous delay variation. Furthermore, to be more realistic, we allow injections to take place in continuous time (according to the injection rate). One time step has a 1ms simulation time duration. All operations that are not concerned with link traversals, e.g., routing decisions, do not consume (simulation) time.

```
 1  //interval [t1,t2]
 2  intervalStart +=
         intervalLength*(timeSlots->doubleValue());
 3  intervalLength =
         ceil(intervalLength*injectionRate);
 4  newInjection = new AdvSchedMess;
 5  newInjection->interInjectionTime =
         (timeSlots->doubleValue())/injectionRate;
 6  newInjection->packetCount =
         floor(intervalLength*injectionRate);
 7  newInjection->atNode = "C"; //where to inject
 8  newInjection->message = new
         AdversarialInjectionMessage("set S2");
 9  newInjection->message->setPathArraySize(4);
10  newInjection->message->setPath(0,4);//last hop
11  newInjection->message->setPath(1,3);
12  newInjection->message->setPath(2,22);
13  newInjection->message->setPath(3,1);//first hop
14  scheduleAt(intervalStart,newInjection);
```

Listing 1: Code-excerpt of the implementation of Adversary 1 in the proposed OMNeT++ package. Only the second interval is shown.

3.1 Nodes

Network nodes are implemented as compound modules as illustrated in Figure 2 Each link is connected by an InOutgate to a slightly modified version of L2Queue, which in turn is connected to the routing layer. A packet's destination address is an array of nodes to visit, in reverse order. Each packet traversing a node is forwarded to the routing module which considers the last entry of the address array. If the array has a length greater than one, then the packet is forwarded, and a static shortest-path scheme is used to determine the output link. The address array length is decreased by one, if the current node is an intermediate hop.

If the array has length one and the local address matches the array's entry, then the packet is forwarded to the application layer, which reports the delay and absorbs the packets.

3.2 Querying the Network State

All previously published adversaries assume a set of packets s in a queue in the network, and schedule injections corresponding to $|s|$. We implement this behavior by giving the adversary the possibility to query for any queue's

[2] http://disco.informatik.uni-kl.de/content/Aqtmodel

Message Type	Description	Representative Properties
QueueLengthRequest	Out-of-band control command, used by the adversary to set up a queue listener	moduleName (node), outAddress (set by adversary), gateID (set by routing layer)
AdversarialSchedulingMessage	Internal scheduling for the adversary's injections, includes a whole set of injections.	packetCount, atNode, interInjectionTime, AdversarialInjectionMessage
AdversarialInjectionMessage	Out-of-band injection command, sent from adversary to application layer of a node.	destAddrArray, injection description, (atNode implicit by sendDirect)
SourceRoutingPacket	Basic packet type which is sent over the network, the path is modified at each hop in destination path	srcAddr, destAddrArray, hopCount, recordRouteArray

Table 1: Different message types used in the implementation. The first three message types are used for injections by the adversary, the last one describes the format of packets sent over the network.

length. In fact, using the analytical model, an adversary would be able to compute the cardinality of s for any phase in advance. However, this would disqualify any comparison between analytical and simulative results: the analytical prediction might estimate a queue's size wrong and the adversary may carry this offset as an additive error across all subsequent phases.

This additional capability of the adversary, which is implicit in analytical models, is implemented as a listener on a queue's length. For modularity, the adversary does not have direct access to the queues but sends a request to a particular node to set up a listener for a link's queue at this node. Having a layered node architecture, it would be best to reuse the already established control path for injection commands from the adversary to the application layer. However, only the routing layer knows the correspondence between links and queues. To simplify implementation, we allow the adversary to also communicate with the routing layer and thereby obtain knowledge of queues to set up a queue length listener.

