DeSiNe: a flow-level QoS Simulator of Networks

T. Kleiberg,* B. Fu, F. A. Kuipers and P. Van Mieghem
Networking, Architectures and Services Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology, The Netherlands
{T.J.Kleiberg, B.Fu, F.A.Kuipers, P.F.A.VanMieghem}@tudelft.nl

S. Avallone
COMICS Lab
Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II, Napoli, Italy
Stavallo@unina.it

B. Quoitin
IP Networking Lab
Computer Science and Engineering Department
Université Catholique de Louvain, Belgium
Bruno.Quoitin@uclouvain.be

ABSTRACT

In this paper we present DeSiNe, a modular flow-level network simulator. DeSiNe is aimed at performance analysis and benchmarking of Quality of Service routing algorithms and traffic engineering extensions. Several well-known QoS routing algorithms and traffic engineering extensions have been implemented in DeSiNe. The flow-level nature provides scalability, such that large networks and heavy-traffic conditions are possible. In this paper, the functional and structural design of DeSiNe are presented and the usability and various features are illustrated by means of several examples. The source code of DeSiNe is publicly available.

Categories and Subject Descriptors

I.6 [**Simulation and modeling**]: General

Keywords

Simulation, Computer networks, Quality-of-Service

1. INTRODUCTION

The growth in both complexity and size of data communication networks makes the tasks of testing and measuring very complex. Testing networks through realistic test-beds requires many nodes, which is rather costly and involves many practical limitations with respect to e.g. network size, and configuration. Measuring is difficult, since typically one cannot arbitrarily sample the topology or the traffic in large networks like the Internet [1]. Moreover, the "repeatability", or trustworthiness, of scenarios in real network is difficult to guarantee. The most accessible method is to study networks and to test new protocols and algorithms via simulation. Based on the abstraction level at which the traffic is modeled, simulators can be roughly categorized as packet-level and flow-level simulators. A third category, hybrid simulators, combine packets and flows.

Packet-level simulation has gained most attention in the past. In packet-level discrete event simulators, the arrival and departure of each packet triggers an event in the simulator. These simulators use a level of detail that closely reflects reality. However, this high level of detail comes at the expense of high computation time and memory usage. Packet-level simulators are mainly applicable to small networks and are often used for studies where a high level of detail is required, e.g. monitoring signalling messages of a new protocol, or the effect of a new buffer management scheme in a router.

Alternatively to packet-level simulation, we can raise the level of abstraction to the flow-level. We regard a flow as a connection established between a source and a destination node in a network. All the packets of that flow follow the same path from the source to the destination during the life-time of that flow. Consequently, in flow-level simulations, the packet-level details are not considered, which makes flow-level simulations more suitable for scenarios with larger networks and a large number of flows. Moreover, in the context of Quality of Service (QoS) routing [2, 3], resources should be reserved, i.e. flows set-up, in order to guarantee QoS. Understanding the behavior of flows in a network is invaluable in designing QoS-aware networks.

In this paper we present DeSiNe. DeSiNe stands for Delft Simulator of Networks and is a scalable flow-level QoS simulator. DeSiNe incorporates QoS routing and traffic engineering algorithms. The purpose of DeSiNe is to study and compare the performance of various QoS routing and traffic engineering implementations at the network level. In particular, DeSiNe supports constraint-based routing and dynamic QoS routing, as well as several on-line traffic engineering algorithms. The good scalability permits the simulation of large networks and heavy-traffic scenarios.

The remainder of this paper is organized as follows. In Section 2 we position DeSiNe in the related work. Its design and features are presented in Section 3. To illustrate the usability of DeSiNe, simulation examples are given in Section 4. We conclude in Section 5.

2. RELATED WORK

The simulation of network protocols is usually performed at the level of packets with tools such as the popular open-source ns-2

*Corresponding author.

QOSIM 2008, March 03, Marseille, France

DOI 10.4108/ICST.SIMUTOOLS2008.3006

[4] and SSFNet [5] simulators. These simulators aim at an accurate computation with high level of detail and high fidelity. They model network protocols at the level of packets. The major advantage of this approach is that the protocol model does not differ largely from the real protocol implementation. At the same time, this accuracy comes at a cost. The main limitation of packet-level simulators is their lack of abstraction as considerable amounts of computer resources are needed to maintain the complete state of network protocols, especially for large network simulations.

There are two main causes to the lack of scalability of packet-level simulators. Firstly, packet-level simulators rely on discrete-event simulation (DES). The principle of DES consists in keeping all the network protocol events ordered according to their time of occurrence, using a priority queue. Typical events in this context are the transmission of a data unit by a protocol or the expiration of a protocol timer. Additionally, intermediate events might also be created in order to model the CPU workload or queueing delays at various levels of the protocol stack. Traditional DES implementations relied for their operation on priority queues which have a time complexity of $O(log(n))$ to insert new events. The advent of calendar queues in modern schedulers has reduced this complexity to $O(1)$ on average.

Secondly, packet-level simulators lack a level of abstraction in the protocol stack. Each protocol model is extremely accurate, often including the complete protocol finite state machine. This requires significant time and space to hold on the computer. Moreover, heavy traffic conditions increase the computational cost of packet-level simulations.

Indeed, each packet-flow is composed of several thousands of packets that must go through the network stack of several nodes, generating a lot of events and increasing the running time of the simulation. One can say that for a single flow, the running time increases linearly with the number of packets in the flow.

To circumvent these fundamental problems of packet-level simulation, various approaches have been proposed. Several methods rely on exact solvable models instead of simulation. For example, Anick *et al.* [6] proposed an exact solvable queuing model that describes the queuing behavior at the queue level in their seminal paper. Although their system allows for high accuracy, their hypothesis of independent, continuous on-off sources is too restrictive. The majority of proposed solutions however seek to enhance the scalability and performance of network simulators by lowering the granularity of the simulation or raising the level of abstraction, specifically the level of traffic abstraction.

