Enhancing Discrete Event Network Simulators with Analytical Network Cloud Models

Florian Baumgartner, Matthias Scheidegger, Torsten Braun
University of Bern, IAM, 3012 Bern, Switzerland

Abstract

Discrete event simulation of computer networks has significant scalability issues, which makes simulating large scale networks problematic. We propose to extend the approach by combining it with analytical models for network clouds, which are much more efficient, if less accurate, than node-by-node models. Thus, simulation scenarios containing several ISP networks become feasible. We introduce a set of suitable analytical network cloud models and also describe how these models can be implemented in the ns-2 simulator using a hot-plug mechanism.

1 Introduction

In traditional packet-based simulators the "world" is modeled in terms of nodes and links with individual capacities and delay characteristics. When simulating whole Internet domains this approach quickly becomes problematic, due to the sheer amount of events to be processed. A multitude of approaches to this scalability problem have been proposed, each with slightly different application ranges. Parallel simulation ([CM81], [ARF99]) is probably the most prominent one, but there are also the approaches of fluid flow simulation ([YG99], [LGK+99], [LFG+01]), time stepped hybrid simulation [GGT00] and packet trains [AD96], amongst others.

A simplified view of the network can significantly reduce the complexity of large scale simulations, but one must give great care to not oversimplify things. Here we propose a model, which we hope will result in far more efficient simulations than traditional approaches but should still give a good approximation of real network behavior.

In our modeling view the network is divided into domains and inter-domain links. For each domain, the set of edge nodes and their links to other domains are known, but the topology inside domains is of no concern (i.e. we have a so called black box model). The connections between such domains are modeled by inter-domain link models, which implement properties like link capacity, queuing behavior or Service Level Specifications (SLSs). Figure 1 gives a graphical rendering of this modeling view.

Domain and inter-domain link models implement different aspects of network behavior. On one hand, domains are concerned with load distribution, i.e. they model the dependencies between load on the inbound link to the resulting load on the outbound links. Inter-domain link models on the other hand model the effects these resulting network loads between domains, which are packet loss due to link overload and SLSs enforcement, amongst others. Further components of this model system are the application traffic models concerned with traffic load. They serve to scalably simulate large aggregates of application traffic (VoIP, Video, HTTP, etc.) and their specific properties. Moreover, if these models are derived from network measurements their future properties can be predicted by applying statistical means like ARIMA models.

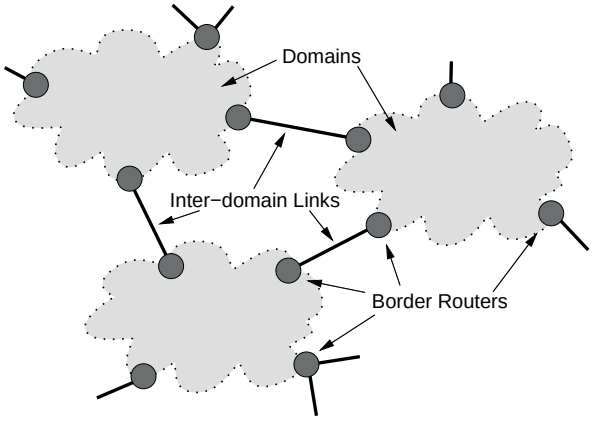

Figure 1: The basic modeling view

This structure was based on the assumption that congestion is unlikely to occur inside ISP networks since they are usually controlled by a central entity, which can, for example, change the routing to distribute traffic if there is a danger of congestion. In reality this assumption does not generally hold of course, but we expect it to be an acceptable approximation.

Since the models described here represent a situation in the real world, measurements from the network are needed to configure them. For traffic load models, these measurements consist of the loads on the inbound links of a domain and their distribution to the outbound links. Furthermore, knowledge of inter-domain topology, inter-domain link capacities and SLSs is required for realistic network modeling.

The remainder of the paper is organized as follows: In Sections 2 and 3 we describe the models for domains and inter-domain links, respectively, and Section 4 shows how

to combine them to analytical multi-domain models. The integration of these models into ns2 is explained in Section 5. Some preliminary evaluation results are shown in Section 6. Finally, Section 7 concludes the paper.

2 Domain Model

There are two main aspects to the proposed domain model. One is the traffic matrix describing the dependencies of the load parameters of *inbound* and *outbound* links, which is described in the following section. The other aspect is the continuous adaption of this matrix based on measurements done on the live network. It is covered in Section 2.3.

As mentioned above the purpose of the domain model is to describe the distribution of traffic flowing into a domain to other domains. We will investigate the scenario where we know two things: the loads on the outbound links at a given time and the origin of these loads by share of the inbound links.

2.1 Transit Matrix

As mentioned above, we need information about "traffic forking" — the distribution of traffic from an inbound link to the outbound links. Gathering this information from the ingress nodes is not easy in all cases because this might require knowledge of intra-domain topology and routing. It is often easier to determine how much of the load on an outbound link comes from a specific inbound link. As a result we get the outbound loads $O_t = (O_{1,t}, \ldots, O_{m,t})$ at time t and the relative contributions σ_{ji} of inbound link j to the load on outbound link i, for every pair (i, j). Thus, the load going from inbound link j to the outbound link i is given by $\sigma_{ji}O_{i,t}$. Earlier, we stated the assumption that there is only negligible congestion, and therefore packet loss, in a single network domain. We can therefore state that "inbound load = outbound load", or more formally¿

$$\sum_{i=1}^{m} O_{i,t} = \sum_{j=1}^{n} I_{j,t} \tag{1}$$

With this result it follows that the load on a given inbound link j is

$$I_{j,t} = \sum_{i=1}^{m} \sigma_{ji} \cdot O_{i,t} \tag{2}$$

so the whole system can be written as

$$\begin{pmatrix} I_{1,t} \\ \vdots \\ I_{n,t} \end{pmatrix} = \begin{pmatrix} \sigma_{11} & \cdots & \sigma_{1m} \\ \vdots & \ddots & \vdots \\ \sigma_{n1} & \cdots & \sigma_{nm} \end{pmatrix} \begin{pmatrix} O_{1,t} \\ \vdots \\ O_{m,t} \end{pmatrix} \tag{3}$$

The sum of elements of the matrix' column vectors is 1.

