

### 3 Description of MesoNetHS

MesoNetHS is a medium-scale (i.e., mesoscopic) packet-level simulation, which represents transport flows that regulate the generation of packets, routers that queue (or discard), forward and transmit packets, and links that subject packets to propagation delays. MesoNetHS achieves scale reduction by eliminating packet sizes (expressing capacities in terms of integral packets) and by simplifying routers to share forwarding capacity across all attached outgoing links. Each backbone router contains a single, drop-tail queue of finite length. Each lower-level router contains two, finite-length, drop-tail queues – one for packets moving toward the network core and one for packets moving toward the network edge. These simplifications prevent MesoNetHS from providing precise quantitative estimates but enable qualitative understanding of relationships driving behavior in networks of reasonably large size: hundreds of routers transporting hundreds of thousands of flows. In what follows, we explain the structure of MesoNetHS and then describe how to configure the model for particular simulations. We also define measurements that the model can produce, including the ability to provide detailed traces of flow state and packet exchanges. We close with a brief discussion of how MesoNetHS is coded in SLX [81-82]; a discussion intended to aid those who wish to review the model's source code.

#### 3.1 Model Structure

MesoNetHS was motivated by finite-element, discrete-time (FEDT) models, also called cellular-automata (CA) models [72a], which are often used by physical scientists when attempting to understand general relationships that drive spatiotemporal evolution in complex physical systems, such as collections of particles and structural arrangements of chemicals in materials. In CA models, the state of each element is updated at each discrete time step. MesoNetHS was originally designed and implemented as a CA model. In this form, MesoNetHS worked fairly well for networks of limited size (on the order of up to 30,000 sources) and forwarding speed (on the order of 13 million packets per second). However, when MesoNetHS simulated networks of larger size (hundreds of thousands of sources) and faster speed (hundreds of millions of packets per second), the CA modeling foundation proved quite inefficient; simulating 1.5 million time steps of network evolution could take more than two weeks on contemporary (circa 2007) PC-based servers. A CA model can require substantial useless processing when elements within the model have little to do for relatively long periods of simulated time. While the CA version of MesoNetHS was constructed as efficiently as possible, expanding the size and simulated speed of model elements produced a model with too much overhead. Ultimately, for models of a size and speed of interest for the studies reported in this document, the CA version of MesoNetHS needed about 5 CPU seconds to simulate 6 time steps of network evolution.

MesoNetHS was subsequently recast into a discrete-event simulation (DES) model [90]. This means that model elements (i.e., all elements except for simulated flows and packets, which are transient) persist for the duration of the simulation and that the state of each model element is updated at varying time intervals dictated by the global sequence of events, where each discrete event is generated by some model element. A DES model skips over idle time and processes only when events occur. As a result,

processing speeds increase and the increase in processing speed (over the CA version of MesoNetHS) increases with model size and simulated network speed. For models of a size and speed used in the current study, the DES version of MesoNetHS needed about 5 CPU seconds to simulate 26 time steps of network evolution; thus, 1.5 million time steps of simulated network evolution could be computed in about 4 days (instead of 16 days). To achieve this speedup, the DES version of MesoNetHS requires about double the memory of the CA version.

In what follows, we introduce the six fundamental elements of MesoNetHS and then describe how these elements are structured into a simulated network topology. We also introduce our representation of simulated packets, which are the mechanism through which MesoNetHS elements interact. Packets are originated by sources and receivers, flow through a topology of routers and links and then cease to exist. We close with a brief discussion of our DES model, including one means to relate abstract time to the typical notion of real time in networks.

### 3.1.1 Model Elements

The persistent elements of MesoNetHS interact through two types of transient elements: packets and flows. Packets are the fundamental messages that move among the model's persistent elements. Flows are collections of packets that move between specific sources and receivers over a defined time period. Flows then represent logically related spatiotemporal sequences of packets. The operation of each flow is controlled jointly by three elements: a source, a receiver and an access router (to which the source is attached). Each source generates data packets, subject to various constraints. Each receiver generates acknowledgment packets as required to indicate reception of expected data. Each receiver also generates negative acknowledgment packets to indicate that expected data was missed. Access routers update appropriate state variables for subordinate sources based upon information in arriving acknowledgments and negative acknowledgments. Access routers, point-of-presence (POP) routers and backbone routers manage the forwarding of packets. The simulated propagation of packets over distance is achieved through backbone links. Below, we describe the detailed operation of each of these element classes.

*3.1.1.1 Sources.* MesoNetHS sources represent user behaviors associated with data sources. Sources alternate between ON and OFF periods. During each ON period, a source generates data packets under regulation of congestion-control rules implemented by a specific, simulated transport-level protocol. During each OFF period, a source simply waits for simulated time to advance to the beginning of the next ON period. The alternation of a source between ON and OFF periods simulates a user surfing the Web and downloading selected files of interest. Initially, sources are placed into the ON state with a specified probability. Sources that start in the OFF state are scheduled to enter the ON state at some random time after the model starts execution.

Upon entering the ON state, a source selects a file size, which designates the number of data packets to transfer to a receiver. The source remains in the ON state until the receiver has acknowledged the required number of data packets. The file size is selected from a distribution (typically a Pareto distribution with a designated mean and shape) that represents the aggregate size of objects that might typically be downloaded to

display a Web page. With some probability, a given file size is multiplied by a factor, which represents the situation where a Web-surfing user decides to download a particular file (e.g., document, service pack or movie).

After selecting a file size, a source selects a specific receiver with which to exchange the file. The source selects a receiver associated with a different backbone router than the source. This ensures that each simulated flow will transit the network backbone. Each eligible backbone (first-tier) router has a uniform probability of being selected. Once a backbone router is selected, the source chooses among one of the second-tier routers beneath the backbone router. For this selection, the probability of choosing a particular second-tier router is proportional to the number of receivers encompassed by each eligible router. If a selected second-tier router is a point-of-presence (POP) router, then there will be third-tier (access) routers beneath it. In this case, the source selects a specific access router. For this selection, the probability of choosing a particular third-tier router is proportional to the number of receivers encompassed by each eligible router. Once a specific access router is determined, the source chooses any idle receiver encompassed by the access router. If no receiver is idle, then the source conducts another selection process, beginning with selection of a backbone router. The selection process continues until the source finds an available receiver. If all model receivers are engaged (which should not occur as long as the topology contains more receivers than sources), then the model will enter an infinite loop.

After selecting a receiver, a source initiates a connection process, intended to simulate the establishment of transmission-control protocol (TCP) connections [6, 9-10]. Without including a connection process, the model would tend to generate congestion patterns inconsistent with real TCP traffic, which must establish a connection before attempting to exchange data. MesoNetHS simulates a two-way handshake, with behavior patterned after TCP implementations deployed in Microsoft Windows™ operating systems [12]. The source sends a SYN packet and then waits 3000 time steps for a connection to be established. If a connection is not established, then the source sends a second SYN and waits 6000 time steps for a connection to be established. If a connection is not established, then the source sends a third SYN packet and waits 12000 time steps for a connection to be established. If no connection is established after 21000 time steps, then the connection is declared to have failed and the source enters the OFF state after selecting the next time for the source to reenter the ON state.

After successfully establishing a connection, a source enters the CONNECTED state and begins to transfer data packets. Upon initially entering the CONNECTED state, a source checks the time step and flow class (see 3.1.2.3 below) to determine whether the file size should be increased. MesoNetHS permits qualified flows to be designated as jumbo flows during particular time periods. The size of a jumbo flow is multiplied by an additional factor, intended to simulate the exchange of large data sets between research organizations. Upon transferring the first data packet of a file, the flow state is initialized – consistent with requirements of the particular transport protocol used on the flow. For all flows, both the congestion window and slow-start threshold are set to initial values and the flow is placed into the SLOW-START phase. The flow also establishes an initial estimate for the smoothed round-trip time (SRTT) between the source and receiver. This initial estimate is derived from the network topology used in the simulation (see 3.1.2

below). The first packet is transferred to the access router that encompasses the source. If the incoming queue in the access router is full, then the data packet is dropped.

After transferring the initial data packet of a flow, a source enters its main processing mode that attempts to transfer data packets whenever permitted and until all required data packets have been sent. The source maintains a retransmission timeout (CRTO) that allows it to recover from situations where required feedback does not arrive. The first step of processing is to determine if the retransmission timer has expired. If so, following rules associated with a specific transport protocol, the source reduces the slow-start threshold and resets the congestion window to its initial value. The source also doubles and resets the retransmission timeout and then reenters the SLOW-START phase.

Before sending additional data packets, a source determines how many data packets it may send. Roughly, this is the congestion window minus the number of data packets sent but not yet acknowledged by the receiver. Of course, the number of data packets to be sent must also be capped by the file size for the flow. If any data packets may be sent, then the source transfers one data packet to the access router that encompasses the source. If the incoming queue in the access router is full, then the data packet is dropped. Sending multiple data packets requires multiple invocations of a source. Thus, the number of packets that a source may send per time step is capped by the source's capacity (in packets per time step, or ppts), which is established by a configuration parameter (see 3.2.4 below).

*3.1.1.2 Receivers.* MesoNetHS receivers are responsible for sending positive and negative acknowledgment packets as directed by entries in an incoming queue. In MesoNetHS, each data packet associated with a flow is assigned a monotonically increasing (integer) sequence number. Acknowledgement (ACK) and negative acknowledgment (NAK) packets also contain a sequence number that identifies the sequence number of the next data packet expected by a receiver. In MesoNetHS, packets flow along a fixed route; thus, packets may be lost but not reordered. For this reason, the sequence numbers of packets on a flow can continually increase and still indicate packets losses and receptions. When a data packet arrives with an expected sequence number then an ACK packet should be sent with the next sequence number. When a data packet arrives with an unexpected sequence number then a NAK packet should be sent with the next sequence number. When a source receives either an ACK or a NAK, this means that a receiver got a data packet, though perhaps not the expected data packet. Thus, by counting the number of ACK and NAK packets from a receiver, a source can determine how many data packets were received. Data packets lost before reaching a receiver or ACK and NAK packets lost before reaching a source will cause a source to send additional data packets, which are retransmissions.

The processing of a receiver is quite simple. If there are no entries in its queue, then the receiver does nothing. If there are entries in its queue, then the receiver removes the first entry and generates an outgoing SYN+ACK, ACK or NAK packet as indicated. The outgoing packet is given to the access router associated with the receiver. If the incoming queue of the access router is full, then the packet is discarded. Sending multiple ACK or NAK packets requires multiple invocations of a receiver. Thus, the number of

packets that a receiver may send per time step is capped by the receiver's capacity (in ppts), which is established by a configuration parameter (see 3.2.4 below).

*3.1.1.3 Access Routers.* Access routers are the most complex elements of a MesoNetHS model. Each access router has two queues: one for packets bound up toward the network core and one for packets bound down toward sources and receivers. At each invocation, an access router processes at most one packet, alternating between the up and down queues. If no packet is available, then the access router simply does nothing. Access routers are also responsible for three roles: (a) forwarding outgoing packets from sources and receivers toward the network core, (b) processing incoming SYN and data packets on behalf of subordinate receivers, and (c) processing incoming SYN+ACK, ACK and NAK packets on behalf of subordinate sources. Each of these roles is addressed in turn.

If a packet is outbound toward the network core, then the access router removes the packet from the up queue and finds the next-hop router (either a POP router or a backbone router) and forwards the packet to that router. If the incoming queue of the destination router is full, then the packet is discarded.

If a packet is inbound toward the network edge, then the router removes the packet from the down queue and applies processing that depends upon the packet type. If the inbound packet is a SYN, then the access router creates an entry for the destination receiver requesting generation of a SYN+ACK and places the entry into the receiver's work queue. If the inbound packet is a SYN+ACK, then the access router changes the phase of the destination source to be CONNECTED. If the inbound packet is a data packet, then the access router compares the packet sequence number to the sequence number expected by the receiver. If the sequence numbers match, then the access router creates an ACK entry and places it into the work queue of the destination receiver. If the sequence numbers do not match, then the access router creates a NAK entry and places it into the work queue of the destination receiver. The access router also updates the sequence number of the destination receiver to be one greater than the sequence number in the received data packet.

If the inbound packet is an ACK or NAK packet, then the access router updates the destination source state to reflect the feedback. First, the access router decrements the number of data packets that remain to be sent on the flow. If no more data packets remain to be sent, then the source is placed into the OFF state and a time is selected for the source to return to the ON state. If the inbound ACK or NAK is an interim packet in the flow, then the access router updates the SRTT and retransmission timeout for the source. The access router also sets the highest sequence number acknowledged for this flow to be the sequence number in the ACK or NAK. If the sequence number in the ACK or NAK is lower than a sequence number recorded when the last NAK was received for the source, then the remainder of the feedback processing is skipped because the packet provides outdated information. The feedback processing that remains is to update the state of the source's congestion-control variables (notably the congestion window and slow-start threshold) based on (a) packet type (ACK or NAK), (b) current state of the congestion-control variables, and (c) rules of the specific congestion-control algorithm being simulated for the flow associated with the packet.

MesoNetHS increases the congestion window identically for all flows for which the congestion window is below the slow-start threshold. The procedures adopted

correspond to limited slow-start, as suggested by Sally Floyd [8]. Upon receiving an ACK, if the congestion window is below some `MAX_SS_THRESHOLD`, then the congestion window is increased by one, which amounts to an exponential increase. If the congestion window exceeds `MAX_SS_THRESHOLD`, then the congestion window (*cwnd*) is increased by  $1/(cwnd/(0.5 \times \text{MAX\_SS\_THRESHOLD}))$ , which amounts to a logarithmic increase. When the congestion window exceeds the slow-start threshold, the increase resulting from an ACK depends upon the particular congestion-control algorithm being simulated for the flow. For TCP Reno, receiving an ACK during congestion avoidance increases the congestion window by  $1/cwnd$ , which results in a linear increase.