One approach is to consider the traffic at the level of flows. For example, Flowsim [7] uses DES, but coarsens the granularity of the network traffic by aggregating individual packets into packet-trains. Hence, the number of events is reduced at the expense of the accuracy of the simulation.

A second approach, known as the fluid-flow model [8, 9], consists in keeping track of the sending rate of sources. This is in contrast with DES which keeps track of any single packet transmission. The fluid-flow model is particularly suitable for studying the impact of congestion on the traffic sending rate, as is the case with TCP. Under light or moderate traffic conditions, the number of events raised in flow-level simulators is typically much less compared to packet-level simulators. However, it has been shown that under heavy traffic conditions and when several flows share the same available resources, one change in the sending rate of a single flow may influence the sending rate of many other flows. This can cause an avalanche of sending rate updates with a dramatic impact on the running time of the simulation. This effect is known as the *"ripple effect"* [8, 10], and may lead to drastic performance degradation, such that packet-level simulation will outperform flow-level simulation.

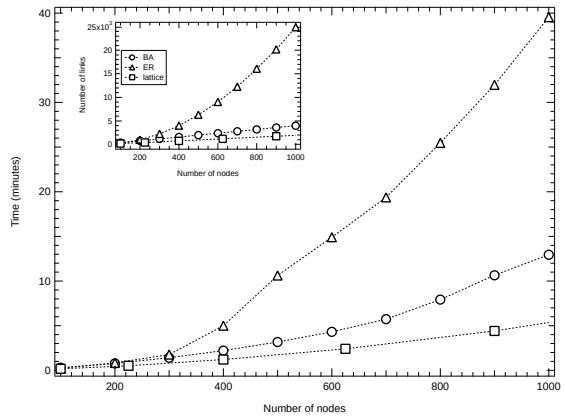
Another scalability improvement can be obtained by switching from a packet-level event-driven simulation to a packet-level time-stepped simulation. In a time-driven fluid simulation, the continuous traffic flow is discretized into time intervals. The time-driven nature relieves the problem of the ripple effect, since the network is only sampled at fixed intervals. This is the approach exposed in [11, 12]. By using coarser time scales it is possible to further speed up packet-level time-stepped simulators. Hybrid packet/fluid-flow simulators use packets for foreground traffic, but model the background traffic at fluid-level. Examples of hybrid simulators have been proposed by Nicol *et al.* [13] and Yung *et al.* [14]. The Hybrid Discrete-Continuous Flow Network Simulator (HDCF-NS) [15] is another example, but little information is given about the actual model and how flows and packets interact. The IP-TN simulator [16] defines hybrid nodes that are capable of mixing both traffic models. The hybrid nodes estimate the aggregate input rate of the packets and the capacity is shared with the flows proportional to the combined input rates.

Most simulation studies on QoS routing have created a simulation environment dedicated to the evaluation of their proposed QoS algorithm or policy. When we consider the field of QoS routing algorithms, often only the algorithm is implemented and run on several static networks, without considering traffic and flow dynamics. Several QoS simulators have been published and we will briefly mention them, here. Zhang *et al.* [17] have developed the packet-level QoS Routing Simulator (QRS), which was later extended to EQRS [18] to capture DiffServ MPLS networks. A. Shaikh [19] has developed a flow-level event-driven QoS simulator called *routesim*, which focusses on the evaluation of link-state update policies. Sivasankar *et al.* [20] developed a flow-level simulator, called MuSDyR. DeSiNe differs from the previous simulators, because it contains all, instead of only some, of the following features: (a) Dedicated to evaluate all aspects of QoS routing, including traffic engineering. (b) Flow-level abstraction, for scalability, but also as a natural consequence of QoS routing. (c) Many built-in QoS mechanisms. (d) Modular design, such that it can easily be extended with new QoS mechanisms.

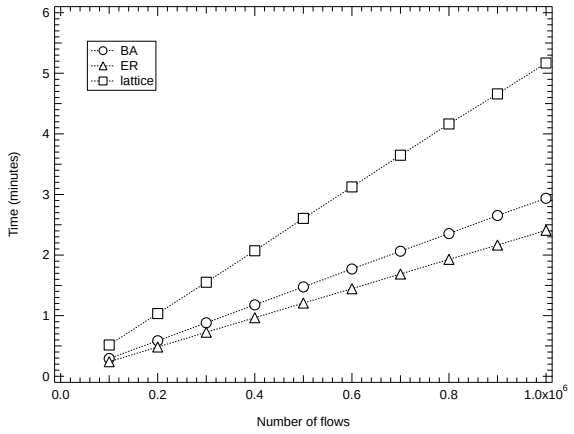
3. SIMULATOR DESIGN

DeSiNe is developed to study QoS and traffic engineering in large networks. DeSiNe models the traffic streams as flows and assumes fixed flow rates and independence between flows, i.e. the arrival or departure of a flow has no effect on the already existing flows. The motivation for taking fixed flow rates is two-fold. Firstly, one of the purposes of DeSiNe is to investigate QoS technology, which implies resource reservations. When the required resources are reserved for a particular flow, they remain unchanged during its lifetime. Secondly, fixed flow rates prevent from running into the scalability problems encountered from variable flow rates arising from feedback-based protocols like TCP. Additionally, the homogeneity and high level of abstraction make DeSiNe easy to use and configure, and suitable for simulation of large networks with many flows. The processing and memory requirements of DeSiNe scale linearly with the number of flows. Figures 1(a) and 1(b) show the CPU time required for simulations with a given number of nodes and flows, respectively. Several types of topologies have been used (Barabási-Albert, Erdös–Renyí random graphs, and lattices), with varying numbers of nodes, up to 1000. We also vary the number of flows, up to 100,000. We observe that the simulation time scales linearly with the number of flows.