3.3 Adversaries and Packet Injections

Adversaries share common functionality. In particular, all adversaries need to translate the definition of injections as introduced in Section 2.4 into discrete packets. This process is implemented in the super-class *AdvancedAdversary* from which adversary implementations are derived. Deriving a new adversary works by providing a class inheriting from AdvancedAdversary, defining a function to set up the initial state of the network, and defining a method to handle the injection scheme. The injection scheme is specific to each adversary and is executed repeatedly, at each phase's start: the adversary queries the network state and schedules sets of injections. An adversary such as Adversary 1 would consist of several blocks of injections, each one similar to Listing 1 and is organized in several stages. Some key message types are explained in Table 1.

The initial state of the network is only constructed once, at time step 0. Dependent on the configuration parameter *InitialSetSize* an initial configuration of packets in queues is prepared. This may include packets at multiple queues in the network, however, each queue's size must be strictly less than the initial set size. Additionally, adversaries may set up queue listeners with *QueueLengthRequest*-packets to nodes in the network. Doing this once gives the possibility to learn a queue's length at the start of a new phase.

The injection scheme consists of four steps and is executed once every phase. Firstly, the adversary queries some queues' lengths. Depending on this information, injection sets are created and represented as messages of the type *AdversarialSchedulingMessage*. These messages are scheduled with waiting time $1/r$ as a self message and are handled by the super-class AdvancedAdversary. Upon receiving an AdversarialSchedulingMessage, the AdvancedAdversary derives injection commands for single packets from this representation. These injection commands are then sent out-of-band as *AdversarialInjectionMessage* to the application layer of a network node. Each such message triggers the immediate injection of a *SourceRoutingPacket* in the application layer which is passed on to the routing layer, and handled according to the description in Section 3.1.

3.4 Channels

Previous work in the adversarial queueing framework has modeled changes, e.g., to a link's capacity, by allowing the adversary to change a link's state every time step [6]. Our goal is different, since we want to assess the robustness of adversarial effects. Therefore we introduce randomized channels with the capacity modeled by a random variable.

We have evaluated two implementations of randomization. The first implementation allows for a change in a channel's capacity in each time step. Although this would be a typical approach from the classical AQT viewpoint, this model is debatable: for a real-valued random variable the duration of traversing a link might not match our time-step abstraction. In particular, it might happen that two consecutive packets observe an unchanged channel capacity. The second implementation extends the basic DatarateChannel so that the channel's capacity is an independent random variable for each traversal.

In both implementations, we sample each link's capacity from a Normal distribution. To work around the issue of negative channel capacity, the truncated Normal distribution is used. The OMNeT truncnormal implementation rejects negative values by sampling again until a positive value is obtained. We chose the mean capacity so that it translates into a one time step duration for our uniform packet size, and we introduce the standard deviation as an additional floating-point parameter.

3.5 Summary

Our proposed OMNeT++ simulation model has six basic parameters which can be set in the main configuration file:

1. the topology implemented as a NED-file;
2. the adversary which a user may provide as a C++ class or reuse from existing examples;
3. the initial set size which sets a maximum allowable queue size for a network's initial configuration

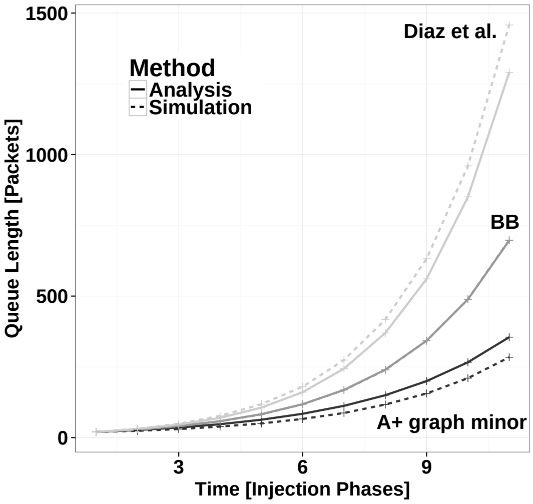


Figure 3: Number of packets in the network at the end of phase x: comparison between analytical prediction and simulative result for three adversaries.

4. the injection rate r;
5. the mean of the variable channel delay;
6. the standard deviation for the variable channel delay.