In a simulation scenario the given values are usually the inbound loads. What we would like to have is a so called *transit matrix* T, so we can write

$$O_t = \begin{pmatrix} \tau_{11} & \cdots & \tau_{1n} \\ \vdots & \ddots & \vdots \\ \tau_{m1} & \cdots & \tau_{mn} \end{pmatrix} \cdot I_t + \varepsilon_t \tag{4}$$

where ε_t is the error term. A way to convert the original result to the transit matrix notation is necessary to accommodate for that. As seen in Equation (2) the inbound load $I_{j,t}$ is the total of m terms of the form $\sigma_{ji}O_{i,t}$, for some j. The elements τ_{ij} of the transit matrix can thus be calculated by

$$\tau_{ij} = \frac{\sigma_{ji}O_{i,t}}{\sum_{k=1}^{m} \sigma_{jk}O_{k,t}} \tag{5}$$

or, if I_t is known, by

$$\tau_{ij} = \frac{\sigma_{ji}O_{i,t}}{I_{j,t}} \tag{6}$$

given that $I_t \neq 0$. Otherwise, we can assume $\tau_{ij} = 0$. Again, the resulting transit matrix has the property that the element sum of the column vectors equals 1.

2.2 SLSs and Traffic Classes

The transit matrix model discussed above only considers one load parameter per inbound or outbound link. However, there often are several load parameters for various traffic classes defined in service level agreements. Based on the assumption of a "well-behaving network domain" (see Equation (1)) these traffic classes can be considered as independent traffic aggregates. This allows us to describe multiple traffic classes by using one transit matrix per traffic class and domain. Instead of a single transit matrix T the domain model then consists of a set of matrices $T_1, \ldots, T_C$ where C is the number of separate traffic classes.

2.3 Matrix Adaption

Network domains have constantly changing characteristics. Accordingly, a domain's traffic matrix also changes over time. While the calculation of transit matrices already requires a time series of outbound load vectors and distribution matrices, modeling the changes of domain behavior requires a time series of traffic matrices.

From a time series of outbound load vectors O_t and distribution matrices D_t ($t = 0, \ldots$) we thus have to calculate a time series of transit matrices T_u ($u = 0, \ldots$). Let s be the number of observations required to get a good estimate of the momentary transit matrix of a domain. Then the calculation of matrix T_u is based on O_v and D_v where $v = u \cdot s, \ldots, u \cdot (s + 1) - 1$. I.e. we make s-sized groups of outbound load vectors and distribution matrices and calculate a transit matrix from each.

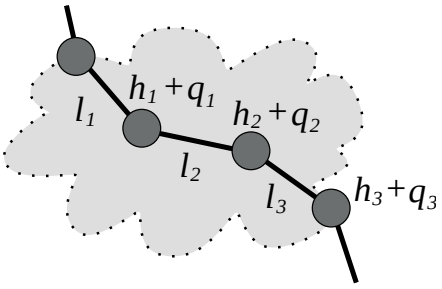

Figure 2: General Domain Delay Model

Based on this time series of transit matrices we can use several prediction mechanisms like moving average and autoregressive models. However, one problem still remains: The predicted matrices should have the same properties as "real" transit matrices, such as the ones discussed above. Predicting each matrix element in isolation cannot guarantee this. This problem has not been resolved yet and will be a focus of further research.

2.4 Domain Delay

A well-known way of looking at end-to-end delay is to divide it into link, processing and queuing delay. These effects appear in both, domains and inter-domain links. It is a viable approach to build their delay models by deciding whether and how to model these elements of delay.

When looking at the case of a path going through a domain it appears clear that all three effects are strongly dependent on the number of hops inside the domain involved. The following formula describes a very general model based on this assumption:

$$Delay = \sum_{j=1}^{m} l_j + p_j + q_j$$

Here, l_j are the link delays, p_j the processing delays and q_j the queuing delays of hop j. Figure 2 shows this graphically for the case $m = 3$.

The link delays l_j are probably not equal to each other but all of them are constant. We can therefore replace these terms in the formula by the constant L. Processing delays on the other hand are not constant and often depend on small timing variations inside the routers. Experiments performed in our testbed have shown that processing delays are approximately poisson distributed.

Because we assume that network domains are well-behaving and congestion only occurs in inter-domain links, queuing delays can only be caused by the burstiness of traffic, which gets smaller the more "backbone characteristics" the domain has. Traffic in backbone domains tends to be smooth because of the great number of microflows it consists of. During validation we will try to show that queuing delays inside domains are really negligible or at least describable by simple means. If the above proves true the domains delay model can be reduced to the simpler formula

$$Delay = L + \sum_{j=1}^{m} p_j$$

where L is constant and p_j are poisson distributed random variables.