Upon receiving a NAK, the access router records the next data sequence number for the source when the NAK was received. This marks the ACK or NAK sequence number that must be received before feedback processing can recommence on this flow. In addition, receiving the NAK causes the access router to reduce the source's congestion window. For TCP Reno, the congestion window is reduced by  $\frac{1}{2}$  and the slow-start threshold is set to the reduced congestion window, which ensures that the flow enters the congestion-avoidance phase and increases *cwnd* linearly upon receiving new ACKs. The explanation of other congestion-control algorithms is postponed until Sec. 5.

Whenever an access router is invoked it processes only one packet. Thus, to process multiple packets an access router must be invoked multiple times. The rate at which an access router is invoked denotes its capacity (in ppts). This capacity is established by configuration parameters (see 3.2.3 below).

*3.1.1.4 Point-Of-Presence Routers.* Point-of-Presence (POP) routers provide intermediate points to connect access routers to the network backbone. (Note that some access routers may be connected directly to backbone routers, as discussed below in 3.1.2.2). POP routers serve only to forward packets from access routers toward backbone routers and to forward packets from backbone routers toward access routers. Thus, POP routers act as statistical multiplexor and demultiplexor elements. MesoNetHS topologies do not permit POP routers to be connected to each other or to be connected to multiple backbone routers.

Each POP router has two queues: one for packets bound up toward the network core and one for packets bound down toward the network edge. At each invocation, a POP router processes at most one packet, alternating between the up and down queues. If no packet is available from the appropriate queue, then the POP router is idle; otherwise, the POP router removes the first packet from the queue. If the packet is inbound, then the POP router looks up the next-hop access router and forwards the packet. If the incoming queue of the access router is full, then the packet is discarded. If the packet is outbound, then the POP router looks up the next-hop backbone router and forwards the packet. If the queue of the backbone router is full, then the packet is discarded. To process multiple packets, a POP router must be invoked multiple times. The rate at which a POP router is invoked denotes its capacity (in ppts). This capacity is established by configuration parameters (see 3.2.3 below).

*3.1.1.5 Backbone Routers.* Backbone routers connect to each other through backbone links and also connect to POP or access routers. Thus, backbone routers forward outbound and transit packets to backbone links and forward inbound packets to POP or

access routers. Each backbone router has a single queue from which one packet is processed on each invocation. If the queue is empty, then the backbone router does nothing. If the queue is not empty, then the backbone router removes and processes the first packet. If the packet is inbound, then the backbone router looks up the next-hop POP or access router and forwards the packet onward. If the incoming queue of the next-hop router is full, then the packet is discarded. If the packet is an outbound or transit packet, then the backbone router forwards it on to the appropriate backbone link. To process multiple packets, a backbone router must be invoked multiple times. The rate at which a backbone router is invoked denotes its capacity (in ppts). This capacity is established by configuration parameters (see 3.2.3 below).

*3.1.1.6 Backbone Links.* Backbone links are simplex links connecting (source and destination) pairs of backbone routers and delaying packets for a specified propagation time prior to delivering them to the next backbone router. Generally, backbone links are paired so that if one goes from source backbone router A to destination backbone router B, then a second goes from source backbone router B to destination backbone router A. Also, generally each of the paired backbone links exhibits the same propagation delay, though the model permits configuring asymmetric propagation times. Backbone links are simulated as queues sized to hold as many packets as a backbone router can forward in one time step times the number of time steps required to propagate packets on the link. At each invocation, a backbone link processes at most one packet. If the queue is empty, then the backbone link does nothing. If the queue is not empty, then the backbone link examines the first packet. If it is not yet time for the packet to arrive, then the backbone waits. Once the time has come for the packet to arrive at the destination backbone router, then the backbone link removes the packet from its queue and forwards it on to the backbone router. If the backbone router queue is full, then the packet is discarded.

### **3.1.2 Network Topology**

Network topology (i.e., the arrangement of routers and links and routes that transit them) plays a key role in shaping macroscopic behavior in a packet network. The study of network topologies, particularly of the larger Internet topology (a network of networks), remains subject to research [13-16, 19-23, 30, 33]. Given access to selected measurement points and little other information, researchers attempt to generate maps of network topologies [18, 24-25, 28-29, 32]. This becomes quite a difficult problem for the Internet as a whole, where a topology of autonomous systems is the measurement aim. Even generating maps of individual autonomous systems, each perhaps representing the topology of an Internet service provider (ISP), can be quite challenging when making measurements from outside the topology [17, 31].

MesoNetHS models a single ISP (Internet Service Provider) and thus the topology of interest is a collection of routers and links that compose a single autonomous system. MesoNetHS allows a user to define any network topology as long as a few restrictions are followed. First, POP routers may only connect to one parent (backbone) router, but may connect to multiple children (access) routers. Each access router may only connect to one parent router, either a POP router or a backbone router. This means that POP and access routers may not connect to routers of the same type. This results in fixed routing for packets flowing toward and away from the network backbone. Second, the network

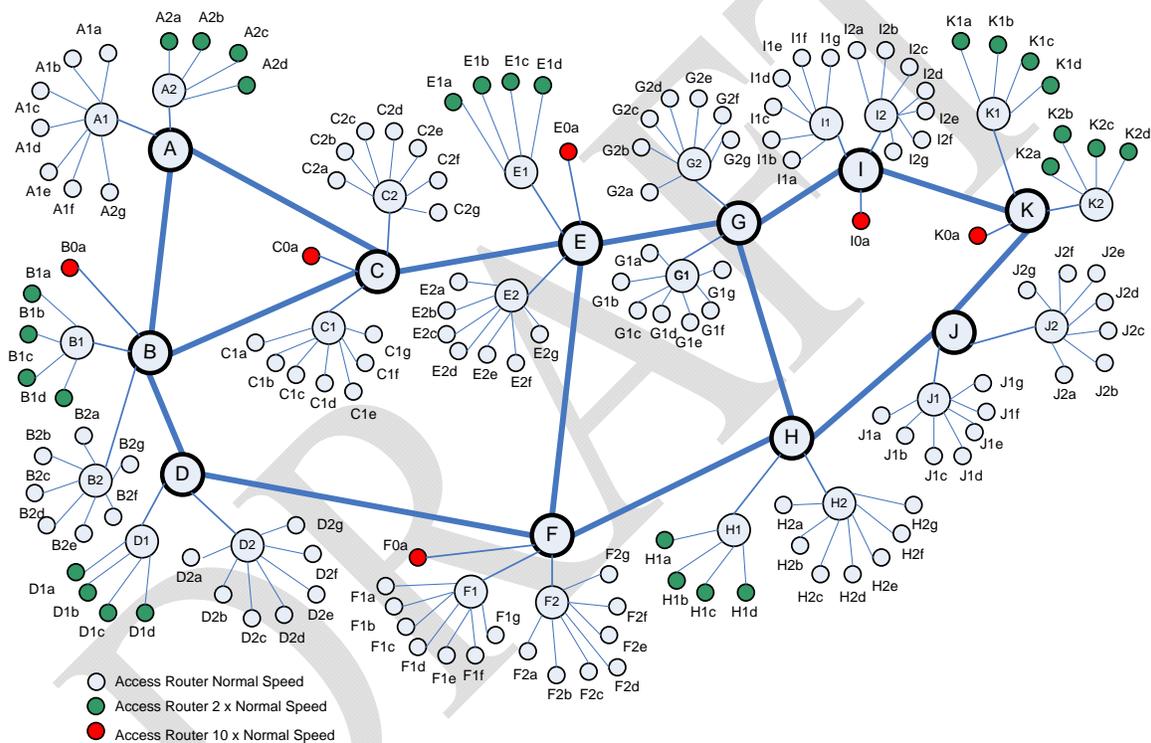
backbone must define fixed routes for packets flowing in a given direction between pairs of backbone routers. This means that packets flow on fixed routes among backbone routers. These restrictions deviate from some real network topologies in a couple of ways. First, some real network topologies provide multiple links between pairs of routers. These multiple links provide increased capacity when all links are operating and also provide increased reliability when selected links are down. For MesoNetHS, higher capacity links would be modeled as higher capacity routers. MesoNetHS does not support capacity reductions that would occur when selected links within a multiple-link set fail. Second, some real network topologies allow links between routers at the same level. These same-level links may provide multiple paths to adapt around temporary outages. MesoNetHS does not represent such alternate routes. Also, by assigning link costs, multiple paths may be used to engineer traffic flows along various routes. In such cases, link costs are not associated with link propagation delay but are chosen to create desired traffic patterns. MesoNetHS can be used to represent such traffic engineering because MesoNetHS does not explicitly require that routes be assigned based on link propagation delays.

MesoNetHS topology restrictions have several implications. First, packets in each direction between a source and destination flow on the same fixed path. Second, the link capacities in MesoNetHS topologies remain fixed for the duration of a simulation.

For the experiments discussed in this report, MesoNetHS uses a single topology, shown in Fig. 3-1. The backbone for this topology was adapted from the original Abilene backbone [26]. Other aspects of the topology were derived from investigations of ISP topologies, as reported in the published literature [27]. The simulated topology defines a fixed route for packets flowing between each source-receiver pair. While MesoNetHS may specify routes based on any criteria, the routes specified in the experiments reported in this document are shortest path routes that are based typically on link propagation delays. The assigned paths in both directions between a given pair of routers are usually, but not always, mirrors of each other. This implies that routes are not always determined by link propagation delays. The use of asymmetric routes is similar to the definition of traffic-engineered routes defined for many real ISP topologies. The routes used in this study are specified below in Table 3-2.

The resulting simulated topology was compared against an example topology used by an ISP. The comparison provided confidence in the main features of the topology shown in Fig. 3-1. Differences arose from topological restrictions imposed by MesoNetHS, as well as from the need to execute simulations within a tractable amount of computing time. There were four main differences observed between the topology in Fig. 3-1 and the example ISP topology. First, the real ISP topology used fixed path routing with weights assigned to achieve traffic engineering objectives rather than assigning weights based on propagation delays. Thus, while routes taken through both topologies are fixed, none of the routes in the real topology were shortest-path routes based on propagation delay. Second, the real ISP topology used multiple links to connect many pairs of routers. This provided an increase in both capacity and reliability. The simulated topology included only single links between router pairs. Thus, the simulated topology could not support as much traffic as the real topology. This restriction was adopted to decrease simulation time. Further, the simulated topology could not represent capacity decreases resulting from link failures. This restriction was adopted because the periods of

time simulated (between 20 and 60 minutes) were of short duration and because investigations were not focused on network changes arising from link failures. Third, the real ISP topology exhibited multiple links from POP routers to backbone routers and also between POP routers. These multiple links were used for traffic engineering and for improving reliability. Fourth, the real topology included around 60 backbone routers and 81 bi-directional links, 85 POP routers and 663 access routers. The simulated topology includes only 11 backbone routers and 14 bi-directional links, 22 POP routers and 139 access routers. This restricts the simulated topology to carry less traffic than the real topology. Of course, restricting the size of the simulated topology reduces the computation time required to simulate the entire network over a given period of time. This is especially important when attempting to simulate high-speed forwarding capacities, ranging up to 384 Gbps in simulations described in this report.



**Figure 3-1. Topology used for simulation experiments discussed in this study**

In summary, the topology simulated in this study represents one instance of a real backbone topology (the original Abilene network). The remainder of the topology was inspired by traits from real ISPs, as reported from investigations in the published literature. The main restrictions adopted in the simulated topology deal with reducing the number and capacity of routers and links. These restrictions were adopted to allow for feasible simulations of network evolution within available computing resources. The sections that follow describe the key aspects of the topology used in experiments throughout this study.

*3.1.2.1 Four-Tier Structure.* The topology consists of a four-tier structure. The top tier is the network backbone, composed of 11 backbone routers (A-K in Fig. 3-1) and 14 bi-directional links. Routers forward packets at some specified rate. Propagation delays are imposed only when packets transit backbone links. The second tier comprises 22 POP routers (A1-K2 in Fig. 3-1). The third tier comprises 139 access routers (A0a-K2d in Fig. 3-1). The fourth tier (not shown in Fig. 3-1) consists of sources and receivers that represent computers exchanging data over the Internet.

Data packets are generated at a source, forwarded to an associated access router and then on to the access router's parent, either a POP router or a backbone router. Outgoing data packets that reach a backbone router are forwarded on a link connecting to the next-hop backbone router on the route and so on until reaching the backbone router under which the receiver may be found. The destination backbone router forwards the packet on to the destination child router (either POP or access router). An inbound data packet reaching a POP router is forwarded on to the destination access router and then to the destination receiver. Acknowledgment packets flow in a similar fashion but in the reverse direction from receiver to source. This four-tier structure is similar to the structure that exists in many ISP networks.

In general, backbone routers operate at higher speed than POP routers and POP routers operate at higher speed than access routers. Further, topologies are constructed so that backbone and POP routers can handle the expected load, leaving bottlenecks to arise at the level of access routers. For the topology in Fig. 3-1 to mimic this behavior requires that router speeds be configured properly.

*3.1.2.2 Heterogeneous Composition.* Recent research, which attempts to map the network topology for selected ISPs, suggests a heterogeneous composition for real networks [27]. For example, some access routers operate at higher speeds than others and some access routers connect directly to backbone routers rather than transit across POP routers. The topology in Fig. 3-1 includes both forms of heterogeneity. First, access routers operate at three different speeds: normal, twice normal speed (green routers in Fig. 3-1) and ten times normal speed (red routers in Fig. 3-1). Second, selected access routers connect directly to backbone routers. The main reason access routers connect directly to backbone routers is because they require higher bandwidth. For this reason, the topology in Fig. 3-1 has access routers, operating at ten times the normal speed, connect directly to backbone routers. Further, POP routers can handle fewer high-speed access routers than access routers of normal speed. The topology in Fig. 3-1 reflects this by allowing only four fast access routers to connect to a POP router, while a POP router can handle seven access routers of normal speed. Further heterogeneity is possible, for example, by allowing three normal access routers and two fast access routers to connect to POP routers.

