4.2 NetBench: Discrete Packet Simulator

To assess routing in static topologies, a packet simulator using discrete events was implemented named *NetBench*. It was created with especially the design principles of maintainability, extensibility, and understandability in mind. The three main modules are discussed in the following sections. First, the infrastructure features are described in section 4.2.1. Second, the routing initialization is described in section 4.2.2. Third, the traffic generation model is explained in section 4.2.3.

4.2.1 Infrastructure

The infrastructure of a simulation refers to the modeled physical components of the network. There are five main choices that need to be made: the link, the output port, the network device, the flowlet intermediary, and the transport layer. There is no model for input port queuing. The interplay between these five components is visualized in figure 4.1.

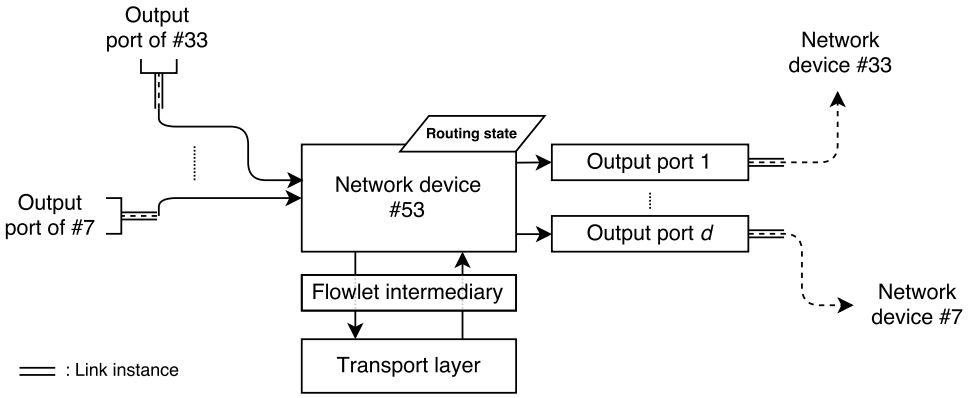

Figure 4.1: NetBench infrastructure example. Network device #53 has two bi-directional links to #33 and #7.

Link

The link instance quantifies the capabilities of the physical link, which the output port adheres to. The modeled capabilities are: (a) delay (ns), (b) bandwidth (bit/ns), and (c) drop rate (probability of a packet not arriving, e.g. through interference). In all experiments, links are used with a bandwidth of 10 bit/ns (10 Gbit/s) and a delay of 20ns without any chance of dropping a packet due to link failure. A bi-directional cable of a data center is modeled as two links with respective output port in opposite directions.

Output port

The network simulation only models output ports of network devices. An output port receives a packet from the network device and does its best effort according to its protocol and restrictions to transmit it to the *target network device*. A single implementation is used throughout the experiments: an *ECN tail drop output port*. The tail drop queue accepts packet into its FIFO queue until the maximum buffer queue size is reached, at which point it simply drops packets. It marks packets with the ECN flag that arrive when the current buffer queue size exceeds the ECN threshold. Pseudo-code of the algorithm it uses is described in appendix D.1.

Network device

The network device resembles the concept of a node in a graph. It always has a collection of output ports. A network device with a transport layer attached to it is a server, whereas a network device without is only a switch. A network device directly receives incoming packets from the output ports of which it is the target network device. When receiving 'foreign' packets, it decides based on the destination which of its output ports should queue it, or if it should pass it on to the underlying transport layer typically if that the packet's destination. It can also receive packets from the underlying transport layer, which it can either directly forward or for example encapsulate to encode routing-specific state.

There are four base network devices that are used in tests and the experiments:

- *Forwarder-Switch:* received packets are forwarded to a single next-hop neighbor;
- *ECMP-Switch:* a hash based on (source, destination, flow_id, flowlet_id) is calculated that decides to which of the next-hop neighbors it should forward the packet;
- *Valiant-Switch:* upon reception of a packet from the transport layer, a valiant encapsulation is created. A hash based on (source, destination, flow_id, flowlet_id) is calculated, which determines to which of the available valiant nodes the encapsulation is sent. Routing of the encapsulation is done the same as for an ECMP-switch. Once an encapsulation has passed the valiant node, it is removed and is directed to its actual destination.
- *Source-Routing-Switch:* upon reception of a packet from the transport layer, a source routing encapsulation is created. A hash based on (source, destination, flow_id, flowlet_id) is calculated, which determines which of the k paths to the destination are taken. The chosen path is

stored in the encapsulation, and is forwarded by the switches exactly as the path describes.

Besides these four base network devices, specific variants of the network devices have been created to test out slightly different tactics.