The scaling of the simulation time as a function of nodes is not



(a) Scalability with respect to the network size.



(b) Scalability with respect to the number of flows.

Figure 1: Scalability of DeSiNe.

linear because of different link densities of the topologies, as well as different computation times for the Dijkstra algorithm on different types of topologies. Note that unit weights are used on the links.

Moreover, DeSiNe is useful when examining "classes" of networks with specific properties, e.g. Erdös–Renyí random graphs, lattices and scale-free graphs. Such studies require automated generation of many randomized graph realizations. Implementation and incorporation of additional routing protocols, topology generators and link-state update policies is easy and straightforward.

A crucial factor in QoS and traffic engineering is the knowledge about the actual state of the network. Routing algorithms with traffic engineering extensions rely on trustworthy information on the actual available network resources in the path computation process, which is of vital importance to guarantee the QoS requirements of each flow. The acquisition and distribution of network information is a task of the *link-state update policy* (LSUP). As real-time link-state monitoring and link-state advertisements are costly in terms of network resources, a trade-off must be made between the accuracy and overhead associated with the link-state updates. Based on the network information, the *routing algorithm* computes the path. The routing algorithm computes the best path according to an optimality criterion. In QoS-aware networks, the path must obey a set of QoS requirements and can thereby include or exclude a certain objective function. In traffic engineering, the paths assigned to flows try to optimize a local or global objective, i.e. prevent congestion on links or minimize the blocking of future flows. We refer to [2, 3] for a detailed discussion on QoS routing and to [21, 22, 23] for on-line traffic engineering algorithms. Sections 3.1 and 3.2 present the network model and functional design of the simulator, respectively. Section 3.3 presents an overview of the structural design of DeSiNe and Section 3.4 gives an overview of the features that have been implemented in DeSiNe.

3.1 Network Model

A network is modeled as a graph $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where $\mathcal{N}$ denotes the set of nodes and $\mathcal{L}$ denotes the set of links connecting the nodes. The number of nodes is $N = |\mathcal{N}|$ and likewise the number of links equals $L = |\mathcal{L}|$. A subset of the nodes can serve as source or destination nodes, while other nodes only act as internal nodes or intermediate hops. Each link is assigned a positive value: the *maximum capacity*, which expresses the maximum amount of traffic the link can carry. The actual utilization of the link is maintained by two values. The first value always contains the true *available capacity* of the link. The second value is the *reservable capacity* that is advertised by the routers and used in the routing process. The link-state update policy synchronizes these values. In case the available and reservable capacity are not synchronized, the link-state information is inaccurate or *stale*.

Each link is also associated with a number of QoS weights related to additive QoS measures (e.g., delay and jitter). Additive measures are such that their value along a path is the sum of the values associated with each constituting link [2, Chap. 12]. Multiplicative QoS measures such as the probability of packet loss can be dealt with by taking the logarithm. The QoS link weights are used by QoS routing algorithms to compute feasible paths, as described in Section 3.4. We allow for static and dynamic link weights. In case of dynamic link weights, the weight can be a function of the throughput of the link. All links are either directed, allowing traffic in a single direction, or undirected, in which case the traffic flows in both directions.

The traffic in the network is modeled by a set $\mathcal{F}$ of flows, where $F = \|\mathcal{F}\|$ denotes the number of flows. A flow $f \in \mathcal{F}$ represents a packet-stream flowing from a source $n_s \in \mathcal{N}$ to a destination $n_d \in \mathcal{N}$. To each flow, a set of QoS requirements can be assigned that must be met and guaranteed by the network during the life-time of the flow. The QoS requirements may consist of a single metric, e.g. capacity, or a set of metrics, e.g. delay, jitter, packet loss, etc. When the flow is set up, the required capacity along the path is reserved exclusively to the flow. The resources remain reserved during the full life-time of the flow to guarantee the QoS requirements.

3.2 Event triggering

DeSiNe is triggered by the arrival of a flow. When a flow f_j arrives at the network at time t_j, all events in the network that have occurred after the arrival of flow f_{j-1} and prior to t_j are evaluated and processed at the epoch of the new arrival. Figure 2 schematically illustrates the arrival of flow f_j at time t_j. The last arrival was at t_{j-1}. All events between t_{j-1} and t_j, such as flow departures and link-state updates, must be processed before flow f_j is routed and allocated in the network. Figure 2, provides an example where the link state is displayed schematically from time t_{j-1} to t_j. Between t_{j-1} and t_j one flow departs at time s_{j-x} and the value of the reservable capacity, i.e. the capacity that is advertised by the routers, is updated at T_n.

Figure 3 depicts the flowchart diagram of DeSiNe. The simulation begins with the initialization of the network. The network topology can be generated, e.g. random graph, or read from a file. After the network is created, the simulation enters a loop that is re-

peated for each flow arrival, until the last flow has been generated. The loop begins in step 1 with the generation of a flow. The distribution of each property, such as the flow duration or inter-arrival time, is configured at the start of the simulation. The source and destination nodes are chosen uniformly from the set of valid endpoints. In step 2 the network is scanned for expired flows and time-based (see Section 3.4) link-state updates. If a time-based link-state update policy is used, then the reservable capacity is updated with the available capacity value, as illustrated in Figure 2. Next, the flow is routed in step 3. If no link-state update policy is used, and all link-state values are updated instantaneously, the reservable capacity equals the actual capacity. The links with insufficient capacity are pruned from the topology. The routing algorithms described hereafter may assign a cost to each link and select the shortest length path, the length of a path being the sum of the costs of the constituting links. Link costs may be specified as a function of the capacity and of the QoS weights. Several different functions are pre-defined in DeSiNe. If the routing is successful and a feasible path is found, the resources are reserved along the path (step 4). In case the routing fails, or when the routing is successful, but the reservation fails, the flow is rejected by the network and discarded. The reservation may fail due to stale link-state information, such that the advertised link capacity is more than the actual available capacity. If the reservation is successful, the flow is allocated in the network in step 5. The advertised link weights are consequently updated in step 6. Finally, in step 7, the loop returns to step 2 to the next flow arrival. If no new arrivals are queued, the simulations ends.