It is important to note that the black box domain model does not allow to store delay characteristics per node in the domain. Each path between two edge nodes must have an own model, include hop count as well as link delay and processing delay characteristics. The adequate model for a path can then be selected by looking at routing information.

3 Inter-Domain Link Model

When choosing an analytical model for inter-domain links, possible alternatives are the traditional queuing theory approach or a fluid simulation approach. While the former may become difficult to calculate in the case of complicated SLSs, the latter may be expected to perform better in such situations but also to yield inferior exactitude.

3.1 Fluid Approach

Unlike domain models the inter-domain link model has only one load source, which can usually be described with a single scalar I_t. If multiple traffic classes have to be distinguished — in DiffServ environments for example — this parameter becomes $I_t = (I_{1,t}, \ldots, I_{C,t})$, where C is the number of different traffic classes. The vector elements each describe the bandwidth used by the traffic aggregate of a single traffic class. Models of multiple levels of complexity can be defined based on this input parameter and additional knowledge, such as priorities of traffic classes.

3.1.1 Traffic Class Model Tree

Interrelations between traffic classes can take various forms. Expedited Forwarding for example has absolute priority over other traffic classes, as long as the used capacity remains below a previously fixed value. A more complex example is Assured Forwarding with its three subclasses called dropping precedences, which influence themselves while the whole of them influences other traffic aggregates.

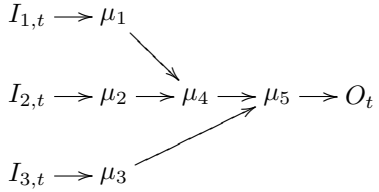

Figure 3: Example of a Model Hierarchy

Using a model hierarchy these interrelations can easily be modelled. The leaves of this hierarchy tree model the

behavior of single traffic classes. Nodes higher in the hierarchy model the interrelations between multiple traffic aggregates below them. Figure 3 shows an example with three leaf nodes and one intermediate node, and of course one root node. In algebraic form the model would read

$$O_t = \mu(I_t) = \mu_5(\mu_4(\mu_1(I_{1,t}), \mu_2(I_{2,t})), \mu_3(I_{3,t}))$$

Below a few functions useful to construct such model hierarchies will be defined. In order to make the functions freely combinable they all have to be of a certain algebraic form, which is

$$\mu(B, \vec{P}, \vec{M}, \vec{A}). \tag{7}$$

Parameter B is the maximum available bandwidth, $\vec{P}$ is a vector of parameters specific to μ, $\vec{M}$ is a vector of n submodels and $\vec{A}$ is a vector of argument vectors for these submodels. The elements of $\vec{A}$ again have the form $(\vec{P}, \vec{M}, \vec{A})$, although with other vector sizes. The inbound bandwidths $I_{i,t}$ are considered as constant functions in this context. Furthermore, the following function will help readability:

$$m(b, \mu, \vec{a}) = \min(b, \mu(b, \vec{a})) \tag{8}$$

which according to the above algebraic form can also be written as

$$M(B, (b_1, \ldots, b_n), \vec{M}, \vec{A}) = \begin{pmatrix} m(b_i, \mu_i, \vec{a_i}) \\ \vdots \\ m(b_n, \mu_n, \vec{a_n}) \end{pmatrix}$$

if and only if $B \geq \sum_{i=1}^{n} b_i$ holds.

Fixed Bandwidth Shares The simplest useful function is the one modeling fixed bandwidth shares. Incoming bandwidth from the models in $\vec{M}$ is simply truncated to a maximum bandwidth B according to bandwidth shares in $\vec{P} = (s_1, \ldots, s_n)$. We get the *fixed share* function

$$F(B, \vec{P}, \vec{M}, \vec{A}) = \begin{pmatrix} m(s_1 B, \mu_1, \vec{a_1}) \\ \vdots \\ m(s_n B, \mu_n, \vec{a_n}) \end{pmatrix} \tag{9}$$

Priority Multi-Queue Multi-queue systems often implement a system of fixed priorities where queues with small priorities only can send if all queues with higher priority are empty. This property can be modelled by the function

$$P(B, (), \vec{M}, \vec{A}) = \begin{pmatrix} m(\beta(1), \mu_1, \vec{a_1}) \\ \vdots \\ m(\beta(n), \mu_n, \vec{a_n}) \end{pmatrix} \tag{10}$$

The name P stands for the priority-dependent behavior of the function. The helper function β yields the bandwidth available to μ_i and is defined as

$$\beta(i) = \begin{cases} B & i = 1 \\ B - \sum_{j=1}^{i-1} P_i & otherwise \end{cases} \tag{11}$$

Dropping Precedences Assured Forwarding (AF) defines three dropping precedences—low, medium and high—which cannot be modelled with the functions above. AF packets are normally marked with low dropping precedence when they are sent and only change that status to a higher dropping precedence if they are detected to be "out of profile" by a router on their path. Then, in case of congestion, routers drop packets with higher dropping precedence first. The following function models a generalized variant of this approach with an arbitrary number of dropping precedences:

$$D(B, (b_1, \ldots, b_{n-1}), \vec{M}, \vec{A}) = \begin{pmatrix} D_1 \\ \vdots \\ D_n \end{pmatrix}$$