4. VALIDATION AND RESULTS

The goal of this section is to demonstrate the usefulness of AQT simulations as a complementary method to analysis when investigating instability effects of packet-switched networks. To that end, we provide a non-exhaustive set of representative use cases where simulations deliver insights that go beyond what is feasible in analysis:

- relaxation of analytically convenient assumptions;
- investigation of the effect and interaction of initial set sizes and injection rates;
- evaluation of the dependence of adversarial scenarios on strict synchronization;
- assessment of the tightness of analytical bounds.

We find some surprising results that are interesting in themselves, for example on the interaction between the size of initial sets and the injection rate and that strong desynchronization by randomization can destruct adversarial effects. In most experiments we deal with the following three prominent adversarial scenarios: Baseball (BB) [2], Diaz et al.[9], and the previously introduced graph minor $\mathcal{A}^+$ [23]. We use $r = 0.96$ (if r is not a parameter), an initial set size of 20 (if not a parameter), a buffer size which is never exceeded, and evaluate 60,000 time steps with OMNeT++ 4.2.2.

4.1 Basic Comparison to Analytical Results

The first set of experiments is focused on the relaxation of analytically convenient assumptions such as the continuation of discrete events, e.g., working with a fraction of packets in the analysis. Additionally, this comparison between analysis and simulations of the adversarial scenarios serves to some degree as a validation of our simulation model.

Figure 3 shows the comparison of analytical vs. simulative results for the queue size evolution over time for the different adversarial scenarios under study. While for all scenarios analysis and simulations stay fairly close, it is not always a perfect fit. This demonstrates that, while qualitatively instability always occurs, the convenient relaxations can have an effect on quantitative characteristics. Interestingly this differs for the different adversarial scenarios: BB is a perfect fit, for the $\mathcal{A}^+$ graph minor analysis overestimates, and for Diaz analysis underestimates the adversarial effect. It also provides evidence that these relaxations have to be done with some care as they are not easy to control in terms of providing upper or lower bounds always.

Note that in Figures 3 and 7 the x-axis is labeled *phases*, because analytical methods yield estimates for the results at the end of a phase, but Figures 5 and 6 show time steps to give a more realistic temporal point of view.

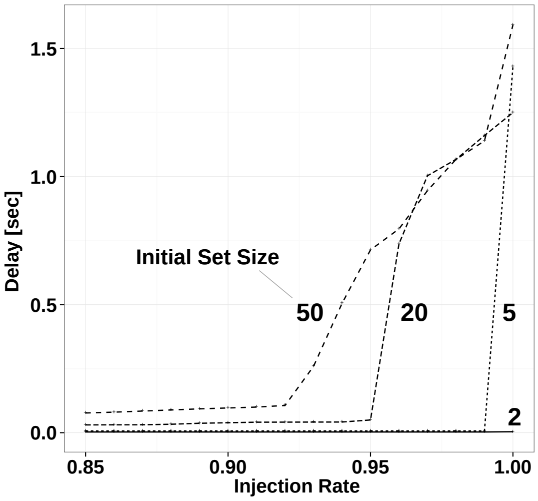


Figure 4: Injection rate vs maximum delay with different sizes of initial sets. This shows that interaction between the two parameters is possible and that networks with small initial sets may remain stable although the injection rate exceeds the theoretic instability threshold $r > 0.93$.

4.2 Interplay of Initial Sets and Injection Rate

Next, we investigate the effect of initial set sizes and injection rates as well as their interaction. Especially, the effect of initial set sizes and its interplay with injection rates has rarely been addressed in AQT literature, presumably because it is difficult to do analytically. Here, we restrict the study to the $\mathcal{A}^+$ graph minor scenario for brevity.

