We now consider a model depicting a cloud-computing architecture shown in Fig 1. There is a solid, directed edge from component type i to component type j if $j \in \Gamma_i$, and the label on the edge is $\phi_{i,j}$. The dashed, undirected edge shows the connection of our system to the public internet. Our system is designed to handle hypertext transfer protocol (HTTP) requests and simple mail transfer protocol (SMTP) requests. NS represents a network switch; $LB1$ and $LB2$ are two types of load balancers; $FW1$ and $FW2$ denote two types of firewalls; $HV1$ and $HV2$ signify two types of hypervisors; and $SR1$ and $SR2$ portray two types of server racks. This organization of components specifically shows a firewall "sandwich" as described in [1].

We set $r_{NS} = r_{LB1} = r_{LB2} = r_{FW1} = r_{FW2} = r_{HV1} = r_{HV2} = 2$ and $r_{SR1} = r_{SR2} = 3$. Let υ_i define the number of components of type i that need to be up for the system to remain operational. We need at least two server racks of each type $SR1$ and $SR2$ to be up to be able to parallel process across server racks. For other component types, we require at least one component of each type of the system to be up for the system to remain operational. Thus, $\upsilon_{NS} = \upsilon_{LB1} = \upsilon_{LB2} = \upsilon_{FW1} = \upsilon_{FW2} = \upsilon_{HV1} = \upsilon_{HV2} = 1$ and $\upsilon_{SR1} = \upsilon_{SR2} = 2$. Additionally, our system operates in two environments: high demand ($e = 0$) and low demand ($e = 1$). This gives us a state space of size 69984. Solving this using the previous version of DECaF takes xxx seconds, whereas using the new version's tree-generation algorithms given in Section **??** takes xxx seconds.

We build the system such that all component types that are located to the left of the network switch (i.e., types with a "1" in their names) handle SMTP requests only, whereas all component types located to the right of the network switch (i.e., types with a "2" in their names) handle HTTP requests only. Thus, both the branches need to be operational for the system to be able to handle HTTP and SMTP requests. This type of decoupling ensures that hardware failure on one side does not immediately propagate to the other. In this model the system-up condition is an AND of the individual-up conditions. Note that both the previous and new versions of DECaF allow for a general boolean combination of inequalities.

We assume that failure of any component causes a downward-propagating cascading failure because components on levels below the failed component end up going offline and reach a dormant state. The propagation cannot continue via firewalls onto server racks because although the failure of a firewall renders server racks vulnerable, they are still "online". As stated in [2], "hypervisors almost always cause other system components to fail and