Note the parameters $b_1, \ldots, b_{n-1}$: In contrast to the bandwidth share parameters above they stand for absolute bandwidths. Additionally we assume $b_1 \leq B$. Again, we need helper functions to describe the elements of the result vector. First, function r calculates the amount of bandwidth of a given dropping precedence that has to be retagged to a higher dropping precedence:

$$h(i) = \begin{cases} \max(\mu_1(b_1, \vec{a_1}) - b_1, 0) & i = 1 \\ \max(\mu_i(b_i, \vec{a_i}) + h(i-1) - b_i, 0) & i > 1 \end{cases} \tag{12}$$

Using this function we can see how much bandwidth remains for a given dropping precedence by

$$r(i) = \begin{cases} m(b_1, \mu_1, \vec{a_1}) & i = 1 \\ \mu_n(\infty, \vec{a_n}) + h(n-1) & i = n \\ m(b_i, \mu_i, \vec{a_i}) + h(i-1) & otherwise \end{cases} \tag{13}$$

so we can finally write

$$D_i = \begin{cases} \min(r(1), B) & i = 1 \\ \min(r(i), B - \sum_{j=1}^{i-1} D_j) & otherwise \end{cases} \tag{14}$$

Fair Queueing Fair Queueing and Weighted Fair Queueing are popular approaches to manage QoS. Our proposed system should therefore provide a rendering of their behavior, or rather just of Weighted Fair Queueing since it is a generalization of Fair Queueing. The parameter vector has again the form $\vec{P} = (s_1, \ldots, s_n)$ here. If not all of the submodels completely use up their bandwidth share, the remaining bandwidth can be used by the other submodels, according to their share. The calculation of the resulting bandwidth shares is best done algorithmically. We need the following definition:

$$\alpha(i, b) = \begin{cases} 1, & \mu_i(b, \vec{a_i}) \geq b \\ 0, & otherwise \end{cases} \tag{15}$$

To begin let $\iota = \{1, \ldots, n\}$, $W = 1$ and $b_1 = \ldots = b_n = 0$. The sum of the unused bandwidths of the single flows is calculated by

$$U = \sum_{i \in \iota} \alpha(i, \frac{s_i}{W} B + b_i)(\frac{s_i}{W} B + b_i - \mu_i(\frac{s_i}{W} B + b_i, \vec{a_i}))$$

Then, for every $i \in \iota$, reassign

$$b_i = \begin{cases} \frac{s_i}{W}B + b_i, & \alpha(i, \frac{s_i}{W}B + b_i) = 1 \\ \mu_i(\frac{s_i}{W}B + b_i, \vec{a_i}), & otherwise \end{cases}$$

In the next round only the "unsatisfied" models participate. We reassign

$$\iota = \{i : i \in \iota \wedge \alpha(i, \frac{s_i}{W}B + b_i) = 1\}$$

The sum of weights must also be adjusted, thus we reassign

$$W = \sum_{i \in \iota} s_i$$

From here go to the calculation of U until ι is the empty set. The variables b_i then contain the bandwidths assigned to the models μ_i. The complete model is then given by

$$W(B, \vec{P}, \vec{M}, \vec{A}) = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix} \tag{16}$$

An Example An example should help clarifying the use of the above system: Consider an inter-domain link with the queueing system shown in Figure 4.

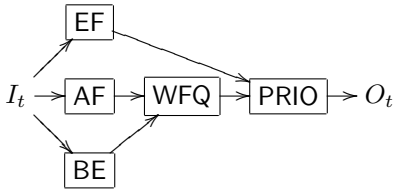

Figure 4: Inter-Domain Link Example

Let the total link bandwidth be 100Mbit/s and the bandwidths for the AF dropping precedences be 40Mbit/s, 8Mbit/s and 2Mbit/s, for low, medium and high dropping precedence, respectively. Further, let the WFQ weights be 0.5 for both inputs and 25Mbit/s be the maximum bandwidth allowed for EF traffic. Using the prototypical functions from above we get

$$M(\mathsf{X}, (25), (I_{1,t}), ())$$

for the EF queue and

$$D(\mathsf{X}, (40, 8, 2), (I_{2,t}, I_{3,t}, I_{4,t}), ())$$

for the AF queue (X will be replaced by the function higher in the hierarchy). The Best Effort queue simply uses as much bandwidth as it can get, so we can directly take $I_{5,t}$ as a model for it. Combining these we get

$$W(\mathsf{X}, (0.5, 0.5), (D, I_{5,t}), \vec{A_W})$$

for the WFQ queue with

$$\vec{A_W} = (\underbrace{((40, 8, 2), (I_{2,t}, I_{3,t}, I_{4,t}), ())}_{D}, \underbrace{()}_{I_{5,t}})$$

Using a P function to combine these, we get the final model

$$O_t = P(100, (), (M, W), (\underbrace{((25), (I_{1,t}), ())}_{M}, \vec{A_W}))$$

3.2 Queuing Theory Approach

Traditionally, analytical network models have been based on the queuing theory originating from operations research and the like. Although creating such a model for a given system is often non-trivial, the results are both accurate and efficient. On the other hand, larger systems become very complicated to model.