In Figure 4, we can observe the effect on the maximum observable delay (over a finite time horizon of 10000 time steps) for different initial set sizes as the injection rates grow. Note that in the simulations we can now easily use the delay in a metric such as seconds as a measure of interest, something that would not be simple, using analysis. Interestingly, we can observe that for a very low initial set size of 2 no significant delay can be observed. It appears that the adversarial effect simply does not show up even under high injection rates close to 1. This changes for an initial set size of 5 but requires high injection rates. For initial set sizes of 20 and 50, delay becomes excessive already at lower injection rates, clearly showing the interaction of these two parameters.

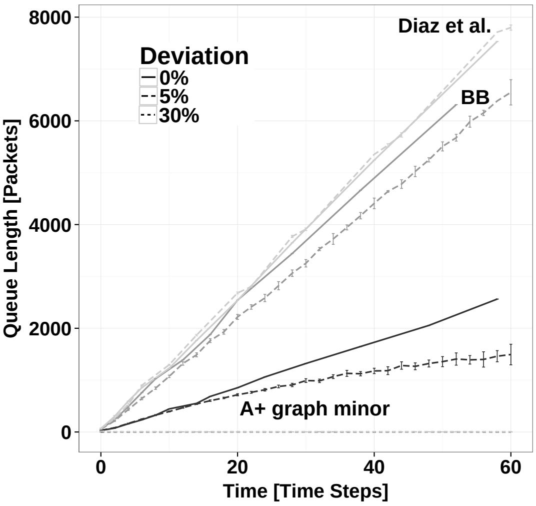


Figure 5: Effect of randomized channel delay on adversarial effect: mean queue length at the end of a phase with 30 repetitions. The queue lengths grow steadily over time[4]for a deterministic system and small variation of channel delay, e.g., 5% deviation of a time step. For a highly variable channel delay, the adversarial effect disappears.

4.3 Channel (Re-)Randomization

It is considered a virtue of AQT to not rely on stochastic assumptions to assess network stability. On the other hand, it is clear that for the adversarial effects to appear, the deterministic synchronization of events plays a role. In this subsection, we use the flexibility of our simulation model to investigate how crucial the role of synchronization is in adversarial scenarios. We (re-)randomize the channel delay to some degree, which is also a reality check for AQT as perfect synchronization is typically unrealistic. Specifically, we consider a channel whose traversal time is modeled as a normally distributed random variable with mean one time step and a standard deviation of a certain percentage of the unit delay. Both implementations of randomized channels have been used to derive the results. We find the results very similar, and report only those obtained from the second implementation (Section 3.4). Note that a standard deviation of 0% denotes the original deterministic setting. Because these experiments are not deterministic anymore, 30 repetitions of each experiments are performed.

Figure 5 shows the effect of randomized channel delays on the evolution of the queue size over time for the different adversarial scenarios. For a strongly randomized channel (30% standard deviation), all scenarios become harmless and the adversarial effect is not exhibited any more. For a moderately randomized channel (5% standard deviation), adversarial effects still show up for all three scenarios, more or less in a quantitatively attenuated form. A more detailed view of the queue size process over time for the deterministic and strongly randomized channel for the $\mathcal{A}^+$ graph minor can be found in Figure 6 and provides deeper insight into why the adversarial effect disappears.

Thus, by using simulations, we are able to shed some light on the issue of the robustness of adversarial effects against desynchronization effects. Clearly, this is still a preliminary study of this important aspect, but serves as a further illustration of the usefulness of AQT simulations.