3.3 Structural design

To enhance the extensibility of DeSiNe, a modular approach has been used. The modules are grouped based on their function and new modules can be added and removed independently of each other. The modules can be classified according to Figure 4, where we have omitted a few utility modules. The module *Main* controls the flows and performs the data collection, which is written to disk with use of the *IO* (Input/Output) module. Based on the parameters, the *Network* is initialized, holding the actual topology with the traffic information. If the topology is stored on disk, the *IO* module is used to read the corresponding file. Likewise, a routing algorithm is selected from the *Algorithms* module. During the simulation, the *Network* module uses the *LinkStateUpdate* and *LinkCostFunction* to perform link-state advertisements and compute actual link cost, respectively. The module *Random* implements random-number generator including several well-known distributions.

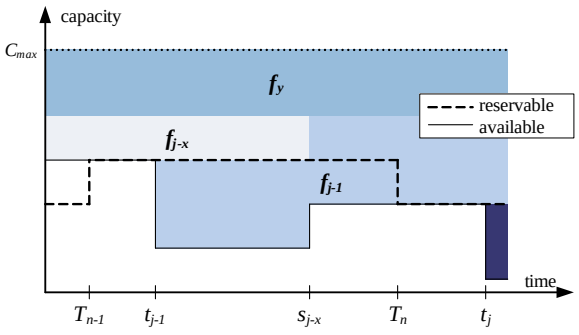


Figure 2: Example of the link state, showing the evolution of the available and reservable capacity subject to flow-arrivals and departures. Each colored block corresponds to one flow. First, the advertised link-state value is updated at T_{n-1}. Subsequently, flow f_{j-1} arrives at t_{j-1}. Next, flow f_{j-x} expires at time s_{j-x}. Then, the next periodical update takes place at T_n. Finally, flow f_j arrives at t_j. Before flow f_j is processed, the departure at s_{j-x} and the link-state update at T_n are processed.

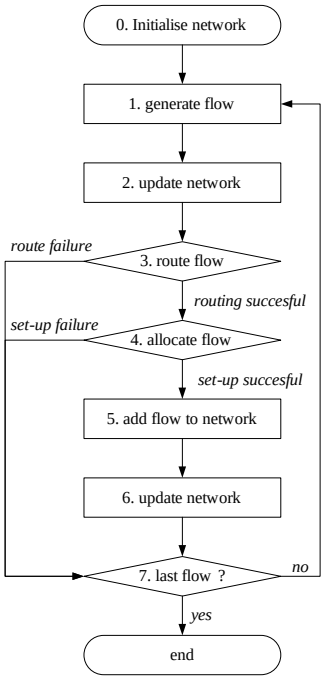


Figure 3: Functional view of the DeSine.

3.4 Functional design

In this section we will briefly discuss the QoS routing algorithms implemented in DeSiNe, as well as the link-state update policies. We differentiate here between routing algorithms that only consider "static" topology information during path computation and routing algorithms that compute the path based on the actual traffic conditions.

QoS routing algorithms solve the Multi-constrained Path Problem (MCP) [3]. The MCP problem consists of finding a path that satisfies m constraints. When the path meets all the constraints, it is said to be feasible. The problem of finding the shortest length path, given a definition of path length, among the feasible ones is known as the Multi-constrained Optimal Path Problem (MCOP). Traffic Engineering algorithms can be considered a sub-class of QoS routing algorithms, without multiple constraints.

The multi-constrained routing algorithms that have been implemented in DeSiNe are: SAMCRA, the self-adaptive multiple constraints routing algorithm [3] and TE-DB [22]. SAMCRA is proposed by Van Mieghem *et al.* [3] and exactly solves the MCOP problem. Banjeree *et al.* [22] proposed the TE-DB algorithm which uses TAMCRA [24], a predecessor of SAMCRA, and the max-flow concept used in MIRA.

The routing algorithms that incorporate TE extensions utilize the link-state information provided by the link-state update policy to find the optimal path. The link-state information consists of a set of metrics that describe the (transient) state of the link. Various TE algorithms have been implemented in DeSiNe: several variations of MIRA [23], New MIRA [21], and SMIRA, the simple minimum interference routing algorithm [25]. MIRA takes

