RED Protocol Simulation

Ryan Gambord

January 29, 2025

In this programming assignment, your group will implement a tick-based simulation to investigate the protocols we have discussed in class: UDP, TCP, and RED. You may use whichever programming language you are most comfortable with, but I strongly suggest that you use Python since packages like numpy can make parts of this project much easier.

1 Overview

At a high level, the network you will implement consists of Nodes connected to each other via Links. Each Node may either be a Router or a Host, with each Host connected to exactly one Router, and Routers connected to each other and to other Hosts. Hosts generate traffic and send it to other Hosts in the network. Traffic is modeled as a flow of Packets.

Packets are the basic unit of traffic flow in this simulation, and are assumed to all have the same size and equivalent to a TCP segment (=1 MSS). Each packet has the following header fields: source, destination, and packet type, which may be UDP, TCP. TCP packets additionally have a sequence number and an ACK bit.

Links model a connection between two Nodes in the network, each with a fixed propagation delay and capacity of one packet per tick. A packet that is injected onto a link appears at the other side after the propagation delay + 1 (transmission delay). Links are bi-directional and symmetric.

2 Traffic Generation

Hosts generate the packets that flow through the network. Hosts alternate between generating a packet every tick (ON, Active state) and not (OFF, Silence state). The duration of each state is determined by sampling a Pareto distribution, which is commonly used for modeling realistic network traffic,

$$P(T > t) = \begin{cases} \left(\frac{x_{\min}}{t}\right)^{\alpha} & t \geq x_{\min} \\ 1 & t < x_{\min} \end{cases}$$

Where,

- t is the duration of the ON or OFF period.
- x_{min} is the *scale parameter* which controls the minimum duration. We use a value of 1 in this project.
- α is the *shape parameter* which controls how bursty the traffic is. Larger values of α produce less bursty traffic patterns. We have two α parameters, α_{ON} and α_{OFF} for controlling the on and off times independently.

If you aren't familiar with the Pareto distribution, the CDF is $F(t) = 1 - \left(\frac{x_{\text{min}}}{t}\right)^{\alpha}$, and its inversion is $T = x_{\text{min}}(1-U)^{-\frac{1}{\alpha}}$, so you can sample it, for example, in C as,

```
double
pareto(double x_min, double alpha)
{
  double u = (double)rand() / RAND_MAX;
  return x_min * pow(1-u, -1.0 / alpha);
}
```

In python, you can sample the Pareto distribution using numpy,

```
import numpy as np

t = (np.random.pareto(alpha) + 1) * x_min
```

When a host enters the ON (packet generating) state, it randomly selects another host in the network for the generated packets to be addressed to. It also decides whether the packets will be UDP packets or TCP packets. All packets generated in the same "burst" are addressed to the same Host and are of the same packet type. Once a packet is generated, it is placed in the appropriate queue.

Each host models an already active TCP connection with every other host in the network, with a separate queue and congestion window for each. When TCP packets are generated, they are placed in the appropriate TCP queue to be sent out. UDP packets are all placed into a shared queue. These queues are modeled as infinitely large.

Each tick, if its outgoing link is idle, each Host examines its queues to see which are able to send a packet. UDP packets are always ready to be sent. TCP packets can be sent if they fall within the congestion window. The host randomly selects a queue with a ready packet and injects it into the link. For UDP packets, the host then discards the packet from the queue; for TCP packets, recall that the host must keep a copy of the packet until it receives an ACK for it.

3 TCP Protocol

You will implement basic TCP with AIMD. Packets are able to be sent if they fall within a congestion window (CWND) which spans from the oldest un-ack'ed packet.

3.1 Additive Increase

TCP keeps track of how many ACKs have been received. When CWND packets have been ACK'd, the CWND is increased by 1, and the count is reset. This increases the CWND every RTT.

3.2 Multiplicative Decrease

When three duplicate ACKs are received, TCP halves the CWND size and marks every outstanding packet for retransmission.

3.3 Retransmission Timeout Timer

TCP maintains a RTO timer for detecting loss events. This timer's starting value is based on two statistics that are maintained. Each time an ACK is received for an outstanding packet that has not been retransmitted, the host will update its EstimatedRTT for that connection using an exponential weighted moving average with a fixed smoothing factor of 0.125,

$$\texttt{EstimatedRTT} = (1-\alpha)\times\texttt{EstimatedRTT} + \alpha\times\texttt{SampleRTT},\ \alpha = 0.125$$

It will then update its DevRTT using the following formula,

$$\texttt{DevRTT} = (1-\beta)\times\texttt{DevRTT} + \beta\times|\texttt{SampleRTT} - \texttt{EstimatedRTT}|,\ \beta = 0.25$$

When a packet is transmitted, if the timeout timer is not currently running, the RTO value is updated using $\texttt{RTO} = \texttt{EstimatedRTT} + 4\times\texttt{DevRTT}$, and the timer is started. When an ACK for the *oldest* unacknowledged packet arrives, and there are more outstanding packets the RTO is recalculated and the timer is reset, otherwise the timer is stopped. If the timer reaches 0, the oldest unack'ed segment is marked for retransmission, the RTO values is doubled (not recalculated), and the timer is reset. The CWND is set to 1.

3.4 Acknowledgement

When a host receives a TCP packet, it generates and Acknowledgement packet and places it in an ACK queue, which is essentially identical in behavior to the UDP queue. If the packet arrives out of order, send an ACK for the most recent in-order packet. That is, you should be tracking the last seq# received and if the incoming packet's isn't seq#+1, send an ACK for the current seq#.

4 Routers

Routers carry traffic across the network from Host to Host. Each router has a fixed-length queue for every link that is connected to it. When it receives a packet on a link, it places that packet into the appropriate outgoing queue based on its routing table. If the queue is full, the packet is dropped.

4.1 RED

Implement the RED algorithm as described in class and the paper reading. Packets are to be marked by dropping them (i.e. no ECN bit). Implement both method 1 (geometric distribution) and method 2 (uniform distribution) for selecting packets to drop.

5 Running the simulation

The parameters of the simulation are,

- Number of routers
- Number of hosts
- The host traffic distribution parameters α_{ON} and α_{OFF}.
- Router buffer size parameter
- RED parameters w_q, min_{th}, max_{th}, and max_p.
- A propagation delay scaling factor, λ_p, described below.

At the start of the simulation, each host and router is assigned a (uniform) random pair of (x,y) coordinates each in the range 0 to λ_p, as if placed on a two-dimensional grid. Connect each host to its nearest router where the link length is the discretized Euclidian distance ($\left\lceil \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \right\rceil$) between the host and router. Then, use Prim's algorithm to generate a minimum spanning tree of the routers in the network. Use this to generate a routing table for each router.

Warm up the network for a few thousand ticks before collecting data. Collect the following data for another few thousand ticks, and then rerun the simulation a few times with different randomized topologies, under the same testing conditions,

- Proportion of congested (full) queues
- Average queue length
- Number of packets dropped
- Number of packets received

First run the simulation with max_p set to 0 to disable RED. Adjust the host traffic parameters until you start seeing an increase in congestion via the above parameters. Then turn on RED and adjust the parameters until you see a reduction in congestion. Rerun these simulations varying the ratio of routers to hosts (2 ratios), router buffer sizes (2 sizes), and propagation delay scaling factor (2 scales), with RED enabled and disabled. This should result in 16 separate sets of data.

6 What to Turn In

Submit your finished programming project, along with a 2-3 page report describing (high-level documentation of) the work that you did and the results of your simulations. Report your data and any conclusions you can draw from running your simulations.