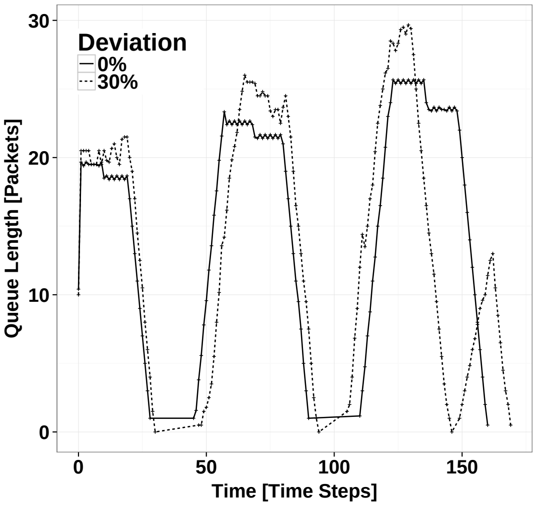


Figure 6: Detailed observations of the queue of the link C–A in the $\mathcal{A}^+$ graph minor. We compare the case of 30% delay deviation to zero variation. For zero variation we observe symmetric behavior, e.g., for the shape of the burst pattern, or the time between two bursts. Irregular behavior occurs for 30% deviation: the first two bursts are farther spread out than the second to third, and the queue length overshoots occasionally. After time step 150 the next burst has a much smaller size than previous bursts.

4.4 Tightness of Analytical Bounds

Finally, we study the use of simulation to assess the tightness of analytically derived bounds. This is very important as in many recent adversarial scenarios the complexity of the adversaries has grown to a level where analysis often resorts to bounds instead of exact results. The first published example for this is the adversary by Lotker et al. [18]. The network topology consists of a variable number M of building blocks called gadgets. Each gadget has a length n and consists of two node chains. Lemma 3.6 and Theorem 3.17 in [18] give lower bounds on how to choose these parameters. In order to simulate as tight as possible, we choose minimal values. For the results presented here, we have used an initial set of 2965 packets, $n = 10$, and $M = 20$ – a network of about 400 nodes. From Lemma 3.15 in [18] we obtain an analytical bound on the queue size after each (sub-)phase. We compare this analytical bound against simulations of the same scenario in Figure 7.

Clearly, the analytical lower bound, while correct, is very conservative and may not even capture the right order of growth. This is strong evidence that an accurate quantitative assessment of network instability in more complex adversarial scenarios may require simulations.

5. CONCLUSION

In this work, we introduce a novel OMNeT++ package which allows to quantitatively assess adversarial instability effects for a variety of adversaries and topologies. We validate the usefulness of simulations in the adversarial queueing

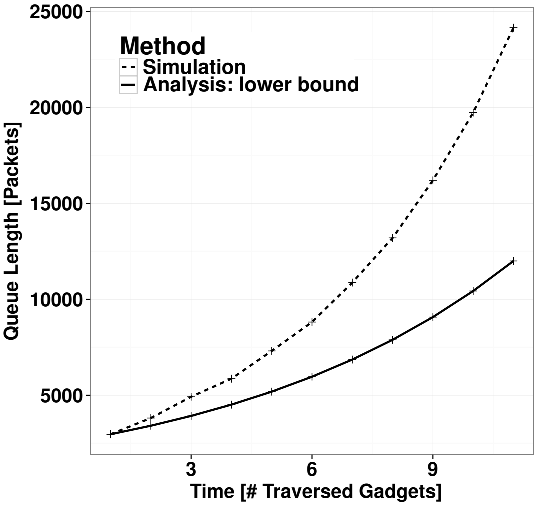


Figure 7: The authors who introduce the first parameterized AQT scenario in [18] derive only a lower bound on the queue length to prove instability. By simulation, we can observe that the adversary is more efficient than analytically anticipated. Already after traversing 11 of 20 gadgets we observe a difference of factor two.

theory concept in several use cases and demonstrate its potential to create a new perspective on adversarial queueing scenarios. For example, an interaction between initial set size and injection rate has not been demonstrated before, as well as investigating the tightness of the bound in [18] shows that quantitative results obtained from simulations can be much more precise.

We are confident that bringing this theory closer to practice bears many more interesting insights and challenges. Implementing a simulation framework is only a first enabling step. Clearly, we are aware that the simulation framework presented here still suffers from many simplifying assumptions about real networks. Nevertheless, we believe it to be a valuable tool to study the important issue of (in)stability. In particular, understanding the interaction of adversarial effects with a layered architecture, e.g., congestion control, is the ultimate goal towards assessing the realistic threat of adversarial effects.