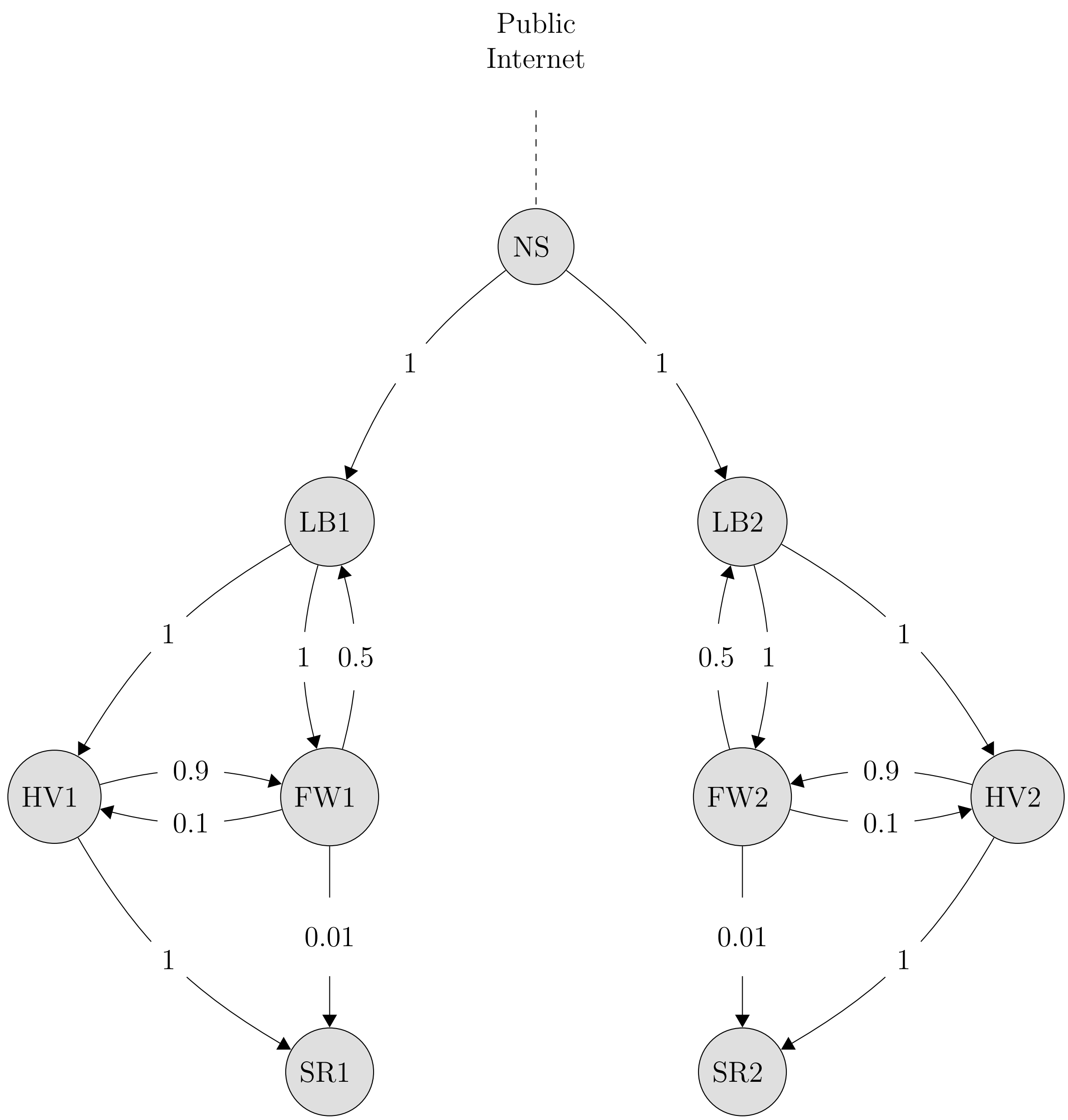

Figure 1: Cloud-computing architechture

certainly cause server racks to fail because of state corruption." Thus, we assume there is probability 0.9 that a hypervisor causes a firewall to fail. A firewall that solely implements an access-control-list verification is one example of a vulnerable firewall. Such a fierewall may let malicious packets get through and cause a failure of a load balancer, which we assume occurs with probability 0.5. We model the probability of this to be 0.5 assuming half the packets received are malicious. The failure of a firewall can cause a hypervisor to fail due to the failed firewall sending malformed packets. We assume the probability of this is 0.1. A firewall in rare cases (which we model by setting $\phi_{FW1,SR1} = \phi_{FW2,SR2} = 0.01$) can expose server racks to attack, though most likely server racks become unusable because of a hypervisor failure.

To estimate the component failure rates, we assume that in a high-demand environment a component is going to fail on average in twice the time of its warranty and in a low-demand environment a component fails in about ten times the period of its warranty. Our time unit is hours. Typical commercial network switches, load balancers and hypervisors have warranties of 90 days (2160 hours). Thus, $\lambda_{NS,0} = \lambda_{LB1,0} = \lambda_{LB2,0} = \lambda_{HV1,0} = \lambda_{HV2,0} = 1/4320$ and $\lambda_{NS,1} = \lambda_{LB1,1} = \lambda_{LB2,1} = \lambda_{HV1,1} = \lambda_{HV2,1} = 1/21600$. Since firewalls almost never fail in isolation, we set $\lambda_{FW1,0} = \lambda_{FW2,0} = \lambda_{FW2,1} = \lambda_{FW1,1} = 1 \times 10^{-10}$. Commercial server racks often have 3-year warranties giving us $\lambda_{SR1,0} = \lambda_{SR2,0} = 1/52560$ and $\lambda_{SR1,1} = \lambda_{SR2,1} = 1/262800$.

For the components' repair rates, we assume that all failed hardware components are swapped out with a repair rate of 1 regardless of the environment, i.e., we can replace one component an hour on average. For simplicity we asume this rate is 1 as well. Firewalls being non-hardware components need to be reconfigured after they fail and this occurs with a repair rate of 0.1. Our system switches between environments once every 12 hours giving us the following environment transition rates $\nu_{0,1} = 1/12$ and $\nu_{1,0} = 1/12$.

References

[1] Salchow Ken Jr. Load balancing 101: Firewall sandwiches. White Paper 888-882-4447, F5 Networks, Inc, 401 Elliott Avenue West, Seattle, WA 98119, 2010.

[2] Ye Michael and Tamir Yuval. Rehype: Enabling vm survival across hypervisor failures. page 1, 2011.