*3.1.2.3 Flow Classes.* Access routers provide network access for sources and receivers. The division of access routers into three speed classes (normal, fast and directly connected) has the effect of creating six possible classes of flows, depending upon the location of a flow's source and receiver: **NN**<sup>1</sup> (normal-normal), **FN** (fast-normal), **DN** (directly connected-normal), **FF** (fast-fast), **DF** (directly connected-fast) and **DD** (directly

---

<sup>1</sup> We color code flow-class designations to indicate their relationship to the types of access routers in the topology.

connected-directly connected). Thus, heterogeneity among access routers leads to heterogeneity among flows, which allows measurements to be made regarding the performance of flows within each of the six classes.

*3.1.2.4 Fixed-Path Routing.* Even when a choice exists among equal-cost routes, most deployed routers in the Internet attempt to ensure that all packets related to an individual flow transit the same fixed route in a given direction [42-44]. The aim is to have packets reaching receivers in sequence and thus to reduce problems associated with fragmentation of reassembly buffers. There exists debate on how successful routers are in achieving this aim [39-40, 46]. Since packet sequencing cannot be completely guaranteed on the Internet, receivers will sometimes indicate (via a duplicate acknowledgment) a packet has been missed, when in fact the packet may have been reordered rather than lost. For this reason, TCP sources typically wait for three duplicate acknowledgments before initiating retransmission of a lost packet. This allows time for reordered packets to arrive at a receiver and be acknowledged. The cost of this heuristic is that reaction to actual lost packets is somewhat delayed.

MesoNetHS topologies are totally successful in achieving packet sequencing along the route in each direction between a source and receiver. The upshot of this MesoNetHS feature is that packets received out of order can be assumed to signal a loss. For this reason, MesoNetHS sources take retransmission actions after receiving one, rather than three, duplicate acknowledgments. (Note that MesoNetHS represents duplicate acknowledgments as negative acknowledgments, or NAKs).

**Table 3-1. Link Propagation Delays in the Base Simulated Topology**

Router Pair	Link Pair	Link Propagation Delay (in time steps)
A-B	1-2	10
A-C	3-4	12
B-C	5-6	12
B-D	7-8	4
C-E	9-10	8
D-F	11-12	20
E-F	13-14	10
E-G	15-16	7
F-H	17-18	9
G-H	19-20	7
G-I	21-22	4
H-J	23-24	7
I-K	25-26	5
J-K	27-28	3

*3.1.2.5 Simulated Abilene Backbone Characteristics.* All experiments described in this study operate within the context of a simulated Abilene backbone network topology (recall Fig. 3-1) with the characteristics described in this section. First, the backbone links in the topology exhibit the base propagation delays shown in Table 3-1. The simulated topology encompasses (14 x 2 =) 28 unidirectional backbone links. The propagation delays, shown in Table 3-1, are the same for both backbone links (one in each direction) that connect the indicated router pairs. The odd numbered link of each pair flows in the

forward direction (e.g., A to B) and the even numbered link flows in the reverse direction (e.g., B to A).

The simulated topology defines fixed routes, given in Table 3-2, for packets flowing between sources and receivers located under specific pairs of backbone routers. The fixed routes indicate links that are followed and so the sum of the link propagation delays determines the route propagation delays (as indicated in Table 3-2). The round-trip propagation times between selected source and destination domains can be determined by summing, from Table 3-2, the route propagation delay in the forward direction and the route propagation delay in the reverse direction. The average round-trip propagation delay for the base topology is 41 time steps; the minimum roundtrip propagation delay is 6 time steps (i.e., route J-K-J) and the maximum roundtrip propagation delay is 100 time steps (i.e., route A-B-D-F-H-J-H-F-D-B-A). One-way paths that are part of the five asymmetric routes are highlighted in red in Table 3-2.

**Table 3-2. Routes across the Backbone from Source (S) to Destination (D) Domain (One-Way Route Propagation Delays in Time Steps Given in Parentheses)**

S\D	A	B	C	D	E	F	G	H	I	J	K
A	-	A-B (10)	A-C (12)	A-B-D (14)	A-C-E (20)	A-B- D-F (30)	A-C- E-G (19)	<b>A-C- E-G-H (34)</b>	A-C- E-G-I (31)	A-B- D-F- H-J (50)	A-C- E-G-I- K (36)
B	B-A (10)	-	B-C (12)	B-D (4)	B-C-E (20)	B-D-F (24)	B-C- E-G (27)	B-D- F-H (33)	B-C- E-G-I (31)	B-D- F-H-J (40)	<b>B-C- E-G-I- K (36)</b>
C	C-A (12)	C-B (12)	-	C-B-D (16)	C-E (8)	C-E-F (18)	C-E-G (15)	C-E- G-H (22)	C-E- G-I (19)	C-E- G-H-J (29)	C-E- G-I-K (24)
D	D-B-A (14)	D-B (4)	D-B-C (16)	-	<b>D-B- C-E (24)</b>	D-F (20)	<b>D-F- E-G (37)</b>	D-F-H (29)	D-F- H-G-I (40)	D-F- H-J (36)	D-F- H-J-K (39)
E	E-C-A (20)	E-C-B (20)	E-C (8)	<b>E-F-D (30)</b>	-	E-F (10)	E-G (7)	E-G-H (14)	E-G-I (11)	E-G- H-J (21)	E-G-I- K (16)
F	F-D- B-A (30)	F-D-B (24)	F-E-C (18)	F-D (20)	F-E (10)	-	<b>F-E-G (17)</b>	F-H (9)	F-H- G-I (20)	F-H-J (16)	F-H-J- K (19)
G	G-E- C-A (19)	G-E- C-B (27)	G-E-C (15)	<b>G-H- F-D (20)</b>	G-E (7)	<b>G-H-F (16)</b>	-	G-H (7)	G-I (4)	G-H-J (14)	G-I-K (9)
H	<b>H-F- D-B-A (43)</b>	H-F- D-B (33)	H-G- E-C (22)	H-F-D (29)	H-G-E (14)	H-F (9)	H-G (7)	-	H-G-I (11)	H-J (7)	H-J-K (10)
I	I-G-E- C-A (31)	I-G-E- C-B (31)	I-G-E- C (19)	I-G-H- F-D (40)	I-G-E (11)	I-G-H- F (20)	I-G (4)	I-G-H (11)	-	I-K-J (8)	I-K (5)
J	J-H-F- D-B-A (50)	J-H-F- D-B (40)	J-H- G-E-C (29)	J-H-F- D (36)	J-H- G-E (21)	J-F-H (16)	J-H-G (14)	J-H (7)	K-I (5)	-	J-K (3)
K	K-I-G- E-C-A (36)	<b>K-J- H-F- D-B (41)</b>	K-I-G- E-C (24)	K-J-H- F-D (39)	K-I-G- E (16)	K-J-H- F (19)	K-I-G (9)	K-J-H (10)	J-K-I (8)	K-J (3)	-

### 3.1.3 Simulated Packets

Simulated packets in MesoNetHS share many elements with real Internet packets, while also lacking some traits of real packets. Each simulated packet comprises six main fields:

(a) packet type, (b) source address, (c) destination address, (d) flow identifier, (e) sequence number and (f) creation timestamp. Each simulated packet also contains a propagation time field used to control propagation across simulated backbone links. MesoNetHS packets can have one of the following types: SYN, SYN+ACK, DT, ACK or NAK. Each source and destination address in MesoNetHS consists of three components: (1) domain identifier, (2) POP identifier, and (3) access-point identifier. Flow identifiers are represented as a pair of pointers: one to a source and one to a receiver. MesoNetHS packets are assigned monotonically increasing sequence numbers; however, numbers are skipped whenever a packet in the flow is lost.

The main difference between MesoNetHS packets and real Internet packets is that MesoNetHS packets lack size. This simplification was adopted to allow MesoNetHS to represent packet-level behavior without having to simulate the lower level of octets or bits. This was intended to make MesoNetHS easier to code and understand and faster to simulate. One could then assume that all MesoNetHS packets have the same size – for example, one could assume that each packet is 1500 octets (12000 bits) in length. In this way, processing rates expressed as packets per time step can be converted into other bases, such as octets per time step or bits per time step.

Real Internet packet flows may consist of data going in both directions (i.e., flows may be full duplex), which also permits ACKs to be piggybacked on data packets. MesoNetHS flows simulate data moving in one direction only (i.e., flows are simplex) and thus ACKs are not piggybacked on data packets but instead are sent individually. The restriction to unidirectional data flows (with ACKs flowing in the reverse direction) combined with lack of packet size has the unrealistic effect that MesoNetHS processes simulated ACKs at the same rate as simulated data packets, which would not occur in the real Internet because ACK packets typically have a much smaller size than data packets. This is one reason why MesoNetHS cannot be used to derive precise quantitative measures of network performance.

Real Internet packets do not include a NAK type – but instead use duplicate acknowledgments. Real Internet packets also include a FIN type, used to close flows. MesoNetHS dispenses with the FIN exchange and instead closes flows implicitly when a source receives the last expected ACK or NAK packet from a receiver. Real Internet packets may also include a RESET type, which can abort a flow. MesoNetHS allows a source to abort a flow with resorting to a RESET packet.

Real TCP implementations may include several optional, optimization features, such as deferring ACKs, pacing data transmissions and selectively acknowledging packets. Further, real TCP implementations implement various procedures, such as keep-alive ACKs that ensure a flow remains active even in the absence of data packets. MesoNetHS does not simulate any of these features or procedures, unless required by particular simulated transport protocols.

### 3.1.4 Relating Abstract Time to Real Time

MesoNetHS simulation progresses with respect to abstract time steps that have no specific meaning within the domain of real time. Various parameters in MesoNetHS specify element capacities in terms of units per abstract time step. This abstract formulation of time is quite unusual for network designers and engineers, who tend to think in capacities dimensioned along real time (e.g., bytes per second, packets per

second or bits per second) instead of abstract time. To achieve mapping between abstract time steps and real time, MesoNetHS includes an explicit parameter to specify the basic simulated time unit. By setting this parameter, an experimenter creates a link between abstract time steps and real time. Configured capacities, given in packets per time step, are converted into delays (i.e.,  $1/\text{capacity}$ ) per packet, and then scaled by the basic time unit. For example, suppose a router is defined as operating at 400 packets per time step and the basic simulated time unit is defined as one millisecond (0.001), then the time taken to process one packet is computed as  $1/400 \times 0.001 = 0.0000025$  seconds and thus the router processes packets at  $1/0.0000025 = 400,000$  packets per second.

## 3.2 Model Configuration

The operation of MesoNetHS elements is controlled by a number of parameters that must be specified by the experiment designer. This section describes these configuration parameters, divided into six categories: (1) simulation control, (2) definition of user behavior, (3) adaptation of the network topology, (4) description of sources and receivers, (5) specification of optional long-lived flows, and (6) transport protocols. The settings of MesoNetHS parameters associated with a specific simulation run are reported in an output file, as explained below in Sec. 3.2.8.

### 3.2.1 Simulation Control Parameters

Parameter `M` defines the length of the measurement interval in abstract time steps. The model will make sample measurements (see Sec. 3.3 below) at the end of each measurement interval. Parameter `MI` defines the number of measurement intervals over which the simulation will execute. Thus, the total time simulated will be  $M * MI$  time steps. Parameter `basicTimeUnit` establishes a link between abstract time steps and simulated real time. This link is explained above in Sec. 3.1.4.

Parameter `MB` defines the number of measurement intervals that can be buffered by the model. When the measurement buffer fills, measurements are appended to appropriate files and the measurement buffer is cleared to accumulate subsequent measurements. This enables very long simulations to be conducted without using excessive memory.

Parameter `exID` can be used to assign a number to a simulation run. This number provides one means of disambiguating measurement file names and of associating a related set of measurement files. The system time is used as another means. Typically, the parameter `exID` would be set to specific run numbers associated with some experiment design. For example, a sensitivity-analysis experiment requiring 64 runs would set `exID` sequentially from 1 to 64 to reflect each of the runs required by the experiment design. This would allow specific output files to be associated with specific experiment configurations.

Parameter `RandOffset` defines the offset used to parameterize seeds for the random-number streams<sup>2</sup> used in MesoNetHS. Typically, for experiment designs that expose different system configurations to similar random conditions, the `RandOffset` should be set to the same value for all runs. Alternatively, when the same system

---

<sup>2</sup> MesoNetHS uses seven random-number streams to control various aspects of the simulation, such as assigning congestion-control algorithms to sources, generating think times, assigning network-interface speeds, starting sources, determining file sizes and file types, and selecting flow receivers.

configuration is to be exposed to varying random conditions, `RandOffset` should be set to different values.

### 3.2.2 Parameters Defining User Behavior

The parameters defining user behavior encompass two main aspects: think time and file size. Parameter `lambdaOFF` specifies the average number of time steps that a user delays between file transfers. This parameter is the average of an exponential distribution. Parameter `lambdaON` specifies the average number of packets that a flow transfers. This parameter is the average of a Pareto distribution. Parameter `alpha` is the shape of the Pareto distribution of file sizes. The Pareto distribution results in heavy-tailed file sizes, which some researchers have observed on the Internet [34-37].

