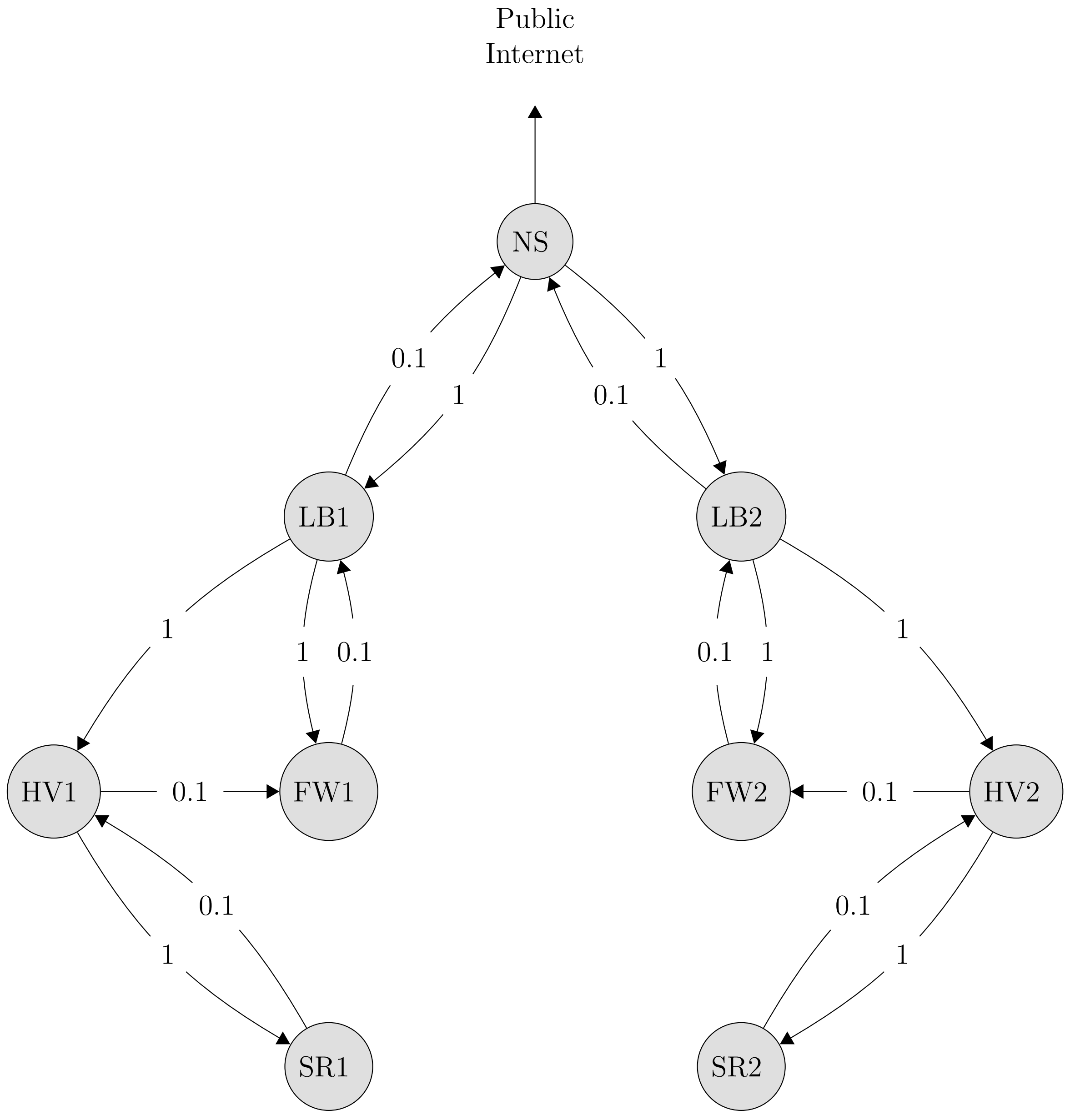
Public
Internet
NS
LB1
LB2
HV1
FW1
FW2
HV2
SR1
SR2
0.1
1
0.1
1
1
1
0.1
0.1
1
1
0.1
0.1
0.1
1
0.1
1

We now consider a model representing a cloud computing architecture. Our system is designed to handle Hyper Text Transfer Protocol (HTTP) requests and Simple Mail Transfer Protocol (SMTP) requests. NS represents a Network Switch; LB1, 2 represent two types of Load Balancers; FW 1, 2 represent two types of firewalls; HV 1, 2 represent two types hypervisors and SR 1, 2 represent 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 $r_{SR1} = r_{SR2} = 3$. We need at least two server racks of both types SR 1, 2 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 up in order 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 and low demand. This gives us a state space of size 69984. Traditionally solving a model of this size using DECAF1 would take a very long time whereas using DECAF2's tree generation algorithms given in section **??** it takes xxxx seconds.

We build the system such that all component types that are located to the left of network switch (i.e. types with a 1 in their name) handle SMTP requests only whereas all component types located to the right of the network switch (i.e types with a 2 in their name) handle HTTP requests only. Thus both the branches need to be operational for the system to be able to handle HTTP and SMTP requests.

We assume that failure of any component causes a certain, downward propagating, cascading failure because components have no option but to go offline and reach a dormant state. To comply with our model dormancy is treated as a failure.

References

[1] Jr. Salchow Ken. Load balancing 101: Firewall sandwiches. Technical report, 2010.