6. REFERENCES

[1] C. Alvarez, M. Blesa, and M. Serna. A Characterization of Universal Stability in the Adversarial Queuing Model. *SIAM Journal on Computing*, 34(1):41, 2004.

[2] M. Andrews, B. Awerbuch, A. Fernández, T. Leighton, Z. Liu, and J. Kleinberg. Universal-stability results and performance bounds for greedy contention-resolution protocols. *Journal of the ACM*, 48(1):39–69, Jan. 2001.

[3] R. Bhattacharjee and A. Goel. Instability of FIFO at Arbitrarily Low Rates in the Adversarial Queueing Model. *SIAM Journal on Computing*, 34(2):318, 2005.

[4] M. J. Blesa. *Stability in communication networks under adversarial models.* PhD thesis, Universitat Politècnica de Catalunya, 2005.

[5] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson. Adversarial queuing theory. *Journal of the ACM*, 48(1):13–38, Jan. 2001.

[6] A. Borodin, R. Ostrovsky, and Y. Rabani. Stability preserving transformations: Packet routing networks with edge capacities and speeds. In *Proc. of SODA'01*, volume 05, pages 1–12, Mar. 2004.

[7] M. Bramson. Instability of FIFO queueing networks. *The Annals of Applied Probability*, 4(2):414–431, May 1994.

[8] M. Chroni, D. Koukopoulos, and S. D. Nikolopoulos. An experimental study of stability in heterogeneous networks. In *Proc. of WEA'07*, pages 189–202, Rome, Italy, 2007. Springer-Verlag.

[9] J. Diaz, D. Koukopoulos, S. Nikoletseas, M. Serna, P. Spirakis, and D. M. Thilikos. Stability and non-stability of the FIFO protocol. In *Proc. of the SPAA '01*, pages 48–52, Greece, 2001. ACM.

[10] A. Erlang. The theory of probabilities and telephone conversations. *Nyt Tidsskrift for Matematik B*, 1909.

[11] A. Goel. Stability of networks and protocols in the adversarial queueing model for packet routing. *Networks*, 37(4):219–224, July 2001.

[12] B. Hajek. Large bursts do not cause instability. *IEEE Transactions on Automatic Control*, 45(1):116–118, 2000.

[13] J. Jackson. Networks of waiting lines. *Operations Research*, 5(4):518–521, 1957.

[14] F. Kelly. Networks of queues with customers of different types. *Journal of Applied Probability*, 12(3):542–554, 1975.

[15] D. Koukopoulos, M. Mavronicolas, and S. Nikoletseas. On the stability of compositions of universally stable, greedy contention-resolution protocols. *Proc. of DISC*, 14186:88–102, 2002.

[16] J. Kriege and P. Buchholz. Simulating Stochastic Processes with OMNeT++. *Proc. of SIMUTools '11*, pages 367—-374, 2011.

[17] P. R. Kumar. Re-entrant lines. *Queueing Systems*, 13(1-3):87–110, Mar. 1993.

[18] Z. Lotker, B. Patt-Shamir, and A. Rosen. New Stability Results for Adversarial Queuing. *SIAM Journal on Computing*, 33(2):286, 2004.

[19] S. Lu and P. P. Kumar. Distributed scheduling based on due dates and buffer priorities. *IEEE Transactions on Automatic Control*, 36(12):1406–1416, Dec. 1991.

[20] D. Poole and A. E. Raftery. Inference for Deterministic Simulation Models: The Bayesian Melding Approach. *Journal of the American Statistical Association*, 95(452):1244, Dec. 2000.

[21] A. Rosen. A note on models for non-probabilistic analysis of packet switching networks. *Information Processing Letters*, (3):0–5, Oct. 2002.

[22] A. Varga. The OMNeT++ discrete event simulation system. In *Proc. of the ESM*, volume 42, pages 319–324, 2001.

[23] M. Weinard. Deciding the FIFO stability of networks in polynomial time. *Algorithms and Complexity*, (3):81–92, 2006.