The fundamental file size parameter (`lambdaON`) should be construed as representing files sizes of typical web pages on the Internet. Sometimes, users may decide to download files, linked from web pages. For example, a user may download a report, a dataset, a song, a photograph or a video. The parameter `Fp` represents the probability with which a user decides to download a linked file. In general, linked files would be larger than typical web pages; thus, the parameter `Fx` represents the multiplier by which linked files will be increased beyond normal web page sizes. Thus, after thinking, a user will select a file size with an average of `lambdaON`. Then, with probability `Fp`, the file size will be multiplied by `Fx` to represent a linked file<sup>3</sup>.

MesoNetHS can also simulate specific periods during which `DD` flows (between sources and receivers under pairs of directly connected access routers) exchange very large scientific datasets. Parameter `Jon` defines the proportion of total simulated time that must elapse before this mode of operation commences and parameter `Joff` defines the proportion of total simulated time that must elapse before this mode of operation ceases. Thus, this special mode commences when simulated time steps reach  $M \times MI \times Jon$  and ceases when simulated time steps reach  $M \times MI \times Joff$ . When this mode is operational, the selected file size of any `DD` flow will be multiplied by parameter `Jx`. Setting `Jon`  $\geq 1.0$  disables this mode of operation.

### 3.2.3 Parameters Adapting Network Topology

While requiring specification of a basic network topology (such as defined in Sec. 3.1.2), MesoNetHS permits an experiment designer to adapt that topology in several ways. Parameter settings allow adjustment of link propagation delays, router speeds and buffer sizes in routers.

Parameter `deltaX` is set to a factor that is used to multiply the link propagation delays that were specified in the base topology. Setting `deltaX` = 1.0 means that base propagation delays will remain unchanged. Setting `deltaX` > 1.0 increases propagation delays by the specified factor. Setting `deltaX` < 1.0 reduces propagation delays by the specified factor.

Parameter `R1` establishes the basic forwarding speed (in packets per time step) of the backbone routers; however, since backbone routers handle transit traffic as well as inbound and outbound traffic, the actual capacity of backbone routers is set to

---

<sup>3</sup> In Sec. 8 we introduce additional file-size probabilities and multipliers that allow simulation of larger files, such as service-pack and movie downloads.

$R1 \times BBspeedup$ , where  $BBspeedup$  could be set to reflect the number of transit links for a typical backbone router in a given topology. By default,  $BBspeedup = 1$ . MesoNetHS enforces the usual relationship that backbone routers are faster than POP routers, which are faster than typical access routers. To ensure this relationship, parameters  $R2$  and  $R3$  are set to divisors that are used to reduce  $R1$ . The speed of POP routers is set to  $R1/R2$ , while the speed of typical access routers is set to  $R1/R2/R3$ . Recall, though, that MesoNetHS topologies allow for some heterogeneity with respect to access routers: some access routers may be faster than typical access routers and some access routers may be connected directly to backbone routers. Parameters  $Bfast$  and  $Bdirect$  are used to increase the speed of such routers. Fast access routers are assigned a capacity of  $R1/R2/R3 \times Bfast$ . The capacity of each directly connected access router is set to  $R1/R2/R3 \times Bdirect$ .

Rather than assigning buffer sizes directly for routers, MesoNetHS implements two alternative buffer-sizing algorithms. One algorithm, which embodies recommended practice [41, 45], sizes each router's buffer to accommodate the estimated average round-trip time (RTT) for routes transiting the router multiplied by the router's capacity ( $C$ ). Thus, this algorithm sets the buffer size of each router to  $RTT \times C$ , where  $RTT$  is the average round-trip time for the topology and  $C$  is the capacity of the specified router – thus faster routers will have larger buffers and larger propagation delays will increase buffer sizes in all routers. The second algorithm, following the suggestion of McKeown and colleagues [38], divides the computed buffer size by the square root of the expected number of flows transiting a router. This algorithm then sets buffer sizes to  $RTT \times C / \text{SQRT}(N)$ , where  $N$  is the expected number of flows transiting a given router. MesoNetHS estimates  $N$  by aggregating the number of sources and receivers under a router (or under subordinate routers) and then dividing by two.

Parameter  $QszAlg$  specifies which buffer sizing algorithm to use: 1 means  $RTT \times C$ ; 2 means  $RTT \times C / \text{SQRT}(N)$ ; 3 means  $(RTT \times C + RTT \times C / \text{SQRT}(N)) / 2$ . Setting  $QszAlg = 3$  amounts to interpolating between the two algorithms. The buffer size, as computed by the specified algorithm, is then multiplied by parameter  $Qfactor$  to establish the final buffer size of each router. Setting  $Qfactor = 1.0$  uses the buffer size computed by the selected algorithm, while setting  $Qfactor > 1.0$  increases buffer size and setting  $Qfactor < 1.0$  decreases buffer size.

### 3.2.4 Parameters Describing Sources and Receivers

MesoNetHS uses several parameters to control the number, distribution and speed of sources and receivers in the fourth tier of the network topology. Every access router in the simulated topology contains some number of sources and receivers. Parameter  $baseSources$  defines the relative scale of the number of sources (and receivers) under a typical access router. For example, if  $baseSources = 100$ , then access routers will have on the order of 100 sources each, subject to variations arising through specification of related parameters, as discussed below. If one knows the number of access routers ( $Na$ ) in a given topology, then one can compute  $Na \times baseSources$ , which gives the approximate maximum number of active flows. For example, the topology in Fig. 3-1 has 139 access routers; thus, for  $baseSources = 100$  the maximum number of active flows would be about 13900. In order to reduce the blocking probability for a source seeking a receiver in any given domain, MesoNetHS ensures that the base number of receivers under each

access routers is  $4 \times \text{baseSources}$ . In the example used here, then, the number of receivers would be on the order of  $4 \times 100 \times 139 = 55600$ .

Parameter  $\text{deltaU}$  permits linear scaling of  $\text{baseSources}$  in order to increase or reduce the approximate number of sources and receivers in a given topology. The approximate number of sources under each access router is then computed as  $\text{deltaU} \times \text{baseSources}$ . Continuing the example, assume that  $\text{deltaU} = 2.0$ ; then the approximate maximum number of active flows would become  $2 \times 100 \times 139 = 27800$  and the number of receivers would be on the order of  $2 \times 4 \times 100 \times 139 = 111200$ .

The sources and receivers in a topology must be distributed in some fashion. One possibility is to assign an equal number of sources and receivers to each access router. This would suggest a peer-to-peer world [21], where flow patterns are equally likely between any pair of access routers. Another possibility is to concentrate sources in a selected set of access routers and to place most receivers in a different set of access routers. This would suggest a client-server world [24], where flow patterns reflect the location of customers and the popularity of content providers. MesoNetHS provides six parameters that can alter the distribution of sources and receivers among access routers in each of three classes: normal (N-class), fast (F-class) and directly connected (D-class).

Given a set of sources and one access router of each class, parameters  $\text{probNs}$ ,  $\text{probNsf}$  and  $\text{probNsd}$  specify, respectively, the probability that a source is located under the N-class router, the F-class router and the D-class router. Since each access router could contain  $\text{deltaU} \times \text{baseSources}$  sources, three access routers would contain  $3 \times \text{deltaU} \times \text{baseSources}$  sources. To distribute those sources under each class of access router, one multiplies the appropriate probability by the number of available sources. Continuing the example, given one access router in each of the three classes, one would find  $3 \times 2 \times 100 = 600$  sources. Suppose that  $\text{probNs} = 0.1$ ,  $\text{probNsf} = 0.6$  and  $\text{probNsd} = 0.3$ . Each N-class router would contain  $600 \times 0.1 = 60$  sources, each F-class router would contain  $600 \times 0.6 = 360$  sources and each D-class router would contain  $600 \times 0.3 = 180$  sources. Given the topology shown in Fig. 3-1, this distribution changes the maximum number of active flows to 17460, i.e.,  $105$  (N-class routers)  $\times 60$  (sources)  $+ 28$  (F-class routers)  $\times 360$  (sources)  $+ 6$  (D-class routers)  $\times 180$  (sources). This means that more flows will originate from fast and directly connected access routers than from normal access routers. Similar adjustments can be made with respect to the distribution of receivers.

Given a set of receivers and one access router of each class, parameters  $\text{probNr}$ ,  $\text{probNrf}$  and  $\text{probNrd}$  specify, respectively, the probability that a receiver is located under the N-class router, the F-class router and the D-class router. Since each access router could contain  $4 \times \text{deltaU} \times \text{baseSources}$  receivers, three access routers would contain  $3 \times 4 \times \text{deltaU} \times \text{baseSources}$  receivers. To distribute those receivers under each class of access router, one multiplies the appropriate probability by the number of available receivers. Continuing the example, given one access router in each of the three classes, one would find  $3 \times 4 \times 2 \times 100 = 2400$  receivers. Suppose that  $\text{probNr} = 0.8$ ,  $\text{probNrf} = 0.1$  and  $\text{probNrd} = 0.1$ . Each N-class router would contain  $2400 \times 0.8 = 1920$  receivers, each F-class router would contain  $2400 \times 0.1 = 240$  sources and each D-class router would contain  $2400 \times 0.1 = 240$  sources. Given the topology shown in Fig. 3-1, this distribution changes the maximum number of active receivers to 209760, i.e.,  $105$  (N-class routers)  $\times 1920$  (receivers)  $+ 28$  (F-class routers)  $\times 240$  (receivers)  $+ 6$  (D-class routers)  $\times 240$  (receivers).

This means that more flows will have receivers in normal access routers than in fast and directly connected access routers.

Taken together, the example used here might describe a client-server view of the distribution of sources and receivers. First, 58% of data sources reside under the 28 **F**-class routers while the remaining 42% of data sources reside under the other 111 access routers. Second, 96% of all data sinks are under the 105 **N**-class routers, while the remaining 4% of data sinks are under the 34 remaining access routers. This distribution of sources and receivers also influences the probability of various flow classes (recall the six flow classes described above in 3.1.2.3). For the example, 57% of all flows will be **FN** flows and 35% will be **NN** flows. The remaining 8% of flows are distributed as follows: 6.2% **DN** flows, 1.9% **FF** flows, 0.6% **DF** flows, and the remaining 0.04% **DD** flows.

Aside from differences in location, sources and receivers might also transmit packets at different speeds. For example, many users have computers that transmit at 100 Mbps; while some users have computers that transmit at 1 Gbps. MesoNetHS provides three parameters to specify speed differences among sources and receivers. Parameter **Hbase** defines the capacity (in packets per time step) of normal sources and receivers, while parameter **Hfast** defines the capacity of fast sources and receivers. Parameter **FastHostProb** specifies the probability that any given source or receiver is fast. Upon generation of a source or receiver, capacity is set to **Hfast** with probability **FastHostProb** and to **Hbase** with probability  $1 - \text{FastHostProb}$ .

Another key characteristic of sources is the mechanism used to react to congestion on flows (see Sec. 5 below). MesoNetHS simulates seven alternate congestion-control algorithms: standard TCP Reno [9-10], Binary-Increase Congestion control (BIC) [62], Compound TCP (CTCP) [59], High-Speed TCP (HS-TCP) [53], Scalable TCP [54], FAST [61] and Hamilton TCP (HTCP) [55]. Sources are assigned one of these seven algorithms with probabilities specified by the following parameters: **prTCP**, **prBICTCP**, **prCTCP**, **prHSTCP**, **prSCALABLE**, **prFAST** and **prHTCP**. Each of these parameters can be set to any real value between 0 and 1, provided that the sum of the seven parameter values is 1. Each source retains its assigned congestion-control algorithm for the duration of a simulation run.

Finally, MesoNetHS uses a probability distribution to determine the initial activation time for sources. Parameter **prON** specifies the probability that a source is activated immediately. Parameter **prONsecond** defines the probability that a source is activated as part of a second wave. Parameter **prONthird** defines the probability that a source is activated with the third wave. Remaining sources are activated in a fourth and final wave. Sources in the second wave are activated after a random delay, using an exponential distribution with a mean  $1/3 \times \text{lambdaOFF}$ . Sources in the third wave are activated after an average random delay of  $2/3 \times \text{lambdaOFF}$ . The remaining sources are activated after an average random delay of **lambdaOFF**.

### 3.2.5 Parameters Specifying Special Long-Lived Flows

MesoNetHS allows selected flows to be designated as long-lived flows, which start at a specified time and then transmit continuously, subject to congestion-control constraints. Selected measurements are taken so that the behavior of long-lived flows can be

monitored individually. Specification of long-lived flows requires the use of six configuration parameters.

Parameter `maxLongLivedFlows` (default value 3) determines the maximum number of long-lived flows that can be described, which dimensions the compile-time arrays used to hold associated measurements. One may change this parameter if the need arises to define more than three long-lived flows. Parameter `LL_FLOWS` defines the actual number of long-lived flows that are specified in a given configuration file. Note that the following constraint must hold:  $LL\_FLOWS \leq maxLongLivedFlows$ .

Four parameters describe the characteristics of a long-lived flow. Each of these parameters is an array with dimension `LL_FLOWS`, where a given array index specifies a particular long-lived flow and can be used to find the four, associated characteristics. Array `LL_FLOW_SOURCES` contains the access-router identifier under which the source of each long-lived flow can be found. Array `LL_FLOW_RECEIVERS` contains the access-router identifier under which the receiver of each long-lived flow can be found. Access-router identifiers are sequentially assigned integers that range between 1 and the number of access routers defined in a topology. For the topology defined in Fig. 3-1, access-router identifiers range over 1 (A1a) to 139 (K2d). For example, suppose `LL_FLOW_SOURCES` = {12, 24, 50} and `LL_FLOW_RECEIVERS` = {131, 102, 62}. These parameters specify three long-lived flows: (a) a flow where a source under access router 12 (B0a) transmits to a receiver under access router 131 (K0a), (b) a flow where a source under access router 24 (C0a) transmits to a receiver under access router 102 (I0a), and (c) a flow where a source under access router 50 (E0a) transmits to a receiver under access router 62 (F0a). Note that a specific source-receiver pair is created and pre-connected for each long-lived flow; thus, connection establishment procedures are not used.