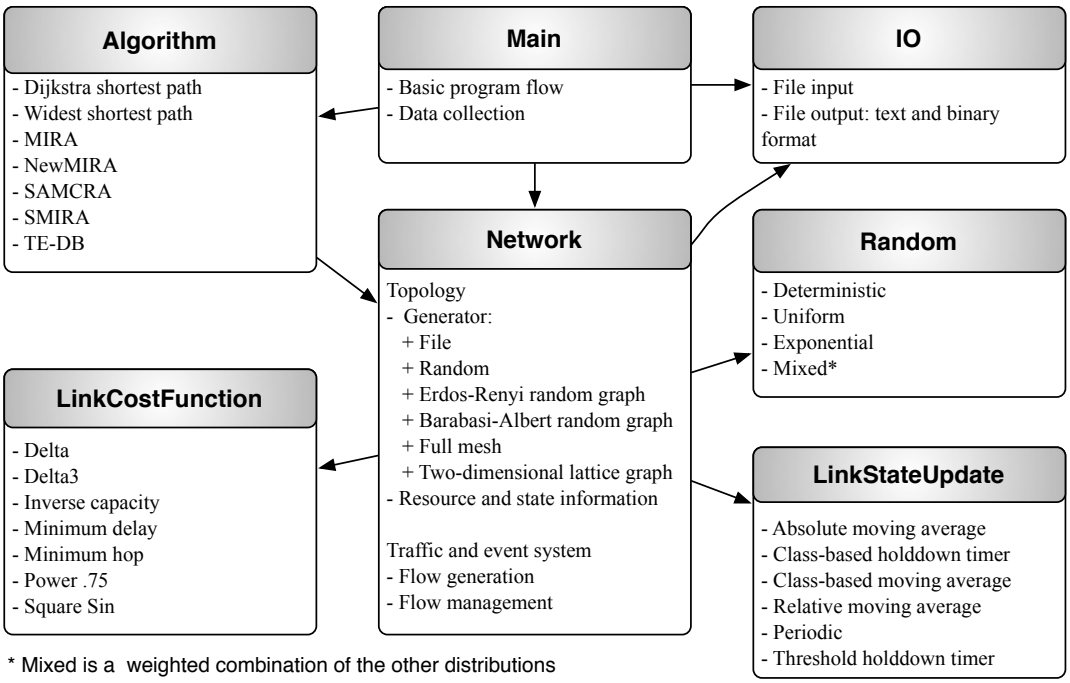


Figure 4: The modular design of DeSine. For each module, its core functionality and features are summarized.

into account the information of ingress-egress pairs (S_i, D_i) and weights them by their importance α_i. To minimize the interference between source-destination pairs, these algorithms maximize the sum of the residual weighted max-flows[1] between all ingress-egress pairs. Upon the arrival of a connection request from S_i to D_i, the algorithms compute the sets of *critical* links L_j for all other pairs $(S_j, D_j)(j \neq i)$. A link l is called *critical* for an ingress-egress pair (S, D), if the reduction in l's capacity leads to the reduction in (S, D)'s max-flow. Then link costs are set according to the link *criticality*, and shortest path is applied on this "properly" weighted topology. Different weighting methods lead to variations of MIRA. The resulting path is called the *minimum interference path*. For further details about MIRA we refer to [23]. The New MIRA and SMIRA algorithms seek to improve MIRA in computation complexity or weighting methods. For further details about New MIRA and SMIRA, we refer to [21] and [25] respectively. The above routing algorithms have all been evaluated in [26].

The link-state update policy is responsible for monitoring the available capacity of each link and the distribution of that information across the network in conformance with the policy. The update policies can be classified into three categories [27, 28]: time-based, trigger-based and window-based. With time-based policies updates on the link-state information are triggered by the actual time. The time-based update policy implemented in DeSiNe is the *periodic update policy*, where updates are sent at fixed time intervals.

Trigger-based policies monitor the real link-state information and send updates when some conditions are met. Two trigger-based policies are implemented: threshold-based and class-based. With threshold-based policies, updates are triggered when the relative difference between the current link state and the advertised link state exceeds a certain threshold. A class-based policy divides the link capacity into a set of classes. When a class boundary is crossed, an update is triggered.

Finally, the window-based policies are used in combination with trigger-based policies and circumvent the problem when the link utilization oscillates around a class boundary or when the traffic is very bursty (threshold-based). Two window-based policies are currently available: hold-down timer and moving-average. The hold-down timer is used to set a minimal spacing between two successive updates. The moving-average takes the average link utilization over a fixed-size time-window and based on this average an update event may be triggered.

4. APPLICATIONS OF DESINE

DeSiNe is able to simulate the performance of various routing algorithms and link-state update policies under different network and traffic scenarios. It can be used to evaluate new routing algorithms or link-state update policies by comparing them with the existing ones under the same network and traffic conditions. Due to its modular design, custom routing algorithms and link-state update policies can be easily added to the simulator and evaluated thereafter. Network and traffic, under which routing algorithms or link-state update policies will be evaluated, can be configured in flexible ways. Users can define their own performance metrics, e.g. the maximum link utilization, the sum of max-flows, etc., and set them as output.

In this section, we illustrate the use of the simulator on different kinds of scenarios with selected routing algorithms, link-state update policies, network and traffic scenarios, and performance metrics, to show some of the features of DeSiNe. In Section 4.1, we compare the performance of different link-state update policies and routing algorithms, where the flows come and leave according to certain processes. We use a number of flows to reach a steady state, and then take statistical results. In Section 4.2, we show the performance of different traffic engineering algorithms. Moreover, we study the performance of classes of networks by using randomized samples of such a class. The results are presented in Section 4.3,

[1]The max-flow for an ingress-egress pair (S, D) is the maximum amount of traffic that can be pushed between this pair. Multiple link-disjoint paths may be used.

where we study the Erdös-Renyí and Barabási-Albert classes of networks.

4.1 Link state updates

We compare, for a given topology, the performance of different link-state update policies and routing algorithms. We use the MCI topology. The MCI topology consists of 19 nodes, out of which 11 nodes are edge nodes and the remaining 8 are core nodes, connected by 33 links. Each link is bidirectional and assigned 600MB/s capacity. A bidirectional link acts as two unidirectional links both with the assigned capacity. The setting of the scenario is the same as in [29].

The arrival process of the incoming traffic flows is modeled as a Poisson process with rate λ flows per second. Source-destination pairs are uniformly selected among the set of edge nodes. The service time of flows, i.e. the flow duration, is described by an exponentially distributed random variable d with mean 10 seconds. We denote by C_r the capacity requirement of each flow, which is uniformly distributed within $[15, 45]$ MB/s.

Following [29], the network load is defined as:

$$\rho_N = \lambda \mathrm{E}[d]\,\mathrm{E}[C_r]\,\mathrm{E}[h]\,/L,$$

where $\mathrm{E}[d]$ is the mean flow duration, $\mathrm{E}[C_r]$ is the mean capacity requirement, $\mathrm{E}[h]$ is the mean hop count of the shortest paths between all pairs of source and destination nodes, and L is the number of links in the network.

