Project 3

April 30, 2015

Group Members:

* Alexandra Lomotan
* David Delgado
* Matthew Chi

Part A:

The simulation has a simpler simulation compared to the actual behavior you might encounter on real Ethernet. In the simulation, inter-frame spacing is ignored, variability in collision time is ignored, and each collision takes up exactly one time slot. On real Ethernet, these factors need to be taken into consideration. Ignoring inter-frame spacing speeds up the process. Allowing each collision to take up exactly one time slot and ignoring collision time can either speed up or slow down delay time.

Part B:

Part C:

Lambda 20: 8/13 = .615 \* 100 = 61.5% of bandwidth

Lambda 18: 8/12 = .666 \* 100 = 66.6% of bandwidth

Lambda 16: 8/13 = .615 \* 100 = 61.5% of bandwidth

Lambda 14: 8/14 = .571 \* 100 = 57.1% of bandwidth

Lambda 12: 8/13 = .615 \* 100 = 61.5% of bandwidth

Lambda 10: 8/14 = .571 \* 100 = 57.1% of bandwidth

Lambda 8: 8/15 = .533 \* 100 = 53.3% of bandwidth

Lambda 6: 8/15 = .533 \* 100 = 53.3% of bandwidth

Lambda 4: 8/34 = .235 \* 100 = 23.5% of bandwidth

**Code:**

Part A:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#define ITERATIONS 100

#define STATIONS 5

int min(int n, int m) {

if (n < m)

return n;

return m;

}

int max(int n, int m) {

if (n < m)

return m;

return n;

}

int computeBackoff(int n) {

int k = min(n, 10);

k = (int)pow(2, k);

return rand() % k;

}

int done(int \*stations) {

int i;

for (i = 0; i < STATIONS; i++) {

if (!stations[i]) {

return 0;

}

}

return 1;

}

void main() {

srand(time(NULL));

int first[ITERATIONS], second[ITERATIONS], third[ITERATIONS], fourth[ITERATIONS], fifth[ITERATIONS];

int iteration;

for (iteration = 0; iteration < ITERATIONS; iteration++) {

// init data for iteration

int timeSent[STATIONS] = {0};

int nextTimeToSend[STATIONS] = {0};

int collisionCount[STATIONS] = {0};

int T = 0;

// loop for all stations to transmit

while (!done(timeSent)) {

// set sending counter to 0, collision to false

int sending = 0;

int collision = 0;

int sendingIndex = -1;

int i;

// check each station

for (i = 0; i < STATIONS; i++) {

int j;

// check if this is the only station trying to send

for (j = 0; j < STATIONS; j++)

if (nextTimeToSend[j] == T) {

sending++;

sendingIndex = j;

}

// set timeSent if only one trying to send, else set new times to send

if (sending == 1) {

timeSent[sendingIndex] = T;

}

else if (sending > 1) {

for (j = 0; j < STATIONS; j++)

if (nextTimeToSend[j] == T) {

collisionCount[j]++;

nextTimeToSend[j] = T + computeBackoff(collisionCount[j]);

}

}

}

// increment time

T++;

}

int minimum = 9999;

int i;

// calculate delay of first transmitted station

for (i = 0; i < STATIONS; i++) {

minimum = min(timeSent[i], minimum);

}

first[iteration] = minimum;

minimum = 9999;

// calculate delay of second transmitted station

for (i = 0; i < STATIONS; i++) {

if (timeSent[i] > first[iteration]) {

minimum = min(timeSent[i], minimum);

}

}

second[iteration] = minimum;

minimum = 9999;

// calculate delay of third transmitted station

for (i = 0; i < STATIONS; i++) {

if (timeSent[i] > second[iteration]) {

minimum = min(timeSent[i], minimum);

}

}

third[iteration] = minimum;

minimum = 9999;

// calculate delay of fourth transmitted station

for (i = 0; i < STATIONS; i++) {

if (timeSent[i] > third[iteration]) {

minimum = min(timeSent[i], minimum);

}

}

fourth[iteration] = minimum;

int maximum = 0;

// calculate delay of fifth transmitted station

for (i = 0; i < STATIONS; i++) {

maximum = max(timeSent[i], maximum);

}

fifth[iteration] = maximum;

}

int i;

for (i = 0; i < ITERATIONS; i++) {

printf("%2d %2d %2d %2d %2d\n", first[i], second[i], third[i], fourth[i], fifth[i]);

}

// calculate averages

int sumFirst = 0, sumSecond = 0, sumThird = 0, sumFourth = 0, sumFifth = 0;