Array `LLon` specifies the proportion of simulation time that must elapse before each designated long-lived flow will be activated. For example, given a configuration where `M` = 200 and `MI` = 7500 (i.e., a simulation requiring 1.5 million time steps), then a long-lived flow with `LLon` = 0.4 will start at measurement interval number 3000 (i.e., at time step 600000). If the specified `LLon` value is 0, then the flow begins immediately. If the specified `LLon` value is 1.0 or greater, then the flow will not be activated.

Array `SOURCE_TYPE` specifies the type of congestion-control mechanism to be used by the source of each long-lived flow. Acceptable values for the type of congestion-control mechanism are shown in Table 3-3. Note that any other value for the type of congestion-control mechanism causes MesoNetHS to assign a type probabilistically; where probabilities are as specified for other sources (see Sec. 3.2.4).

**Table 3-3. Specification of Values for Parameter `SOURCE_TYPE`**

<code>SOURCE_TYPE</code> Value	Congestion-Control Mechanism
1	TCP Reno
2	HSTCP
3	CTCP
4	Scalable TCP
5	FAST
6	HTCP
7	BIC

### 3.2.6 Parameters for Transport Protocols

The remaining parameters required to configure MesoNetHS simulations relate to transport protocols. Most of these parameters concern congestion-control algorithms, and also have recommended values. We defer discussion of these until Sec. 5. Three parameters relate to initial slow-start procedures, where MesoNetHS adopts the same behavior for all sources, regardless of congestion-control algorithm.

Any TCP flow starts without knowledge of the surrounding environment, including network-interface speed, access-link speed and congestion on the network path to a receiver. For this reason, TCP adopts a probing procedure, known (in a counterintuitive way) as slow start. During initial slow start, a TCP flow transmits relatively few packets. As acknowledgments are received successfully, a TCP flow then quickly increases its sending rate until a packet loss is signaled. After a packet loss, a TCP flow reduces its sending rate. Subsequently, receipt of acknowledgments causes a TCP flow to increase its sending rate less quickly than occurred prior to the loss.

The behavior of initial slow start is dimensioned primarily by two parameters. First, the initial congestion window (parameter `INITIAL_TCP_CWND`) determines the number of packets that will be transmitted prior to receiving an acknowledgment. As a default, MesoNetHS sets `INITIAL_TCP_CWND = 2`, which corresponds to the initial congestion window used in Microsoft Windows™ operating systems [12]. During initial slow start, TCP sources increase the congestion window by one packet for each acknowledgment received successfully. This amounts to an exponential increase in the congestion window (and corresponding sending rate). After the congestion window reaches a specified value, known as the slow-start threshold, further acknowledgments increase the congestion window (*cwnd*) by  $1/cwnd$ , which amounts to a linear increase. Thus, a second parameter, `INITIAL_TCP_SS_THRESHOLD`, defines the value of the congestion window after which the increase becomes linear. An appropriate setting for this value is not widely agreed. Some experts [7] suggest that the threshold should be set arbitrarily large in order to quickly discover available bandwidth. Others [5] reason that the threshold should be set based on feedback as to the maximum number of packets that can be buffered by the receiver. Still others [11] suggest a low threshold might be prudent for its positive influence in reducing network-wide congestion.

Sally Floyd observes [8] that using an arbitrarily large threshold in high-speed networks can lead to conditions where a long sequence of packets could be lost on a new flow, which results in wasted network capacity and poor user performance. Floyd also observes that using a low threshold in high-speed networks can lead to conditions where insufficient packets are sent early in a flow, which gives poor bandwidth utilization and poor user performance. Given the tradeoffs between an initial slow-start threshold that is too small and one that is too large, Floyd recommends using a limited slow start, which introduces a third parameter, `MAX_SS_THRESHOLD`. In limited slow start, a TCP source increases the congestion window exponentially with successive acknowledgments up to `MAX_SS_THRESHOLD`. Subsequently, receipt of successive acknowledgments causes the congestion window to be increased by  $1/cwnd/(0.5 \times \text{MAX\_SS\_THRESHOLD})$  until `INITIAL_TCP_SS_THRESHOLD`. In this second stage of initial slow start the congestion window increases logarithmically. After the congestion window reaches `INITIAL_TCP_SS_THRESHOLD`, the congestion window increases linearly with new acknowledgments. Floyd recommends a value of 100 for `MAX_SS_THRESHOLD`.

MesoNetHS adopts limited slow start procedures, as recommended by Floyd. By default, the relevant MesoNetHS parameters are set as follows: `MAX_SS_THRESHOLD = 100` and `INITIAL_TCP_SS_THRESHOLD =` an arbitrarily large integer. To deactivate limited slow start set `MAX_SS_THRESHOLD ≥ INITIAL_TCP_SS_THRESHOLD`.

Given the relatively small size of Web objects and high (and increasing) network speeds and relatively long propagation delays in networks, initial slow-start procedures can have a dominating effect on user throughputs. This follows because sources must wait for feedback before increasing congestion-window size (and corresponding sending rate). The initial size of the congestion window coupled with the rate of increase can dictate observed throughput for small to medium sized files. Thus, observed throughputs could be biased on individual flows that use variations in initial slow-start procedures. For this reason, MesoNetHS ensures that all TCP flows use the same initial slow-start procedures.

### 3.2.7 Parameters Identifying Monitored Links

The configuration file closes with parameters that may be used to identify links in the topology for which MesoNetHS should count packet transmissions over time. The specific meaning of these parameters is given below in Sec. 3.3.7.

### 3.2.8 Reporting Parameter Settings

Prior to commencing a simulation, MesoNetHS writes the settings of parameter values to a text file in the directory in which model measurements will be reported. The file and directory naming conventions mirror those used for measurement files (see Sec. 3.3.1 below). The first blank in the general file naming form is replaced by the token “ConfigurationSettingsFor”. So, for example, parameter settings will be written to a file with a name similar to “ConfigurationSettingsForRun0TimeStamp41540.txt”. This file also reports the value of various derived parameters, which are computed from combinations of parameter settings.

## 3.3 Model Measurements

MesoNetHS measures many facets of the spatiotemporal evolution of a simulated network. This section outlines the measurement approach and details some specific measures taken and reported during MesoNetHS simulations. The measures fall into six general categories. Summary measures report selected information that indicates general workload and performance during a simulation. Aggregate measures describe the temporal evolution of the entire network, as it unfolds during a simulation. Flow-class measures indicate the average temporal evolution of each of the six flow classes (recall 3.1.2.3) simulated by MesoNetHS. Long-lived flow measures follow the temporal evolution of any specific long-lived flows (recall 3.2.5) configured in a particular simulation. Per-router measures monitor the temporal evolution of the behavior of each individual router within a simulated topology. Optional, link-level measures report the temporal progression of traffic on selected links within a simulated topology. To support the needs of specific experiments, MesoNetHS can also be augmented to provide measures that do not fall into any of the predefined categories. Below, we describe selected MesoNet measures reported in each category, as well as how such measures may

be augmented. Prior to describing specific measures, we discuss the general measurement regime, including file-naming conventions, applicable to all categories of measures.

### 3.3.1 General Measurement Regime

Most of the measurements taken by MesoNetHS are reported as time series, where each data point in each series relates to a designated measurement interval. Thus, a typical measurement file will consist of a list of rows, where each row represents a measurement associated with a specific time interval. The first column in each row records the measurement interval and the remaining columns denote specific measures taken during that interval. Given this approach, one may plot each time series and may also compute statistical summaries (such as the average, median and variance) over all or part of each time series. With the exception of summary measures, MesoNetHS does not report column headers; thus, one should consult the descriptions below to understand the contents of each measurement file.

MesoNetHS uses a convention to name measurement files. The general file naming form is: “\_\_\_\_Run\_TimeStamp\_\_\_\_.txt”, where the first blank is replaced by the name of the specific measure, the second blank is replaced by the value of `exID`, and the third blank is replaced by a timestamp taken from the real-time clock of the computer executing the associated simulation. All measurement files for a specific simulation run are placed into a single directory, which is named using the following general form: “Run\_TimeStamp\_\_\_\_”. For example, if a simulation runs with `exID` = 0 and a real-time clock value of 34168, then the simulation would generate measurement files in directory “Run0TimeStamp34168”. One file in the named directory would be labeled as “ActiveFlowsRun0TimeStamp34168.txt”. This particular file reports the temporal evolution of the number of active flows (as discussed below in Sec. 3.3.3). The named directory would also contain additional files reporting other measures, as discussed below.

### 3.3.2 Summary Measures

MesoNetHS emits three files containing summary measures. The file that begins with the name “TotalPackets” reports two rows per measurement buffer (recall Sec. 3.2.1). The first row is a header that defines the columns in the second row. The six columns, from left to right, are: (a) the total number of measurement intervals covered by the row, (b) the total number of data packets input to the network, (c) the total number of data packets output from the network, (d) the total number of flows started during the simulation, (e) the total number of flows completed during the simulation and (f) the average number of SYN packets that were required to establish a successful connection. This information gives a general idea of the workload to which the simulated network was subjected.

The file that begins with the name “FlowThroughputsByType” reports two rows per measurement buffer. The first row is a header that defines the columns in the second row. Each row contains six pairs of columns, where each pair is associated with a flow class. The first column in each pair reports the total number of completed flows of the class and the second column reports the average throughput for a completed flow of the class. Flow classes are given in the following order: **DD**, **DF**, **DN**, **FF**, **FN**, and **NN**. These measures give an idea about the general user experience in each flow class. MesoNetHS measures flow these throughputs as the file size (in packets) divided by the

number of time steps required to send the file and then divided by the `basicTimeUnit` defined for the simulation configuration.

The file that begins with the name “SYNrateByType” reports two rows per measurement buffer. The first row is a header that defines the columns in the second row. Each row contains six columns, where each column gives the average number of SYN packets that were required to establish a successful connection for each flow class. Flow classes are reported in the following order: **DD**, **DF**, **DN**, **FF**, **FN**, and **NN**. These measures give a general idea of the congestion faced by each particular flow class.

### 3.3.3 Aggregate Measures

MesoNetHS reports at least 17 aggregate measures depicting spatiotemporal evolution of a simulated network. See Table 3-4 for a summary. Each associated measurement file has the same general format: a series of two-column rows. The first column in a row gives the measurement interval when the measure was recorded and the second column gives the reported measure. Aggregate measurement files contain one row for every measurement interval in a simulation run. Below we describe each aggregate measure.

The file beginning with the name “ActiveFlows” reports the number of flows that were in the process of transferring packets in the network at the end of each measurement interval. This excludes flows that are attempting to establish connections, which are reported in the file starting with “FlowsConnecting”.

The file beginning with the name “AverageSRTT” reports the average smoothed round-trip time across all active flows at the end of each measurement interval. The average smoothed round-trip time (SRTT) is influenced by the network diameter and link propagation delays, as well as by queuing delays within each router on the path of each flow. To remove the influence of fixed properties (e.g., network diameter and link propagation delays) one could subtract the average round-trip propagation time of the simulated topology from the SRTT. Doing so would estimate the average queuing delay on network paths.

The file beginning with the name “ConnectionFailures” reports the total number of connection attempts that failed during each measurement interval. One could divide the number of connection failures by the number of active connections plus the number of connection failures to compute a connection-failure rate for each measurement interval.

The file beginning with the name “FlowCongestionWindowOverTime” reports for each measurement interval the average size (in packets) of the congestion window across all active flows. In general, a larger congestion window suggests less network congestion and higher throughputs on flows.

The file beginning with the name “FlowsAboveThreshold” reports for each measurement interval the number of flows that are operating with congestion-control regimes different from standard TCP. Most alternate congestion-control mechanisms prescribe normal TCP congestion-control rules until the congestion window passes a particular threshold value. Thus, this measure reports the number of flows operating past the associated threshold. Note that the threshold value is likely to be different for various alternate congestion-control mechanisms (see Sec. 5). Flows operating under standard TCP Reno congestion-control procedures are reported in the file beginning with

“FlowsInNormalCongestionMode”. Flows operating within the initial slow-start phase are reported in the file beginning with “FlowsInInitialSlowStart”.

**Table 3-4. Summary of Aggregate Measures Reported by MesoNetHS**

Aggregate Measure	General Definition
Active Flows	Number of flows that are in the process of transferring data packets
Average SRTT	Average smoothed round-trip time across all active flows in the network
Connection Failures	Number connection attempts that failed within a measurement interval
Flow Congestion Window	Average congestion-window size across all active flows in the network
Flows Above Threshold	Number of active flows operating with alternate congestion-control procedures
Flows Completed	Number of flows completed within a measurement interval
Flows Connecting	Number of flows that are in the process of connection establishment
Flows in Initial Slow Start	Number of active flows that are in the initial slow-start phase
Flows in Normal Congestion Mode	Number of active flows operating with standard TCP Reno congestion-control procedures
Loss Rate	Ratio of (packets input – packets output)/packets input for a measurement interval
NAKs	Average number of NAK packets received by each active flow in the network
No Receivers	Number of instances in a measurement interval where a source could not find an available receiver under a chosen access router
Packets In	Number of data packets entering the network during a measurement interval
Packets Out	Number of data packets exiting the network during a measurement interval
Retransmission Rate	Ratio of file size to number of data packets transmitted averaged over each flow completing during a measurement interval
Timeouts	Average number of timeouts recorded by each active flow in the network
Window Increases	Average number of window increases recorded by each active flow in the network

The file beginning with the name “FlowsCompleted” reports the number of flows that were completed during each measurement interval. Since the number of flows completed depends to some extent on the number of active flows, one could compute the flow-completion rate as the ratio of the number of flows completed over the number of active flows plus the number of flows completed. As the flow-completion rate approaches 50% the simulated network is operating without much congestion. As the flow-completion rate approaches zero the simulated network is quite congested.