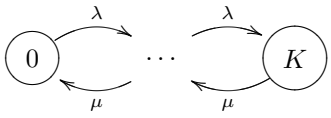

Figure 5: Birth and Death Process

In the simple case of one queue per inter-domain link we can use a classic M/M/1/K queue, that is, a queue with exponentially distributed inter-arrival time τ and service time s, a single "processing station" (the physical link) and a system capacity K. The arrival and service rates are given by $\lambda = 1/E(\tau)$ and $\mu = 1/E(s)$, respectively. This system is a birth and death process as shown in Figure 5. For a birth and death process of this kind the probability p_i of the system to be in state i is given by

$$p_i = \begin{cases} \frac{1-\lambda/\mu}{1-(\lambda/\mu)^{K+1}}, & i = 0 \\ (\lambda/\mu)^i p_0, & i > 0 \end{cases} \tag{17}$$

if $\lambda \neq \mu$, and

$$p_0 = p_1 = \ldots = p_K = \frac{1}{K+1} \tag{18}$$

if $\lambda = \mu$. Because here we are only concerned with the changes to the traffic load caused by the queueing system, there is now a very simple way to simulate the dropping behavior. Arriving packets will only be dropped if the queue is full, which is the case with probability p_K. It is therefore sufficient to randomly drop an adequate fraction of the arriving packets, or in the case of a input load parameter I_t to write

$$O_t = (1 - p_K)I_t \tag{19}$$

Assured Forwarding As mentioned above Assured Forwarding defines three dropping precedences. The differences in behavior towards these precedences are usually implemented by beginning to drop packets at different fill levels of the queue. This can again be modelled by a birth and death process, although a more complicated one (see Figure 6). The arrival rate λ consist of three rates λ_l, λ_m and λ_h, for low, medium and high dropping precedence, respectively, with $\lambda = \lambda_l + \lambda_m + \lambda_h$. The system capacity

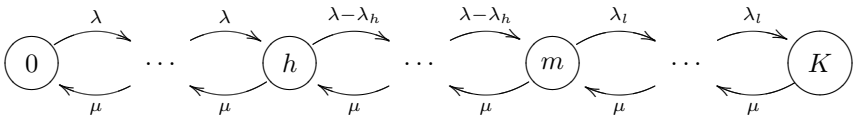

Figure 6: AF Birth and Death Process

is again K. Medium packets can only be queued if the system contains less than m packets, and high packets only if it contains less than h. The system state probabilities for $i > 0$ are

$$p_i = \begin{cases} \left(\frac{\lambda}{\mu}\right)^i p_0, & 0 < i \leq h \\ \left(\frac{\lambda-\lambda_h}{\mu}\right)^{i-h} p_h, & h < i \leq m \\ \left(\frac{\lambda_l}{\mu}\right)^{i-m} p_m, & m < i \leq K \end{cases}$$

State 0 consequently occurs with probability

$$\begin{aligned} p_0 &= 1 - \left[\sum_{i=1}^{h} \left(\frac{\lambda}{\mu}\right)^i p_0 + \right. \\ &\quad \left(\frac{\lambda}{\mu}\right)^h \sum_{i=h+1}^{m} \left(\frac{\lambda-\lambda_h}{\mu}\right)^{i-h} p_0 + \\ &\quad \left. \left(\frac{\lambda}{\mu}\right)^h \left(\frac{\lambda-\lambda_h}{\mu}\right)^{m-h} \sum_{i=m+1}^{K} \left(\frac{\lambda_l}{\mu}\right)^{i-m} p_0 \right] \end{aligned}$$

After some transformations and using the terms

$$A = \left(1 - \frac{\lambda}{\mu}\right),\ B = \left(1 - \frac{\lambda-\lambda_h}{\mu}\right),\ C = \left(1 - \frac{\lambda_l}{\mu}\right)$$

we can write

$$\begin{aligned} p_0 &= ABC / \left[1 - \left(\frac{\lambda}{\mu}\right)^{h+1} BC - \right. \\ &\quad \left(\frac{\lambda}{\mu}\right)^h \left(\frac{\lambda-\lambda_h}{\mu}\right)^{m-h+1} AC - \\ &\quad \left. \left(\frac{\lambda}{\mu}\right)^h \left(\frac{\lambda-\lambda_h}{\mu}\right)^{m-h} \left(\frac{\lambda_l}{\mu}\right)^{K-m+1} AB \right] \end{aligned}$$

Analogous to the simple queue above the output loads are then calculated using

$$O_{h,t} = I_{h,t} \cdot \sum_{i=0}^{h-1} p_i \quad (20)$$

$$O_{m,t} = I_{m,t} \cdot \sum_{i=0}^{m-1} p_i \quad (21)$$

$$O_{l,t} = I_{l,t} \cdot (1 - p_K) \quad (22)$$

Note that the probabilities used also change for every t. Due to the rather heavy calculations involved the above model is not suited to very small sampling intervals.

Schedulers Queuing systems with multiple queues and a single outgoing interface need one or more schedulers to decide which queue is allowed to send when the interface is done sending a packet. Some of the most frequently used schedulers are the Weighted Fair Queuing and Priority schedulers.

To model WFQ we can almost immediately use the Fair Queuing approach from Section 3.1.1. Instead of nesting functions to determine the output loads of submodels we can directly use the service rates $s_i\mu$ ($i = 1, \ldots, n$), and instead of the output bandwidth B we use a known service rate μ. Going through the algorithm yields the adjusted service rates for the queues. By recalculating the models with these rates we get the final outbound loads for every queue.

Priority schedulers can be modeled with a slightly modified version of the approach in Section 3.1.1. The system consists of n queues with arrival rates $\lambda_1, \ldots, \lambda_n$ and service rates $\nu_1, \ldots, \nu_n$. The service rates $\nu_1, \ldots, \nu_{n-1}$ are fixed and have the property

$$\sum_{i=1}^{n} \nu_i \leq \mu$$

where μ is the service rate of the priority scheduler itself. ν_n is given by

$$\nu_n = 1 - \sum_{i=1}^{n} \nu_i$$

The output loads of the queues $1, \ldots, n-1$ does not change. That of queue n is obtained by evaluation it with service rate ν_n.