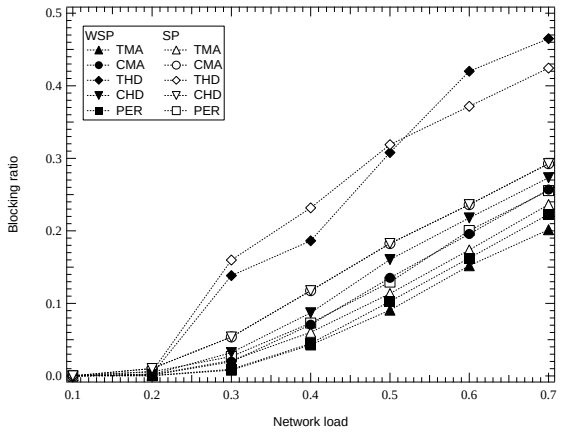
The comparison of several LSUPs and routing algorithms is given in Figures 5(a) and 5(b). Dijkstra's shortest path and widest shortest path [2, 30] routing algorithms are selected both with the Min-Hop link weight function. For each LSUP, we choose one set of parameters for the simulations. For example, for the threshold based moving average LSUP, we set the threshold to be 0.2 and the window size to be 5.

Figure 5(a) shows how each combination of LSUP and routing algorithm behaves as a function of the network load. Most of the evaluated LSUPs give a better behavior under the widest shortest path routing algorithm than under the shortest path routing algorithm. Widest shortest path is known to be better in load-balancing traffic across the network than shortest-path routing.

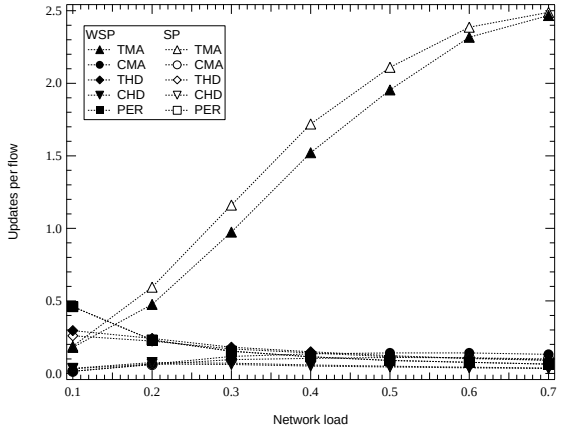
The updates per flow shown in Figure 5(b) estimate the protocol overhead. The moving average LSUPs adapts to the traffic density. As the network load increases, the update rate for the moving average LSUPs increases as well. The time sensitive LSUPs set limitations on the period between 2 updates. As we only tune λ in order to get different network loads, the higher the network load, the smaller the flow arrival interval. The periodical LSUP generates less updates with a higher network load, because under a higher network load there are more flows in a period. The other time sensitive LSUPs take both the time and trigger conditions into account, and give smooth curves for the update rate.

4.2 Routing with traffic engineering extensions

The scenario in this section illustrates the performance of different routing algorithms with traffic engineering extensions. We implemented the scenario of [23]. The network consists of 15 nodes and 28 bidirectional links. The capacity of core links is 4800 units, and that of the other links is 1200 units. Traffic transits between 4 pairs of ingress-egress nodes. Flows with their capacity requirements following the same uniform distribution between 1 unit and 3 units, arrive in a Poissonian manner with the same mean rate for all these 4 pairs. Once a flow is routed and setup successfully in the network, it will never leave the network. We computed after each request the residual max-flow of each ingress-egress pair.



(a) Performance measured as the ratio of blocked flow requests with respect to the total requests.



(b) The average number link updates due to the arrival and departure of one flow.

Figure 5: Simulation results for the MCI topology under varying network load, where the shortest-path (SP) and widest-shortest-path (WSP) routing algorithms are used in conjunction with the threshold-based moving average (TMA), class-based moving average (CMA), threshold-based hold-down timer (THD), class-based hold-down timer (CHD) and periodical (PER) link-state update policies.

Figure 6 shows the results of 1 of the 4 ingress-egress pairs used in [23], (S_1, D_1) in their topology. We have selected the following routing algorithms in Figure 6: static shortest path (SSP), widest shortest path (WSP), dynamic shortest path with link weights being the inverse of residual link capacity (DSP-Inv), and two variants of MIRA (MIRA-TM and MIRA-TM-Cap). MIRA-TM considers in its weighting the traffic between ingress-egress pairs. As all the ingress-egress pairs have the same amount of traffic, MIRA-TM in this scenario is identical to SMIRA used in [23]. MIRA-TM-Cap takes into account not only the traffic between ingress-egress pairs, but also the residual capacity of the critical links.

The x-axis of Figure 6 gives the sum of the connection requests that have been successfully routed in the network so far (total traffic accepted). We scaled this sum to quantify the traffic demands. The y-axis gives the normalized residual max-flow for (S_1, D_1). We observe on Figure 6 that the two MIRA variants perform better than the other routing algorithms, because MIRA aims at maximiz-