The file beginning with the name “LossRate” reports for each measurement interval the ratio of data packets input minus data packets output to data packets input. This gives a rough approximation of the rate at which a simulated network loses packets.

The approximation is rough because the network also buffers packets in queues and on backbone links; thus, packets input in one measurement interval might leave the network in some future interval. Further, the loss rate considers only data packets, while other packet types may also be lost. Over time, the average network loss rate should be reasonably accurate as data points in the time series represent a sequence of rough approximations oscillating around the average.

The file beginning with the name “NAKs” reports the average number of NAK packets received by each flow during each measurement interval. Generally, as the NAK rate increases, the network is becoming increasingly congested. Of course, a congested network may lose an increasing number of NAKs, so severe congestion also exerts downward pressure on the NAK rate. Further, a higher number of active flows cause a given number of NAKs to be prorated over a larger number of flows.

The file beginning with the name “NoReceivers” reports for each interval the number of times a source found no available receiver under an access router selected as the destination for a flow. When a simulation is configured properly and when the destination-selection algorithm is working correctly, this time series should consist of all zeros. This measurement has no value beyond a check that a model is properly configured and uses a reasonable algorithm for selecting destinations.

The file beginning with the name “PacketsIn” records for each measurement interval the total number of data packets injected into a simulated network topology. This measure is also used to compute the loss rate. The number of packets input gives a reasonable picture of the network load in terms of packets per time step. Of course, the actual network load is double the reported number of packets input because data packets stimulate ACK and NAK packets in reply. ACK and NAK packets are not included in the count of input packets. So, for example, a simulation reporting an input of one million packets per time step is actually experiencing a load of two million packets per time step. The file beginning with the name “PacketsOut” records for each measurement interval the total number of data packets exiting a simulated network topology. This measure is also used to compute the loss rate. As with packets input, packets output does not count ACK and NAK packets. One could compute an average network utilization by summing the rates of all simulated access routers and multiplying that sum by the measurement interval size and then dividing that multiplied sum into twice the number of packets output. This should yield a utilization value between zero and one<sup>4</sup>. For example, assume a measurement interval size of 200 time steps. Then, given the topology shown in Fig. 3-1, and given that the 105 N-class access routers operate at 200 ppts, the 28 F-class access routers operate at 400 ppts and the 6 D-class access routers operate at 2000 ppts, then the entire network can output at most  $(105 \times 200 + 28 \times 400 + 6 \times 2000 =)$  44200 ppts. (Note that, assuming a `basicTimeUnit` = 0.001 and a packet size of 1500 bytes, this equates to about  $5 \times 10^{11}$  bits per second.) In one measurement interval then, the simulated network could output at most  $(200 \times 44200 =)$  8840000 packets. Assume that the average number of packets output per measurement interval, computed from a time series of packets output, is 814985, then the average, aggregate network utilization can be estimated as  $(814985 \times 2 / 8840000 =)$  0.184.

---

<sup>4</sup> Note that since the computation is based on direct average measurement of data packets and an estimate for acknowledgment packets (as twice the number of data packets) the resulting utilization estimate is somewhat crude and could sometimes yield a value above one.

The file beginning with the name “RetransmissionRate” records the average retransmission rate across all flows completing in each measurement interval. The retransmission rate for a completed flow is computed as the ratio of the number of data packets actually sent on a flow minus the file size (in data packets) to the file size. Data packets sent beyond the file size comprise retransmitted packets arising from lost (data or ACK or NAK) packets or from timeouts. As a reasonable approximation, the average retransmission rate for a network should be somewhere around twice the loss rate (because both data and acknowledgment packets may be lost). In general, increased network congestion leads to increased losses, which lead to an increased rate of retransmissions. A lower rate of retransmissions indicates lower network congestion.

The file beginning with the name “Timeouts” records the average number of timeouts incurred on each active flow in each measurement interval. In MesoNetHS, timeouts arise most commonly when a data or acknowledgment packet is lost at the end of a data transfer. This occurs because no additional data is sent on a flow to stimulate additional acknowledgments. (Recall that MesoNetHS does not simulate procedures such as keep-alive acknowledgments.) Less frequently, timeouts occur when a path experiences unexpectedly large queuing delays. More rarely, timeouts might result when a complete congestion window of packets are lost. In general, an increased rate of timeouts suggests an increased level of network congestion.

The file beginning with the name “WindowIncreases” records the average number of times each active flow increases its window during a measurement interval. In general, a higher rate of window increases suggests lower network congestion. Of course, the rate of window increases might also be influenced by network capacity and round-trip times.

### 3.3.4 Flow-Class Measures

MesoNetHS reports at least 13 flow-class measures depicting the average temporal evolution of each flow class<sup>5</sup> in a simulated network. See Table 3-5 for a summary. Each associated measurement file has the same general format: a series of seven-column rows. The first column in a row gives the measurement interval when the measure was recorded and the next six columns give the reported measure for each flow class in the following order: **DD**, **DF**, **DN**, **FF**, **FN** and **NN**. Flow-class measurement files contain one row for every measurement interval in a simulation run. Below we describe each flow-class measure.

The file beginning with the name “ActiveFlowsByType” records the number of flows with data transfers in progress in each flow class. This measure gives a good idea about the distribution of flows between various types of access routers across the simulated network topology. The measured distribution of flow classes can be compared with the computed theoretical distribution of flow classes predicted from the simulation configuration. This measure also helps explain the degree of variance observed in temporal throughputs for flows of each class because some flow classes have decidedly fewer active flows than other flow classes. Further, this measure can be used to identify periods during which flows of some particular class were not active. A related file (“FlowsConnectingByFlowType”) reports the number of flows trying to connect in each class.

---

<sup>5</sup> Sec. 8 introduces some other measures associated with different techniques to classify flows based on various additional characteristics.

The file beginning with the name “ConnectionFailuresByType” records the number connection attempts that failed for each flow class during each measurement interval. This measure can be used to compute a connection-failure rate for each flow class and to compare failure rates across flow classes.

**Table 3-5. Summary of Flow-Class Measures Reported by MesoNetHS**

Flow-Class Measure	General Definition
Active Flows by Type	Number of active flows in each flow class
Connection Failures by Type	Number connection attempts that failed in each flow class
Flow Congestion Window by Type	Average congestion-window size for active flows in each class
Flow Retransmission Rate	Ratio of file size to number of data packets transmitted averaged over each completing flow in each class during a measurement interval
Flows Above Threshold by Flow Type	Number of active flows operating with alternate congestion-control procedures in each flow class
Flows Completed by Type	Number of flows completed in each flow class
Flows Connecting by Flow Type	Number of flows of each flow class that are trying to connect
Flows In Initial Slow Start by Flow Type	Number of active flows in each flow class that are operating within initial slow start
Flows In Normal Congestion Mode by Flow Type	Number of active flows in each flow class that are operating under standard TCP Reno congestion-control procedures
NAKs by Flow Type	Average number of NAK packets received by each active flow in each flow class
Temporal Flow Throughputs	Average packets per time step output by flow class, divided by the <code>basicTimeUnit</code>
Timeouts by Flow Type	Average number of timeouts recorded by each active flow in each flow class
Window Increases by Flow Type	Average number of window increases recorded by each active flow in each flow class

The file beginning with the name “FlowCongestionWindowByType” records the average congestion window for active flows in each flow class. This measure can be used to compare congestion windows across flows of various classes. Congestion-window size is influenced by a complex collection of factors; thus, becomes an interesting measure to study.

The file beginning with the name “FlowRetransmissionRate” records the average retransmission rate for completed flows in each flow class. The measure is computed using the same relationships defined for aggregate retransmission rate.

The file beginning with the name “FlowsAboveThresholdByFlowType” records the number of flows in each flow class that are operating with congestion-control procedures other than standard TCP. The ratio of this measure to the measure of active flows by type reveals the proportion of active flows in each flow class that are operating with a congestion window above the threshold defined for the appropriate congestion-

control procedures used for each flow. Related files report the number of active flows in each class operating in initial slow-start phase (“FlowsInInitialSlowStartByFlowType”) and also operating under standard TCP Reno congestion-control procedures (“FlowsInNormalCongestionModeByFlowType”).

The file beginning with the name “FlowsCompletedByType” records the number of flows in each flow class that were completed during each measurement interval. This measure may be combined with the measure of active flows by type to compute a flow completion rate for each flow class.

The file beginning with the name “NAKsByFlowType” records the average number of NAKs received by a flow of each class during each measurement interval.

The file beginning with the name “TemporalFlowThroughputs” records the average instantaneous throughput of flows in each flow class. This measure can be divided by the average file size (computed from the simulation configuration) for each flow class to estimate the average time taken by each flow class to transfer an average size file. For example, if a flow class has an average file size of 50 packets (75 Kbytes) and an average throughput of 100 packets (1.2 Mbits) per second, then it would take  $(50/100 =) 1/2$  second to transfer an average size file.

The file beginning with the name “TimeoutsByFlowType” records the average number of timeouts that occur for a flow of each class in each measurement interval. Similarly, the file beginning with the name “WindowIncreasesByFlowType” records the average number of window increases received by a flow of each class in each measurement interval.

### 3.3.5 Long-Lived Flow Measures

MesoNetHS reports up to 8 long-lived flow measures depicting the temporal evolution of each long-lived flow configured in a simulated network. See Table 3-6 for a summary. Each associated measurement file has the same general format: a series of multicolumn rows. The first column in a row gives the measurement interval when the measure was recorded and each of the remaining columns report a measure associated with a specific long-lived flow. The recording order corresponds to the order in which long-lived flows were defined in the simulation configuration. Below we describe each measure recorded for long-lived flows.

The file beginning with the name “LongLivedFlowCongestionMode” records the congestion-control procedures being used on each long-lived flow at the end of each measurement interval. When a long-lived flow is inactive, the measure reports the value “NONE”. When a long-lived flow is active but using standard TCP congestion-control procedures, the measure reports the value “NORMAL”. Otherwise, the measure reports the specific congestion-control procedure in use on each long-lived flow. Valid values for this measure include: “BIC”, “COMPOUND”, “FAST”, “H”, “HS” and “SCALABLE”. For long-lived flows operating under FAST congestion-control procedures a separate measurement file (“LongLivedFlowFASTalpha”) reports the evolution of the value of the FAST  $\alpha$  parameter (see Sec. 5).

The file beginning with the name “LongLivedFlowCWNDs” reports the size of the congestion window at the end of each measurement interval for each long-lived flow. The file beginning with the name “LongLivedFlowNAKs” records the number of NAKs received by each long-lived flow in each measurement interval. The file beginning with

the name “LongLivedFlowTimeouts” records for each long-lived flow the number of timeouts experienced in each measurement interval. The file beginning with the name “LongLivedFlowWindowIncreases” records for each long-lived flow the number of increases in the congestion window that occurred in each measurement interval. The file beginning “LongLivedFlowSRTT” reports the evolution of the smoothed round-trip time for each long-lived flow.

**Table 3-6. Summary of Long-Lived Flow Measures Reported by MesoNetHS**

Long-Live Flow Measure	General Definition
Congestion Mode	Congestion-control procedures in use on each long-lived flow at the end of each measurement interval
Congestion Window	Size of the current congestion window for each long-lived flow at the end of each measurement interval
FAST Alpha	Value of the FAST $\alpha$ parameter at the end of each measurement interval for each long-lived flow that operates under FAST congestion-control procedures
NAKs	Number of NAKs received in each measurement interval for each long-lived flow
SRTT	Value of the SRTT for each long-lived flow at the end of each measurement interval
Throughputs	Average packets per time step output by each long-lived flow, divided by the <code>basicTimeUnit</code>
Timeouts	Number of timeouts recorded in each measurement interval for each long-lived flow
Window Increases	Number of window increases recorded in each measurement interval for each long-lived flow

The file beginning with the name “LongLivedFlowThroughputs” reports for each long-lived flow the average throughput over each measurement interval. Average throughput<sup>6</sup> is computed as the number of ACK and NAK packets sent by a flow’s receiver in a measurement interval divided by the measurement interval size and then divided by the `basicTimeUnit` configured for the simulation.

### 3.3.6 Per-Router Measures

MesoNetHS reports measurements associated with each router in the simulated network topology. For the topology shown in Fig. 3-1, this would be 172 routers. MesoNetHS reports per-router measures in three classes: (a) backbone routers, (b) POP routers and (c) access routers. Six measures are reported for backbone and POP routers, while 12 measures are reported for access routers. This section describes the relevant measures and related measurement files. The description is divided into two parts: (a) measures common to all routers and (b) additional measures recorded only for access routers.

*3.3.6.1 Measurements Common to All Routers.* MesoNetHS reports six measures for all routers. Table 3-7 gives a summary. Two measures (flows completed and packets forwarded) are aggregate measures summed over all measurement intervals. The

<sup>6</sup> This measure is often referred to as goodput.

remaining measures are time series reported at each measurement interval for each router. The measures are described below, beginning with the aggregate measures.

Files reporting aggregate router measures contain two rows for each measurement buffer. The first row is a header identifying the number of measurement intervals over which the measures were aggregated and the second row contains an aggregate measure for each router. For the topology shown in Fig. 3-1, aggregate measurements for backbone routers would contain 11 columns (routers A through K) while aggregate measurements for POP routers would contain 22 columns (routers A1 through K2) and for access routers would contain 139 columns (routers A1a through K2d).