3.3 Inter-Domain Delay

Inter-domain delays are mainly due to a fixed link delay L and a queuing delay Q. An analytical model is a natural approach to the latter. In the trivial case, a single Best-Effort queue, an M/M/1/K queue, can be used, with arrival and service rates derived from the average observed packet size S, the input load I and the output bandwidth B of the link. The system capacity K (i.e. the queue length plus one) can be set to a typical value if it is not known a priori.

Calculating the delay distribution for an M/M/1/K queue is rather easy: Let $\lambda = I/S$ and $\mu = B/S$ be the arrival and service rates. The system state probabilities are computed ananlogously to Section 3.2. When the system holds i packets, an arriving packet experiences a delay of i/μ. The delay distribution of the whole inter-domain link is thus

$$\begin{pmatrix} p_0 & p_1 & \cdots & p_K \\ L & 1/\mu + L & \cdots & K/\mu + L \end{pmatrix} \quad (23)$$

3.3.1 Service Differentiation

While the above model is simple an efficient it is not suited to areas where the routers differentiate between classes of packets. Well known examples are the Expedited and Assured Forwarding services and schedulers like Round Robin, Weighted Fair Queuing and Priority Scheduling. In the paragraphs below we develop analytical delay models for some of these. Note that Expedited Forwarding is trivial to model using a queuing model as above, with fixed service rate.

Assured Forwarding The three dropping precedence levels defined by Assured Forwarding do not allow us to use the simple M/M/1/K queue from above. Nevertheless, by using the model defined in Section 3.2, a similar method is possible. Again, the variables $p_1, \ldots, p_K$ designate the probaility that the system contains i packets at a given point in time, e.g. at the arrival of a new packet. Since calculating these probabilities is significantly more complex than in the M/M/1/K case we do not repeat the equations here. L and μ take the same roles as above, so the result is again the discrete distribution shown above, although with other probability values.

Priority Scheduler When multiple queues are combined using a priority scheduler the delay behavior of packets in low priority queues is strongly altered. To model this we define the following model:

A set of M/M/1/K queues $Q_1, \ldots, Q_n$ with arrival rates λ_i and service rates μ_i is combined with a priority scheduler with service rate $\mu = \mu_1 + \ldots + \mu_n$. The service rates of the queues are controlled using a token bucket with a bucket size of one packet. The delays for packets in Q_1 are the same as in a single M/M/1/K queue. Packets in lower priority queues get the same queuing delay *plus* a delay when waiting for higher priority packets to leave the system. A packet in queue i must wait for a packet in queue $j < i$ if and only if Q_j has a packet to send (probability $1 - p_0$, where p_0 comes from the occupation distribution of Q_j) and is allowed to send by its token bucket, which is true in μ_i/μ cases.

The random variables $s_1, \ldots, s_n$ give the number of packets a queue can send at a given moment

$$s_i = \begin{cases} 1, & \text{with probability } (1 - p_{i,0})\frac{\mu_i}{\mu} \\ 0, & \text{otherwise} \end{cases}$$

Let now

$$S_i = \sum_{j=1} i s_j.$$

The delay of a packet entering queue i is consequently the queuing delay plus S_i/μ.

4 Multi-Domain Model

All the models defined so far can be used as single entities in a simulator. Routing of traffic along the domains and inter-domain links is straightforward in this case: The routing information used to connect the model topology can directly be used by the simulator to determine the path a packet will take. This information may be derived from the BGP tables of the network to be simulated, for example.

It may be more efficient to combine these models into a single entity from the simulator's point of view, however. Such a multi-domain model cannot use the simulator's routing capabilities and therefore needs knowledge about the routing inside the modelled network area. It appears reasonable to assume that the number of connections from and to the multi-domain model is "small", i.e. the complexity of storing all possible paths is manageable. Given n links to the "outside" the system would have to store $n(n-1)/2$ paths. These paths are uniquely identified by the affected egress links of the first to the second last domain. This approach is also applicable to the models concerned with delay, jitter, etc. A multi-domain model is thus defined by the definitions of the included domain and inter-domain link models and a table of paths between all links connected to the "outside" of the model.

4.1 Multi-Domain Delay

In accordance with the general modeling view, the delay model divides into a model for intra-domain delay and a model for inter-domain delay. The delay caused by a domain or inter-domain link is not exactly the same for each of a series of consecutive packets of a flow. In fact, the delays of a series of packets can be more adequately modeled with probability distributions, rather than with a single value like mean delay. This is also the case for delay variations and the variations in packet interarrival times. As a consequence it is a promising approach to study the delay behavior of a packet stream of packets going through a series of network domains by adding the delay random variables of the domains and inter-domain links the stream goes through. Let the random variable of delay along a path P be

$$D^P = \sum_{i=1}^{n} D_i^P + \sum_{i=1}^{n-2} D_i^I$$

where D_i^D and D_i^I are the random delays caused by the n domains and $n-2$ inter-domain links along the path. The mean end-to-end delay of the path is then $\mathrm{E}(D^P)$ and jitter is the mean interarrival time of two packets, given the distribution of D^P. It can be calculated by taking the difference of two random variables $d_1, d_2 \sim D^P$.

$$\text{Jitter} = \mathrm{E}\left(\sqrt{(d_2 - d_1)^2}\right)$$

5 Simulator Integration

Having described the mathematical approaches for the delay and loss models, the following section focuses on the integration of these models in the ns2 network simulator.