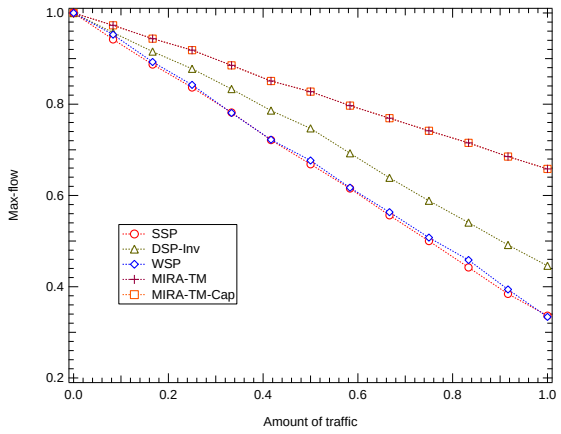


Figure 6: Residual max-flow for pair (S_1, D_1)

ing the residual max-flow between any pair of source-destination pairs. SSP and WSP perform as badly. DSP-Inv that computes dynamic shortest paths depending on the inverse of the residual capacity performs better than both SSP and WSP. The two MIRA variants perform best as they are designed to maximize the residual max-flow on all source-destination pairs. We also observe that the two MIRA variants lead to identical residual max-flow, as in this case the min-cut of this pair does not allow to load-balance traffic for pair (S_1, D_1) without impeding on the max-flow of other source-destination pairs. The ability of traffic engineering traffic differs among the source-destination pairs. Sometimes, alternate paths that do not interfere with other pairs do not exist in the network. Then, traffic engineering cannot help much in load-balancing traffic between specific source-destination pairs since it will impede on other source-destination paris in the network. Note that the results we obtain are consistent with those from [23].

4.3 Random Networks

Online generation of random networks is useful when comparing the performance of a particular routing algorithm or link-state update strategy between different classes of networks. Measures of interest here can be the average hopcount, flow blocking or number of updates. DeSiNe features several built-in topology generators (see Figure 4).

Tables 1 and 2 show the results of several example scenarios using Erdös–Renyí and Barabási-Albert graphs. Table 1 presents the results when using periodic link-state updates. As expected, the number of updates grows with the network size. Furthermore, Table 1 shows that Barabási-Albert graphs cannot sustain as much traffic as Erdös–Renyí graphs, for a comparable link density.

Blocked flows are those that cannot be routed due to a lack of resources in the network. The rejected flows are the flows for which routing is successful, but the setup fails due to stale link-state information. In Table 2 the same networks are used, but now the time-based moving average link-state update policy is used. Clearly, the number of updates is drastically decreased while throughput is improved at the same time.

5. CONCLUSIONS

In this paper, we have presented DeSiNe, a flow-level network simulator. Compared to existing simulators, DeSiNe incorporates in a unique fashion Quality of Service (QoS) routing, traffic engineering and scalability. It provides an end-to-end solution from

	N	E $[h]$	E $[F_{acc}]$	E $[F_{bl}]$	E $[F_{rej}]$	E $[u]$ $/F_{tot}$
ER	100	3.77	391621	551	7828	120
	200	4.16	396352	189	3459	240
	300	4.37	397917	86	1996	363
	400	4.55	398554	64	1381	482
	500	4.68	398942	38	1019	601
BA	100	3.61	391164	51	8784	117
	200	3.89	395601	5	4393	237
	300	4.04	397035	1	2963	357
	400	4.16	397795	0	2204	478
	500	4.23	398188	0	1811	498

Table 1: Simulation results for Erdös-Renyí (ER) and Barabási-Albert (BA) random graphs with varying N nodes, and comparable link densities; for BA $m = m_0 = 3$ and for ER $p = \frac{2m}{N}$. The results are the averages computed over 100 random graph realizations and $F_{tot} = 400.000$ flow arrivals. The widest-shortest path routing algorithm with periodic LSUP has been used with window size $w = 50$. In the table, h is the hopcount, F_{acc}, F_{bl}, F_{rej} the accepted, blocked and rejected number of flows, respectively, and u is the number of link-state updates.

	N	E $[h]$	E $[F_{acc}]$	E $[F_{bl}]$	E $[F_{rej}]$	E $[u]$ $/F_{tot}$
ER	100	3.76	399438	369	192	9.7
	200	4.16	399837	98	65	10.6
	300	4.39	399915	47	36	11.1
	400	4.54	399946	30	23	11.4
	500	4.67	399963	20	16	11.7
BA	100	3.11	399972	11	17	6.85
	200	3.36	399992	2	6	7.43
	300	3.49	399996	0	4	7.80
	400	3.58	399997	0	3	8.06
	500	3.65	399997	0	3	8.28

Table 2: Simulation results for Erdös-Renyí (ER) and Barabási-Albert (BA) random graphs with varying N nodes, and comparable link densities; for BA $m = m_0 = 3$ and for ER $p = \frac{2m}{N}$. The results are the averages computed over 100 random graph realizations and $F_{tot} = 400.000$ flow arrivals. The widest-shortest path routing algorithm has been used with time-based moving average LSUP with window size $w = 50$ and threshold 0.1. The symbols correspond to those of Table 1.

topology generation to simulation and data collection. This makes DeSiNe well-suited for the performance analysis at system-level of new QoS routing and traffic engineering algorithms in any network topology. DeSiNe has been written in C++ and is modularly built, with extensibility as a major design goal. New algorithms, protocols or other techniques can easily be incorporated into the existing simulator. Moreover, DeSiNe is not built on top of any simulation framework or libraries, which minimizes undefined behavior and dependency issues. The source code is publicly available at the Networking, Architectures and Services group website [31] under the section *Research*.

6. ACKNOWLEDGEMENT

The work is funded by the STW foundation as part of the Network Dynamics and QoS project (DTC.6421) and by the European Union NoE CONTENT (FP6-IST-038423). The authors wish to acknowledge Steve Uhlig for the useful discussions and valuable contributions.

7. REFERENCES

[1] D. Alderson, H. Chang, M. Roughan, S. Uhlig, and W. Willinger, "The many facets of internet topology and traffic," *Networks and Heterogenous Media*, vol. 1, no. 4, pp. 569–600, 2006.

[2] P. Van Mieghem, *Data Communications Networking*, Techne Press, Amsterdam, 2006.

[3] P. Van Mieghem and F. A. Kuipers, "Concepts of exact QoS routing algorithms," *IEEE/ACM Transactions on Networking*, vol. 12, no. 5, pp. 851–864, Oct. 2004.