**Table 3-7. Summary of Measures Reported by MesoNetHS for Each Router**

Router Measure	General Definition
Active Flows	The number of flows transiting each router at the end of each measurement interval
Flows Completed	The aggregate number of flows carried by each router over all measurement intervals
Losses	The number of packets discarded by each router in each measurement interval
Queue Length	For each router, the ratio of packets queued to buffer size at the end of each measurement interval
Packets Forwarded	The aggregate number of packets forwarded by each router over all measurement intervals
Utilization	The average utilization for each router over each measurement interval

The file beginning with the name “BackboneRoutersFlowsCompleted” gives the aggregate number of flows that transited each backbone router over all measurement intervals. Similar files, beginning with the names “SubnetRoutersFlowsCompleted” and “LeafRoutersFlowsCompleted”, give the same measure for each POP<sup>7</sup> and access router, respectively. The file beginning with the name “BackboneRouterPacketsForwarded” records the aggregate number of packets forwarded by each backbone router over all measurement intervals. Similar files, beginning with the names “SubnetRouterPacketsForwarded” and “LeafRouterPacketsForwarded”, report the same measure for each POP and access router.

Files reporting the spatiotemporal evolution of routers contain a time series with one row for each measurement interval. Each row contains multiple columns. The first column gives the measurement interval with which the row is associated and the remaining columns give a measure for each appropriate router, depending upon category. Thus, for the topology shown in Fig. 3-1 time series related to backbone routers would contain rows of 12 columns, while time series related to POP routers would contain rows of 23 columns. Similarly, any time series related to access routers would contain rows of 140 columns.

The file beginning with the name “BackboneRoutersActiveFlows” records the number of active flows transiting each backbone router at the end of each measurement interval. Similarly, files beginning with the names “SubnetRoutersActiveFlows” and

<sup>7</sup> MesoNetHS code refers to second-tier routers as Subnet routers rather than POP routers and refers to third-tier routers as Leaf routers rather than access routers.

“LeafRoutersActiveFlows” report the number of active flows transiting each POP and access router, respectively. The file beginning with the name “BackboneRouterLosses” records the number of packets discarded by each backbone router in each measurement interval. Similar files beginning with the names “SubnetRouterLosses” and “LeafRouterLosses” records drop by each POP and access router.

The file beginning with the name “BackboneRoutersQLength” reports the ratio of packets queued to buffer size for each backbone router at the end of each measurement interval. Similar files, beginning with the names “SubnetRoutersQLength” and “LeafRoutersQLength” report the same ratio for each POP and access router. The file beginning with the name “BackboneRouterUtil” records the ratio of packets forwarded to capacity for each backbone router in each measurement interval. Similar files, beginning with the names “SubnetRouterUtil” and “LeafRouterUtil” record the same ratio for each POP and access router.

*3.3.6.2 Measurements Unique to Access Routers.* MesoNetHS reports six additional measures that are recorded only for access routers. These additional measures relate to the activity of flows transiting specific access routers. Table 3-8 gives a summary.

**Table 3-8. Summary of Added Measures Reported by MesoNetHS for Access Routers**

Access-Router Measure	General Definition
Connection Failures	The number of failed connection attempts for flows transiting each access router
NAKs	The average number of NAKs on flows transiting each access router
No Receivers	The number of instances when no receivers were available under each router
SYN Rate	The ratio of SYNs sent to first SYNs sent on flows transiting each access router
Timeouts	The average number of timeouts on flows transiting each access router
Window Increases	The average number of window increases on flows transiting each access router

The file beginning with the name “LeafConnectionFailures” reports the number of connection failures during a measurement interval for flows that would have transited each access router. This includes flows where either an intended source or receiver was subordinate to the access router. The file beginning with the name “LeafNAKs” records for each measurement interval the average number of NAKs received on flows transiting each access router. Similar files, beginning with the names “LeafTimeouts” and “LeafWindowIncreases”, report for each measurement the average number of timeouts and congestion-window increases, respectively, on flows transiting each access router. The file beginning with the name “LeafNoReceivers” reports for each measurement interval the number of times each access router could not accommodate a flow because no receiver was available. Finally, the file beginning with the name “LeafSYNrateLeaf” records for the ratio of SYNs sent to first SYNs sent during connection establishment procedures involving each access router. This file reports the average SYN rates each time a measurement buffer is dumped; dump is preceded by a header line indicating the number of measurement intervals over which the SYN rate was averaged.

### 3.3.7 Optional, Link-Level Measures

MesoNetHS allows an experimenter to select up to one simulated (unidirectional) link to monitor in each router tier. Link selection is controlled by three parameters in the configuration file. Parameter `BB_LINK_TO_MONITOR` identifies which backbone link in the topology will be monitored. Backbone links are numbered sequentially for a specified topology. For example, given Table 3-1, setting `BB_LINK_TO_MONITOR = 1` would monitor the link from backbone router A to B, while setting the parameter to 2 would monitor the link from backbone router B to A. Similarly, assigning a valid POP-router identifier from a topology to `POP_LINK_TO_MONITOR` will activate monitoring on the incoming link from a parent backbone router to the designated POP router. POP routers are identified with sequentially increasing integers from one (POP router A1 in Fig. 3-1) to the number of POP routers (i.e., the identifier is 22 for POP router K2 in Fig. 3-1). A third parameter, `ACCESS_LINK_TO_MONITOR`, controls monitoring on an incoming link from a parent router (either backbone or POP) to an access router. The link is specified by assigning `ACCESS_LINK_TO_MONITOR` a valid access-router identifier. Access routers are identified with sequentially increasing integers from one (access router A1a in Fig. 3-1) to the number of access routers (i.e., the identifier is 139 for access router K2d in Fig. 3-1). By default all link-selection parameters are set to zero, which means that no links are monitored.

When link monitoring is active, MesoNetHS records the number of packets transiting each monitored link during each measurement interval and writes this information as a time series, where each row contains one two-column observation. The first column identifies the measurement interval and the second column gives the number of packets observed transiting the associated link during the measurement interval. The file beginning with the name “MonitoredBBLink” contains the time series for the specified backbone link. Similar files, beginning with the names “MonitoredSubnetLink” and “MonitoredLeafLink”, hold the time series for the specified POP and access links. Link-monitoring files will not be produced if link monitoring is inactive.

### 3.3.8 Augmenting Measures

MesoNetHS can be augmented by an experimenter to make specific measures that are not already incorporated. The purpose of this section is to describe the general approach one should take to accomplish such augmentation. The approach is illustrated by an example involving capturing a traffic-flow matrix. This augmentation is already incorporated into MesoNetHS as a set of comments. Reviewing these comments should provide further guidance regarding how to augment the measures recorded by the simulation.

To extend measurements made by MesoNetHS, one must add: (a) an array to hold the measurement of interest, (b) code to write the measurements to disk, (c) code to clear the measurement array and (d) in-line code to make the required measurements at the appropriate points in the model. To illustrate how this might be done, we consider an example where an experimenter decides to monitor traffic flows from selected observation points in a network to each access router.

First, one would define an array to hold the measurement of interests. In this case, the definition is for a three-dimensional array:

```
int inBoundPackets[NUMBER_OBSERVATION_POINTS][LEAF_ROUTERS][M_BUFFERS]
```

used to count packets. The first dimension is the number of observation points; the second dimension is the number of access (leaf) routers; the third dimension is the number of measurement intervals in one measurement buffer. In this case, the number of observation points is defined to equal the number of POP routers plus the number of directly connected access routers. The definition of the number of observation points is applied automatically, as long as the number of POP routers and the number of directly connected access routers is specified in the topology file.

Next, one adds code to the `write_measurements()` procedure. The added code must do four things: (1) define a name for the file in which measurements are recorded, (2) open the file, (3) append the measurements to the file and (4) close the file. The conventions for naming measurement files are to prepend a directory name (`Dname`) to the file name chosen by the experimenter and to append a time stamp (`RTC`), followed by the extension “.txt”. The values for `Dname` and `RTC` are already defined by the program, so the experimenter must use them. Here is an example of the code to define the name for the file to record a traffic-flow matrix:

```
write string=Fname(Dname, RTC) "../FlowMatrix_.txt".
```

This code places the constructed file name into the variable `Fname`. This must be followed by a statement that opens the file for append. Then one provides code to loop through the three-dimensional array and write out each measurement. In this case, each line written will contain four fields: (1) the number of the observation point where the traffic was observed, (2) the number of the access router to which the traffic was bound, (3) the measurement interval in which the observation was made and (4) the number of packets observed. The measurement-writing code automatically tracks the starting (`previousEnd`) and ending (`currentEnd`) measurement intervals for the measurement buffer; thus, the measurement interval is identified by adding the appropriate loop-control variable to the variable `previousEnd`. (For more details, the reader should see the related source code at the end of the `write_measurements()` procedure.) After writing out the buffer, the file must be closed because the file descriptor is reused for each file that is written.

The experimenter must also provide code in the `clear_measurements()` procedure so that the measurement buffer can be cleared after it is written to disk. The exact nature of the clear code depends on the construction of the measurement array. At the outermost level, the clear procedure loops through the measurement intervals ( $i$ ) in the buffer. Thus, for arrays dimensioned only on time, one can simply add code in this outer loop. The clear procedure also loops through various second ( $j$ ) dimensions (e.g., POP routers, access routers, backbone routers, flow types and long-lived flows). Due to this, one can add code to clear the flow matrix under the loop through access routers. However, the flow matrix has a third ( $k$ ) dimension (observation points), so one must add a loop over the observation points. Inside this innermost loop, one simply sets `inBoundPackets[k, j, i]` equal to zero.

All that remains is to add code to record the required measurements. This must either be done in-line in elements of the model or in the forever loop within the actions clause of the class `StateMonitor`. Where to add the measurement code depends on the nature of the measurements. Sample-oriented measurements, taken periodically, should

be added to the class `StateMonitor`, while event-oriented measurements, recorded when they occur, should be added in-line within the appropriate model elements. Recording of the flow matrix requires event-oriented measurements, so code must be added in-line. In this case, we wish to record each packet that successfully reaches an outgoing access router when sent from each source or receiver. This means that we must increment the `inBoundPackets` array in both sources and receivers. For a source, this requires incrementing the array each time a packet is injected successfully. A source injects packets in four places: (a) upon initially attempting to connect (SYN), (b) upon retrying a connect attempt (SYN), (c) when initially sending a data (DT) packet after becoming connected and (d) when sending each subsequent data (DT) packet. Thus, one must add a line of code to increment the array in each of these places. For a receiver, there is only one place where packets are injected into the network; thus, one must add a line of code at this point to increment the array. (The reader can find these five places in the model code by searching for `inBoundPackets`).

### 3.4 Tracing Flow Behavior

To support debugging or to enable monitoring of individual behavior of flows, MesoNetHS provides facilities to print traces associated with selected flows. The model traces only one flow at any given time and randomly selects which flows to trace. When the flow currently being traced is closed, then tracing also ceases for that flow. When a flow turns on and no flow is currently being traced, then the new flow is selected for tracing. (If one knows the identity of a specific flow that should be traced, then the tracing variable may be set directly in the code to ensure that the desired flow is traced.)

In general, the tracing of flow behavior is disabled<sup>8</sup> because copious file writes occur and the model can be slowed significantly or (for long runs) can produce massive amounts of trace information. To enable flow tracing, one needs to define a symbol `TRACE_TCP`. This symbol is already defined in the code; however, by default the symbol is commented out – thus, to activate tracing, one simply must uncomment this symbol. Once tracing is activated, MesoNetHS will generate two files: (a) one file, whose name begins with “TCPstate”, that records the values of flow state variables at the time of particular events by the source associated with the flow being traced and (b) a second file, whose name begins with “TCPmessages”, that records each packet sent or received by the source associated with the flow being traced.

#### 3.4.1 Tracing Flow States

For each flow being traced, MesoNetHS records the values of the variables defining the state of the flow’s source. These state variables are recorded each time a significant event occurs. Significant events include: (a) initial congestion window (CWND) established, (b) CWND increased, (c) CWND decreased and (d) timeout. Additional recording is possible by defining a symbol `WINDOW_CHECK`. When this symbol is defined, state variables will also be recorded each time an ACK is received and a check is made regarding whether or not the CWND should be increased.

Each state recording consists of a single line in the associated file. The line contains 22 state variables. The first variable (Time) gives the time step when the event

---

<sup>8</sup> The reader is advised to activate flow tracing only for small, short simulations.

occurred, while the second variable (Event) identifies the stimulating event. The next six variables describe the flow, giving the source (Tx) and receiver (Rx), the type of congestion-control algorithm (Type) used on the flow, the flow class (Flow), the time the flow started (OnAt) and the number of packets (MustDeliver) in the flow. Two additional variables outline the general progress of the flow, including (UnDelivered) how many flow packets remain undelivered (i.e., have not yet been acknowledged by the receiver) and (DTsSent) how many flow packets have been sent. The remaining dozen values give the detailed state of the flow variables, including (Phase) the phase of the flow (e.g., slow start or congestion avoidance), the current size (cwnd) of the congestion window and the current value (ssthresh) of the slow-start threshold. These state variables also include (unSentDTs) the number of packets that can be sent by the source and (unACKed) the number of unacknowledged packets that have been sent. Also provided are the values of four sequence numbers: (a) (nexSEQ#) the next sequence number that may be sent by the source, (b) (HighestACK) the highest sequence number from the receiver, (c) (lastNAK) the value of the source's next sequence number to be sent when the last NAK was processed by the source and (d) (TOseq) the value of the source's next sequence number when the last timeout occurred. The remaining three state variables relate to establishing a timeout period. These variables include the next time step (CRTO) when a timeout will occur, the number of time steps (RTO) that will be added to the current time to establish the time of the next timeout and the latest estimate of the smoothed round-trip time (SRTT) measured on the flow.

### 3.4.2 Tracing Packets

MesoNetHS also traces packets as they are sent and received by the source on each flow that is traced. Each packet transmission and reception is recorded on one line of the associated trace file. Each line records five variables from the packet. The recorded variables include: the time step (Time) when the packet was sent or received, the source (Tx) and receiver (Rx) of the flow associated with the packet, the type (Type) of the packet and the sequence number (SQ#) of the packet. Note that packets sent by the receiver will be recorded only if and when they reach the source. On the other hand, packets sent by the source will be recorded when they are sent, regardless of whether or not they reach the intended receiver.