5.1 The ns2 Network Simulator

The network simulator ns2 [ns202a] is a frequently used tool for the evaluation of new protocols and concepts. It follows a packet based, discrete event approach and supports a broad range of network protocols. However, even if ns2 supports most protocols used in the Internet, it uses a more abstract, graph-oriented view of the network topology and does not model the behavior of real network devices. In ns2 typical router functionalities like decreasing a packet's time-to-live value, or traffic conditioning are handled by ns2 links and not, as might be expected, by the nodes. In general ns2 nodes only perform routing and are used to attach agents like traffic sources and sinks. They do not cause any packet processing delays like normal routers do. Following the more abstract topology view of ns2 , we will present an extension, which not only allows single nodes to represent more realistic routers, but also allows them to represent complete router networks.

5.2 Modeling Domains

Using delay and loss models for network domains allows to simulate networks without exact knowledge of the network topology and without the need to simulate each single node within an network. Figure 7 shows how the use of domain models can simplify a network topology. Each network within the three domains can be replaced by a single node, providing the modeled behavior of the full network.

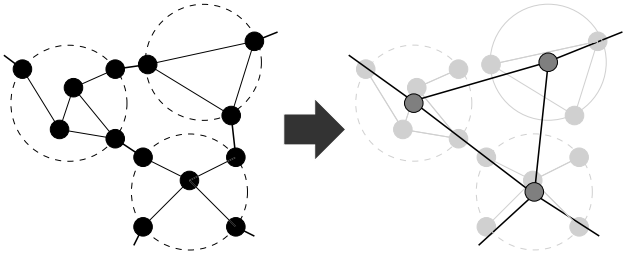

Figure 7: Reducing a network node topology to a topology of domains

This requires the node to not only simulate the routing of the replaced network, but also the delay and packet loss behavior. Therefore the ns2 nodes have to be extended to take care of delaying or dropping packets passing the node, similar to the network the node is replacing. Of course, the delay of a specific packet passing the node or the decision whether or not to drop the packet will only be statistically correct. A modeled domain will, on a packet level, usually not provide the exact same behavior as the simulation of this network.

While the left-hand topology of Figure 7 contains two types of links, intra-domain and inter-domain, in the reduced topology only the inter-domain links are simulated within ns2. The intra-domain links are part of the model. In Section 5.4 the architecture will be extended to allow the modeling of a set of domains as well as the integration of non-packet-based traffic models. However, for ns2 it makes no difference whether a node is only a single node or represents a single or a set of modeled domains.

5.3 Integration into ns2

For the integration into ns2 the node has to be slightly modified, as can be seen in Figure 8 (taken from [ns202b]). The typical ns2 node consists of an address classifier, a port classifier and a set of agents. The address classifier routes incoming packets either directly to outgoing links or to a port classifier forwarding the packets to an appropriate agent.

To delay and discard packets, an additional component, the Delay and Loss Predictor (DePred, LoPred) was implemented and put before the address classifier. Therefore incoming packets are first processed by the predictor module, which decides how long the packet has to be delayed, or whether the packet has to be dropped.

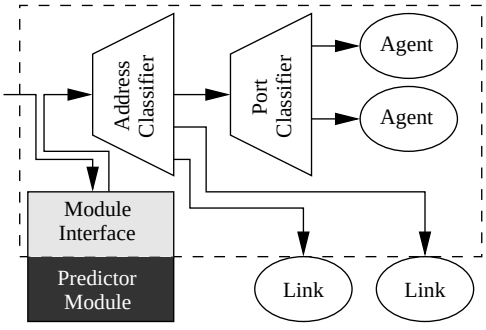

Figure 8: Integrating the Delay and Loss Predictor module into the ns2 node.

The module itself is not part of the ns2 node. The node only provides an interface to the external module and takes care of delaying or dropping the packet. Apart from some functions allowing the dynamic loading and unloading, the initialization and configuration of the module, a predictor module only has to provide a single function which is called for each arriving packet. Depending on the return value, the module interface either discards or delays the packet. With each call of the predictor function, the predictor gets information about the packet itself, the previously passed and the next ns2 node, and the simulation time. Any type of metering has to be done by the predictor itself.

For the ns2 user the access to the modules is very simple. The only difference to standard ns2 is that nodes now can be extended with domain modules. Table 1 shows a

Table 1: `ns2` code, initializing a module, attaching it to a node and configuring the module with a configuration file

```
set ns [new Simulator]
set n0 [$ns node]
set c0 [new ISPmodule]
$n0 attach-isp-module $c0
$c0 config load "module.config"
```

Table 2: Full source code for a simple predictor module

```
#include "isp_sample.h"

ISP_MODULE_INIT(ISP_sample)

ISP_sample::ISP_sample(Node *n) : ISP_module(n)
  { return; }

double ISP_sample::process_packet(Node *prev,
  Node *next, double time, struct ISP_pinfo *p)
  { return 0.1; }
```

script, which instantiates a simulator, creates a node and a module reference, and attaches the module to the node. Finally, the module is configured by loading a configuration file.

Table 2 shows the C++ source code of a minimal domain module causing each packet, entering the node to be delayed by 0.1 seconds. The `process_packet(...)` function is called for each arriving packet, and returns the time the packet has to be delayed. A negative value would cause the node to drop the packet. The parameters `Node *next`, `Node *prev` allow taking into account from where the packet was received and to which next node it will be forwarded. The `struct ISP_pinfo` contains information about the packet itself, like source and destination addresses, protocol and Differentiated Services Code Points.