- *Random-Valiant-Switch:* it is possible to specify a range of network device indices out of which to choose the valiant node.
- *ECMP-then-Valiant-Hybrid-Switch:* until threshold Q (typically set to $100KB$) is reached, the flow is sent over direct ECMP shortest paths. Afterwards, the flow is sent over a valiant load balancing path;

Flowlet intermediary

Besides the default routing logic of network devices, there is a *flowlet intermediaries* for each, which does a pass over every packet that goes into and comes out of the transport layer. The conceptual idea of a flowlet is explained in section 4.1.4.

There are three flowlet intermediaries:

- *Identity:* flowlet identifier is never modified and stays at 0;
- *Uniform:* once a flowlet gap is detected, it increases the flowlet identifier by one and thus the next flowlet will take a different path if the switch logic incorporates the flowlet identifier in its routing decision;
- *DCTCP-detect-uniform:* only changes the flowlet identifier of a flow if a flowlet gap is detected and a draw from a Bernoulli-distribution with $p = DCTCP\text{-}\alpha\text{-}fraction$ comes out positive. This might be useful following for two reasons: (a) it would be adverse to change the path of the next flowlet when there is no congestion, and (b) in e.g. a scenario of two conflicting flows in which only one flow should move, it lessens the probability of both flows changing path.

Transport layer

The transport layer maintains the sockets for each of the flows that are started at the network device and for which it is the destination. There are two possible transport layers: TCP (see section 4.1.1) and DCTCP (see section 4.1.2).

4.2.2 Routing

The routing initializes the routing state stored on network devices. The network devices use their internal logic and routing state to determine to which output port (or to the underlying transport layer) it should send a

packet. Network devices are based on abstract classes that certain routing initialization schemes require, e.g. ECMP-Switch is the base class needed for ECMP routing initialization.

The possible routing initialization strategies are as follows:

- *Single-forward:* first calculates all shortest path lengths using Floyd-Warshall, and then for every (src, dst)-pair selects, of all possible outgoing edges of *src* on a shortest path towards the *dst*, the next-hop to the node with the lowest node index (as a heuristic);
- *ECMP:* first calculates all shortest path lengths using Floyd-Warshall, and then for every (src, dst)-pair saves all the outgoing edges of *src* on a shortest path towards the *dst* as a set of possible next-hops;
- *K-paths:* for every (src, dst)-pair it reads from file (up to) K arbitrary paths between the two and saves them as routing state;
- *K-shortest-paths:* performs Yen's *K*-shortest-paths algorithm to find the *K* paths for every (src, dst)-pair and saves them as routing state. It also saves them in a file such that it can be used via *K*-paths next time.

4.2.3 Traffic

The traffic of the simulator is planned using flow start events. A flow start event has four variables: (a) a time, (b) a source, (c) a destination, and (d) the flow size. A Poisson traffic schedule is used to generate these flow start events before the run of the simulation. Three components are needed to generate a Poisson traffic schedule for a certain experiment duration: (a) λ, the number of flow starts per second; (b) $p_{flow\,size}$, the flow size distribution; and (c) p_{pair}, a communication pair probability distribution. This strategy of flow planning is employed as well by [5] and [10]. The procedure is described in algorithm 2. The choice for the components are explained in the following sections.

Algorithm 2 Generating Poisson traffic schedule

Require: $\lambda\ (starts/s),\ p_{flow\,size},\ p_{pair},\ duration\ (ns)$
1: $time = 0$
2: **while** $time \leq duration$ **do**
3: Indep. draw $pair$ from p_{pair}
4: Indep. draw $flowsize$ from $p_{flow\,size}$
5: Create flow start event $(time,\ pair,\ flowsize)$
6: Indep. draw $intertime$ from λ-Poisson distribution
7: $time = time + intertime$
8: **end while**

Flow start frequency λ

The start frequency determines how many flows are started per time unit. Once the flow size and pair distribution are fixed, it is used to adjust the corresponding load on the network. Skewness of the flow size distribution and (server) pair distribution especially have impact what value of λ creates bottlenecks in the network. [10] indicates that the maximum load should be when the busiest link has an average utilization of 80%, at which [10] indicates 5% of its flows do not finish in its custom simulator.

Flow size distribution $p_{flow\,size}$

The goal of the flow size distribution is to emulate the actual sizes of flows occurring in the target emulated network. Each flow start event independently draws its flow size from this distribution. The range of possible flow size distributions of the experiments is discussed in section 4.3.

Pair distribution p_{pair}

The goal of the pair distribution is to emulate the skewness of communicating nodes or servers in the target emulated network. Each flow start event independently draws its (src, dst)-pair from this distribution. The range of possible flow size distributions of the experiments is discussed in section 4.3.