## 3.5 Notes on Model Construction with SLX

This subsection provides a brief guide to the model as constructed using the SLX [81-82] simulation language and development environment. The intent of this section is to guide those who wish to review the model source code. This section is not intended to provide a detailed description of the code. MesoNetHS is constructed from three SLX files: (a) a configuration file, (b) a topology file and (c) a file defining behavior of model elements. Each of these files is described below in a separate subsection. The file descriptions are followed by a short discussion of the performance properties of MesoNetHS.

### 3.5.1 Configuration File

The configuration file (e.g., MesoNetHSconfigDE.slx) provides the vehicle for defining configuration parameters, explained in Sec. 3.2. The configuration-file parameters are grouped (using comments) with the same headings as given in the subsections of Sec.

3.2; the parameter names in the source file conform to the parameter names used in those subsections. To support sensitivity analyses and other experiments, the model configuration file may be broken into two parts containing: (a) model parameters that remain fixed across experiment runs and (b) model parameters that change with each run, as guided by an experiment plan. In such cases, the variable portion of the file may be constructed by a configuration generator. This approach was used for the experiments described in subsequent sections of this report. When reviewing the source code associated with these experiments, one will likely find two configuration files for each run, where the parameters described in Sec. 3.2 are divided among the two files depending upon whether the parameters remain fixed or are varied from run to run.

### 3.5.2 Topology File

The topology file (e.g., `MesoNetHS-AbileneTopologyI1c.slx`) defines the layout and characteristics of routers and links under which sources and receivers will be deployed. The information in the topology file is used at the start of a simulation to construct a simulated topology. A topology file begins by defining the number and type of routers included in the topology, as well as the number of backbone links. The file also includes some type definitions to define the classes of routers that may attach to backbone routers and to POP (subnet) routers, as well as the classes of access (leaf) routers included in the topology. For each backbone link, the topology file defines (see array `LP_DELAY`) the one-way propagation delay (in time steps). These delays are scaled by the value of the `deltaX` parameter, which may alter the link propagation delays. Since the `deltaX` parameter is defined in the configuration file, the topology file must be included after the configuration file.

The topology file also contains a 2-D matrix (`FORWARDING_LINK`) defining the backbone link over which packets should be forwarded when bound between two backbone routers. The first dimension represents the source backbone router and the second dimension represents the destination backbone router. Two auxiliary matrices (`SOURCE_BACKBONE_ROUTER` and `SINK_BACKBONE_ROUTER`), which are indexed by backbone link, define the source and sink backbone router associated with each backbone link. This information is used to connect backbone routers and links when generating the topology.

Another 2-D matrix (`ROUND_TRIP_DISTANCE`) defines the round-trip times used to seed initial estimates of the round-trip delay in each direction between any pair of backbone routers in the topology. The estimates are computed by summing the one-way propagation delays associated with all backbone links transited along the forward and reverse path defined for each route between each pair of backbone routers. A variable (`EB_DELAY`) defines an estimated buffer delay (in time steps) that is added to the round-trip propagation delay in order to account for some amount of queuing delay that may be experienced on a route. The seed estimates from `ROUND_TRIP_DISTANCE` are used by flows as an initial guess for the round-trip time that might be expected on a path. The initial guess for the round-trip time is used to set the initial timeout period for a flow.

Connections of POP and access routers to backbone routers are described by a 2-D matrix (`SUBNET_PER`). The first dimension represents the backbone routers in the topology. The second dimension represents the class of routers (e.g., POP and directly connected access routers) that may connect to the backbone. The information contained

in this matrix is used to generate POP and directly connected access routers under each backbone router in the topology.

Connections of access routers to POP routers are specified by a three-dimensional matrix (*LEAF\_PER*). The first dimension represents a backbone router. The second dimension represents the maximum number of POP (subnet) routers that may exist under any backbone router. The third dimension represents the class of access routers (e.g., fast or normal) that may exist under a POP router. The information contained in this matrix is used to generate (fast and normal) access routers under each POP router in the topology. Several topology files have been defined for use with MesoNetHS.

### 3.5.3 Model Behavior File

The main model file defines the behavior of model elements (as discussed previously in Sec. 3.1.1) and the overall behavior of the simulation. The main model file also defines measurement buffers, parameter mappings, auxiliary procedures and sets into which model elements of various types are sorted. Below, we discuss these features of the model in the following categories: (a) model elements, (b) simulation control, and (c) measurement buffers. We do not discuss parameter mappings or sets, which should be obvious from examining the source code.

*3.5.3.1 Model Elements.* The primary model elements are defined using SLX “active” classes, each of which has an individual behavior defined within an “actions” block. The active classes include: (a) *LeafRouter*, (b) *SubnetRouter*, (c) *BackboneRouter*, (d) *BackboneLink*, (e) *Source* and (f) *Receiver*. The behavior of each of these classes mirrors the description given earlier in Sec. 3.1.1. Each SLX class also includes an “initial” block, which is executed when the class is created. The “initial” block acts as a class constructor, establishing initial conditions. One active class, *Source*, also contains two methods (*state* and *message*) that support flow tracing. (Note that all active classes defined in this model are self-activating because the last statement of the “initial” block activates the class.) One “passive” class, *Packet*, encompasses the remaining model element. The contents of the *Packet* class mirror the description given above in Sec. 3.1.3.

*3.5.3.2 Simulation Control.* The simulation is started and controlled from the SLX “main” procedure, which is located at the end of the source file. The “main” procedure is also supported by a few auxiliary classes and procedures, which are discussed as the need arises. Model execution begins by constructing the timestamp (*RTC*) and the directory name (*Dname*) used to disambiguate the model output files. If flow-tracing is enabled, the associated output files are also created and opened at this time. Subsequently, the topology is examined, using procedure *computeAverageRTT( )*, and the associated round-trip propagation delays are reported. Then the simulated topology is created, starting from the backbone routers downward. Each backbone router creates its own subordinate routers and those subordinate routers create their own children. The backbone links are constructed after the routers. Once the topology is constructed, the average buffer size is computed and reported for each router class (i.e., backbone, POP and access).

Next, the “main” procedure uses procedure *createLongLivedFlows( )* to set up any long-lived flows that have been defined in the configuration file. Each long-lived flow is scheduled, using procedure *scheduleLongLivedFlow( )*, which creates a new source and

receiver for the flow and sets initial conditions so that the flow is already connected and ready for data transfer at the desired time. As a final step, the procedure uses the SLX anonymous “fork” construct, which splits the processing into two independent “threads”, the calling thread and a forked thread. Upon returning from the procedure, the forked thread waits until the simulation reaches the time step when the long-lived flow should begin and then completes activation of the associated source and receiver.

The main procedure uses the procedure `writeConfiguration ( )` to generate a file containing the settings of MesoNetHS parameters for a given run. When adding parameters to MesoNetHS, one should also insert related parameter-reporting code in procedure `writeConfiguration ( )`. The inserted reporting code should follow the pattern of existing code within the procedure.

Prior to commencing the simulation, the “main” procedure also creates an instance of the `StateMonitor` class, which periodically makes measurements of the simulated system, manages the measurement buffers, writes measurement data to files and clears measurement buffers. To accomplish some of these operations, the `StateMonitor` class uses procedures `write_measurements( )` and `clear_measurments( )`. As the last step before starting the simulation, “main” reports the date and time when the simulation started. The simulation commences when the “main” procedure delays itself, using an SLX “advance” statement, for the duration of the configured simulation time. In fact, the delay is slightly longer than the required time in order to permit the final measurement interval to be taken. Upon completing the simulation, the “main” procedure reports the date and time the simulation finished<sup>9</sup>. If appropriate, the flow-tracing files are closed and the “main” procedure terminates.

*3.5.3.3 Measurement Buffers.* The measurement buffers defined for the simulation appear after the comment line reading “MEASUREMENT INFORMATION”. Note that some of the measurement buffers are guarded by `#ifdef` statements using the symbol `SUBNETS`. Similarly, one will find measurement statements within the source code also guarded by the same symbol. This permits these measurements to be skipped when a topology is defined without any POP routers.

### 3.5.4 Performance Properties of MesoNetHS

Performance of MesoNetHS is largely influenced by the performance of the SLX simulation compiler and run-time. As we will show, SLX performance is quite good. On the other hand, characteristics of the simulated configuration will also influence both processing time and memory requirements. We address these issues using samples from two experiments described in later sections (Sec. 4 and Sec. 6). Table 3-9 provides a summary of characteristic performance for MesoNetHS when used to conduct the two experiments. Both experiments adopt the topology presented earlier (recall Sec. 3.1.2).

In a sensitivity analysis experiment, MesoNetHS was used to simulate 20 minutes of network evolution with an average of about 30000 sources and 160000 receivers. In these simulations, backbone routers operated at either 4.8 Gbps or 9.6 Gbps (depending on the configuration). A typical run required simulating about 10 million flows and processing around 1.2 billion packets. At any given time, about 10000 flows were active and around 65000 packets were in transit across the network. For a simulation of this

---

<sup>9</sup> Later versions of MesoNetHS include logic to periodically estimate a projected completion time.

scale, MesoNetHS required about 5.7 hours of processing time and around 166 Mbytes of memory.

**Table 3-9. Characteristic Performance for MesoNetHS in Two Experiments**

	Experiment	
	Sensitivity-Analysis	Comparison-Robustness
Simulated Minutes	20	25
Sources (avg.)	28,810	226,300
Receivers (avg.)	160,790	1,675,200
Total Flows (avg.)	9,715,570	87,440,063
Total Packets (avg.)	607,814,302	5,548,334,397
Active Flows (avg.)	9,991	32,194
Packets in Transit (est. avg.)	63,291	350,949
CPU Hours (avg.)	5.7	67.0
Memory in Mbytes (avg.)	166	1,192

In a comparison-robustness experiment, MesoNetHS was used to simulate 25 minutes of network evolution with an average of about 225000 sources and 1.7 million receivers. In these simulations, backbone routers operated at either 96 Gbps or 192 Gbps (depending on the configuration). A typical run required simulating over 75 millions flow and processing nearly 14 billion packets. At any given time, about 32000 flows were active and around 350000 packets where in transit across the network. For a simulation of this scale, MesoNetHS required about 67 hours of processing time and around 1.2 Gbytes of memory.

The comparison-robustness experiment increased the simulated system size by about an order of magnitude and ran the simulation for 25% more simulated time, as compared with the sensitivity-analysis experiment. Thus, one might expect resource requirements to grow on the order of 12.5 times. The actual processing requirements grew by 11.75 times, which is within range of the estimate. (This nearly linear growth<sup>10</sup> in processing time may be attributed to the excellence of the SLX compiler and run-time environment.) The actual memory requirements increased only sevenfold. This increase was lower than the expected tenfold increase. The smaller than expected increase may be attributed to changes in the measurement strategy adopted between the two simulations. Further details are provided below: first about processing requirements and then about memory requirements.

**Table 3-10. Processing Requirements for MesoNetHS in Two Experiments**

	Experiment	
	Sensitivity-Analysis	Comparison-Robustness
CPU Time (s) per simulated flow-minute	0.17	0.49
CPU Time (us) per simulated packet	33	44

<sup>10</sup> In subsequent experiments (see Sec. 9 vs. Sec. 8), simulating much larger networks with much higher router speeds for one hour of network evolution, we found a 10-fold increase in network size and speed led to a 16-fold increase in processing time. We attribute this to a substantial increase in the size of the event lists that SLX needed to manage.

*3.5.4.1 Processing Requirements.* As shown in Table 3-10, for the sensitivity-analysis experiment, MesoNetHS required an average of 170 milliseconds of CPU (central-processing unit) time to process a simulated flow-minute, while requiring 490 milliseconds per flow-minute under the comparison-robustness experiment. Table 3-10 also shows that the processing time per packet increased (by 33%) from the sensitivity-analysis experiment to the comparison-robustness experiment. The increase in per-packet processing time may be attributed to the increase in the size of simulation state (e.g., event lists) that SLX needed to process under the comparison-robustness experiment. The exhibited increase in processing times between the two experiments is quite reasonable.

**Table 3-11. Memory Requirements (Mbytes) for MesoNetHS in Two Experiments**

	Experiment	
	Sensitivity-Analysis	Comparison-Robustness
Sources	16.8	100.6
Receivers	20.6	214.4
Sets and Membership	25.5	256.9
Simulation Processing State	26.7	251.6
Measurement Buffers	40.5	10.1
Other Memory	38.3	299.1

*3.5.4.2 Memory Requirements.* Table 3-11 shows the average allocation of memory by SLX for each of the two experiments. The memory allocated to sources and receivers reflects the number of each of these objects in the simulation. Sets and set membership includes memory associated with static model categories, as well as packet queues within routers, links, sources and receivers. Simulation processing state encompasses memory allocated to active objects, as needed to manage time evolution of the simulation. Measurement buffers are allocated as directed by the configuration file. The remaining (other) memory represents memory associated with the SLX run-time, with routers and links and with packets transiting the simulated network.

As shown in Table 3-11, and as expected, memory requirements generally expand by about an order of magnitude across the board. The exception is memory allocated for measurement buffers, which decreases for the comparison-robustness experiment to  $\frac{1}{4}$  the size required for the sensitivity-analysis experiment. This occurs because the sensitivity-analysis experiment allocates measurement buffers to cover 6000 measurement intervals, while the comparison-robustness experiment allocates memory for only 1500 measurement intervals. As a tradeoff, the measurement buffers must be written to disk five times during each run of the comparison-robustness experiment but only once at the end of the sensitivity-analysis experiment.