[4] ns-2, "http://www.isi.edu/nsnam/ns/," .

[5] SSFNet, "http://www.ssfnet.org/," .

[6] D. Anick, D. Mitra, and M. M. Sondhi, "Stochastic theory of a data-handling system and multiple sources," *The Bell System Technical Journal*, vol. 61, no. 8, pp. 1871–1894, 1982.

[7] J. S. Ahn and P. B. Danzig, "Packet network simulation: Speedup and accuracy versus timing granularity," *IEEE/ACM Transactions on Networking*, vol. 4, no. 5, pp. 743–757, 1996.

[8] G. Kesidis, A. Singh, D. Cheung, and W. Kwok, "Feasibility of fluid event-driven simulation for ATM networks," in *IEEE GLOBECOM*, 1996.

[9] V. Misra, W-B. Gong, and D. F. Towsley, "Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED," in *SIGCOMM*, 2000, pp. 151–160.

[10] B. Liu, Y. Guo, J. F. Kurose, D. F. Towsley, and W. Gong, "Fluid simulation of large scale networks: Issues and tradeoffs," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, Nevada, USA, 1999, pp. 2136–2142, CSREA Press.

[11] A. Yan and W-B. Gong, "Time-driven fluid simulation for high-speed networks," *IEEE Transactions on Information Theory*, vol. 45, no. 5, pp. 1588–1599, 1999.

[12] A. Kavimandan, W. Lee, M. Thottan, A. Gokhale, and R. Viswanathan, "Network simulation via hybrid system modeling: A time-stepped approach," in *14th International Conference on Computer Communications And Networks (ICCCN)*, San Diego, CA, 2005.

[13] D. M. Nicol, J. Liu, M. Liljenstam, and G. Yan, "Simulation of large scale networks I: Simulation of large-scale networks using SSF," in *Winter Simulation Conference*, 2003, pp. 650–657.

[14] T. Yung, J. Martin, M. Takai, and R. Bagrodia, "Integration of fluid-based analytical model with packet-level simulation for analysis of computer networks," in *SPIE*, R. D. van der Mei and F. Huebner-Szabo de Bucs, Eds., 2001, vol. 4523 of *Internet Performance and Control of Network Systems II*, pp. 130–143.

[15] B. Melamed, S. Pan, and Y. Wardi, "Hybrid discrete-continuous fluid-flow simulation," in *SPIE International Symposium on Information Technologies and Communications (ITCOM 01), Scalability and Traffic Control in IP Networks*, Denver, CO, 2001, pp. 263–270.

[16] C. Kiddle, R. Simmonds, C. L. Williamson, and B. Unger, "Hybrid packet/fluid flow network simulation," in *17th Workshop on Parallel and Distributed Simulation, PADS*, San Diego, CA, USA, 2003, pp. 143–152, ACM.

[17] P. Zhang, R. Kantola, and Z. Ma, "Design and implementation of a new routing simulator," in *SCS Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, Vancouver, Canada, July 2000.

[18] P. Zhang, Z. Ma, and R. Kantola, "Designing a new routing simulator for DiffServ MPLS networks," in *SCS Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, Orlando, Florida, USA, July 2001.

[19] A. Shaikh, "Management of routing protocols in IP networks," PhD Dissertation, University of California, Santa Cruz (UCSC), Dec. 2003.

[20] R. J. Sivasankar, S. Ramam, S. P. Subrahmaniam, T. Srinivasa Rao, and D. Medhi, "Some studies on the impact of dynamic traffic in QoS-based dynamic routing environment," in *ICC*, 2000, pp. 959–963.

[21] B. Wang, X. Su, and C. L. P. Chen, "A new bandwidth guaranteed routing algorithm for MPLS traffic engineering," in *IEEE International Conference on Communications (ICC)*, May 2002.

[22] G. Banerjee and D. Sidhu, "Comparative analysis of path computation techniques for MPLS traffic engineering," *Computer Networks*, vol. 40, no. 1, pp. 149–165, 2002.

[23] K. Kar, M. Kodialam, and T. V. Lakshman, "Minimum interference routing of bandwidth guaranteed tunnels with MPLS traffic engineering applications," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 2566–2579, Dec. 2000.

[24] H. De Neve and P. Van Mieghem, "TAMCRA: A tunable accuracy multiple constraints routing algorithm," *Computer Communications*, vol. 23, pp. 667–679, 2000.

[25] I. Iliadis and D. Bauer, "A new class of online minimum-interference routing algorithms," in *NETWORKING*. May 2002, vol. 2345 of *Lecture Notes in Computer Science*, pp. 959–971, Springer.

[26] S. Avallone, F.A. Kuipers, G. Ventre, and P. Van Mieghem, "Dynamic routing in QoS-aware traffic engineered networks," in *EUNICE*, Colmenarejo, Spain, July 2005, pp. 222–228.

[27] G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi, "Quality of service based routing: A performance perspective," in *SIGCOMM*, 1998, pp. 17–28.

[28] B. Lekovic and P. Van Mieghem, "Link state update policies for quality of service routing," in *IEEE 8th Symposium on Communications and Vehicular Technology in the Benelux (SCVT)*, Delft, The Netherlands, Oct. 2001, pp. 123–128.

[29] A. Shaikh, J. Redford, and K. G. Shin, "Evaluating the impact of stale link state on quality-of-service routing," *IEEE/ACM Transactions on Networking*, vol. 06, no. 02, pp. 162–176, Apr. 2001.

[30] R. A. Guerin, A. Orda, and D. Williams, "QoS routing mechanisms and OSPF extensions," in *IEEE GLOBECOM*, 1997.

[31] Networking, Architectures and Services group, "http://www.nas.ewi.tudelft.nl/software/desine.html," .