5.4 Multi Domain Models

As an extension to the concept, it is not only possible to model a single domain, but also a set of domains within a single node. While the inter-domain links between single domain modules are provided by `ns2` on a per-packet basis, intra-domain links of multi-domain modules are provided by the module itself. This allows to model intra-domain links in multi-domain modules using the various approaches shown in Section 3.

Furthermore, the domain models may also include non-packet-based traffic sources modeling video or http traffic.

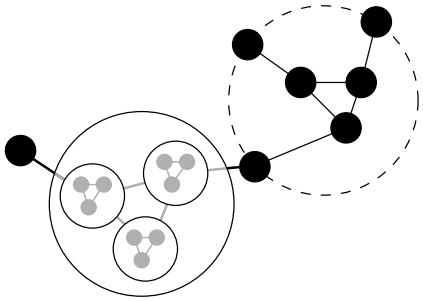

Figure 9: Integrating the Delay and Loss Predictor module into the `ns2` node.

Of course the integration of such traffic sources is limited to the module itself. Since modules most probably will not work on a per-packet basis, these traffic sources so far only may be used within the Predictor module itself.

6 Evaluation

For a preliminary evaluation of the concept, the delay characteristics between the network of the University of Bern and the ETH Zürich have been investigated. In a first step the delay between two hosts in the networks was measured. Both networks are connected by the Swiss scientific network SWITCH [swi03], the distance between the measurement hosts was nine hops. The measurements were done by simply sending a series of echo requests through the network. Based on these measurements, an empiric distribution was computed and used to configure a delay predictor for `ns2` . For the simulation the simple `ns2` network in Figure 10 with only three nodes was set up. While the two outer nodes act as source and sink, the central node has the predictor module attached.

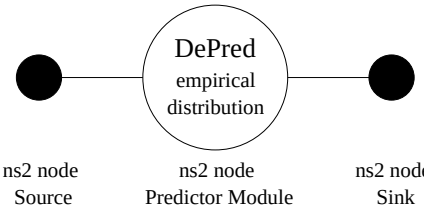

Figure 10: The `ns2` setup to simulate the delay of a single Internet Service Provider.

Figure 11 shows a comparison between the measured delays and the delays in the simulation. Both graphs show almost exactly the same delay behavior for the measurement and the simulation.

7 Conclusion

In this paper we presented a scalable approach to simulating large scale inter-domain networks. This scalability

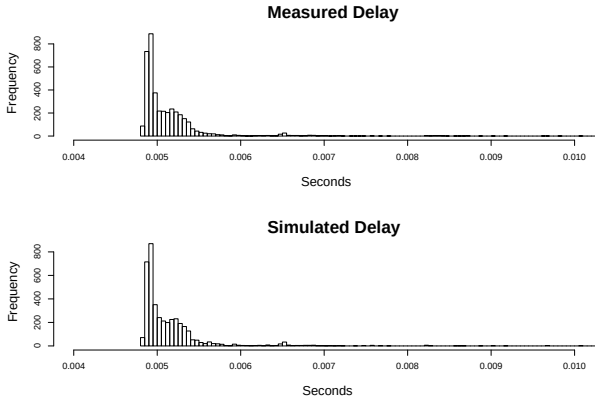

Figure 11: Delay histograms based on measurements (upper graph) and on simulation (lower graph).

is achieved by replacing node-per-node simulation of network domains by analytical models for domains and inter-domain links. These models are configured by measuring the characteristics of a live network and can then predict delay and dropping behavior of this network. In the very large scale, we also combined these models represent multi-domain networks. We also presented the integration of these models into `ns2` , which is done by extending `ns2` with a hot-plug mechanism to dynamically load models into nodes. Some preliminary evaluation has also been done, comparing the measured delay between two real network nodes to the results of the corresponding delay model in `ns2` .

References

[AD96] Jong Suk Ahn and Peter B. Danzig. Packet network simulation: speedup and accuracy versus timing granularity. *IEEE/ACM Transactions on Networking*, 4(5):743–757, October 1996.

[ARF99] M. H. Ammar, G. F. Riley, and R. M. Fujimoto. A generic framework for parallelization of network simulations. In *Proceedings of the 7th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, College Park, MD, October 1999.

[CM81] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 11(24):198–205, April 1981.

[GGT00] Yang Guo, Weibo Gong, and Don Towsley. Time-stepped hybrid simulation (TSHS) for large scale networks. In *Proceedings of IEEE Infocom*, March 2000.

[LFG+01] Benyuan Liu, Daniel R. Figueirido, Yang Guo, Jim Kurose, and Don Towsley. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Proceedings of IEEE Infocom*, April 2001.

[LGK+99] B. Liu, Y. Guo, J. Kurose, D. Towsley, and W. Gong. Fluid simulation of large scale networks: Issues and tradeoffs. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 2136–2142, Las Vegas, NV, June 1999.

[ns202a] The network simulator - ns-2, Information Science Institute, University of Southern California, http://www.isi.edu/nsnam/ns, June 2002.

[ns202b] The ns manual. http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf, June 2002.

[swi03] Switch – the swiss education & and research network. http://www.switch.ch, February 2003.

[YG99] A. Yan and W. B. Gong. Fluid simulation for high speed networks with flow-based routing. *IEEE Transactions on Information Theory*, pages 1588–1599, 1999.