// int i;

for (i = 0; i < ITERATIONS; i++) {

sumFirst += first[i];

sumSecond += second[i];

sumThird += third[i];

sumFourth += fourth[i];

sumFifth += fifth[i];

}

printf("\nAverage first delay: %d\nAverage second delay: %d\nAverage third delay: %d\nAverage fourth delay: %d\nAverage fifth delay: %d\n",

sumFirst/ITERATIONS, sumSecond/ITERATIONS, sumThird/ITERATIONS, sumFourth/ITERATIONS, sumFifth/ITERATIONS);

}

Part B:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#define ITERATIONS 100

#define MAX\_STATIONS 100

int min(int n, int m) {

if (n < m)

return n;

return m;

}

int max(int n, int m) {

if (n < m)

return m;

return n;

}

int computeBackoff(int n) {

int k = min(n, 10);

k = (int)pow(2, k);

return rand() % k;

}

int done(int \*transmitted, int num\_of\_stations) {

int i;

for (i = 0; i < num\_of\_stations; i++) {

if (!transmitted[i]) {

return 0;

}

}

return 1;

}

void main(int argc, char \*argv[]) {

srand(time(NULL));

int stations;

for (stations = 20; stations < 101; stations += 20) {

int first[ITERATIONS];

int iteration;

for (iteration = 0; iteration < ITERATIONS; iteration++) {

// init data for iteration

int timeSent[MAX\_STATIONS] = {0};

int nextTimeToSend[MAX\_STATIONS] = {0};

int collisionCount[MAX\_STATIONS] = {0};

int T = 0;

// loop for all stations to transmit

while (!done(timeSent, stations)) {

// set sending counter to 0

int sending = 0;

int sendingIndex = -1;

int i;

// check each station

for (i = 0; i < stations; i++) {

int j;

// check if this is the only station trying to send

for (j = 0; j < stations; j++)

if (nextTimeToSend[j] == T) {

sending++;

sendingIndex = j;

}

// set timeSent if only one trying to send, else set new times to send

if (sending == 1) {

timeSent[sendingIndex] = T;

}

else if (sending > 1) {

for (j = 0; j < stations; j++)

if (nextTimeToSend[j] == T) {

collisionCount[j]++;

nextTimeToSend[j] = T + computeBackoff(collisionCount[j]);

}

}

}

// increment time

T++;

}

int minimum = 9999;

int i;

// calculate delay of first transmitted station

for (i = 0; i < stations; i++) {

minimum = min(timeSent[i], minimum);

}

first[iteration] = minimum;

}

// calculate averages

int sumFirst = 0, sumSecond = 0, sumLast = 0;

int i;

for (i = 0; i < ITERATIONS; i++)

sumFirst += first[i];

printf("\n%3d stations average first delay: %3d\n", stations, sumFirst/ITERATIONS);

}

}

Part C:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <math.h>

#define ITERATIONS 50

#define STATIONS 20

int min(int n, int m) {

if (n < m)

return n;

return m;

}

int max(int n, int m) {

if (n < m)

return m;

return n;

}

int computeBackoff(int lambda) {

return (0 - lambda) \* (log((double)rand()/(double)RAND\_MAX) / log(2));

}

int done(int \*transmitted) {

int i;

for (i = 0; i < STATIONS; i++) {

if (!transmitted[i]) {

return 0;

}

}

return 1;

}

void main(int argc, char \*argv[]) {

srand(time(NULL));

int lambda;

for (lambda = 20; lambda > 3; lambda -= 2) {

int contentionInterval[ITERATIONS] = {0};

int iteration;

for (iteration = 0; iteration < ITERATIONS; iteration++) {

// init data for iteration

int timeSent[STATIONS] = {0};

int nextTimeToSend[STATIONS] = {0};

int lastAttemptTime[STATIONS] = {0};

int T = 0;

// loop for all stations to transmit

while (!done(timeSent)) {

// set sending counter to 0, sendingIndex to invalid

int sending = 0;

int sendingIndex = -1;

int i;

// check how many stations want to send in this interval

for (i = 0; i < STATIONS; i++) {

if (nextTimeToSend[i] == T-1 || nextTimeToSend[i] == T) {

sending++;

if (nextTimeToSend[i] == T)

sendingIndex = i;

}

}

// if only one station wants to send in the interval, and it wants to send

// at time T, then send.

if (sending == 1 && sendingIndex != -1) {

if (nextTimeToSend[sendingIndex] == T) {

timeSent[sendingIndex] = T;

}

}

// if more than one station wants to send in the interval, check if something tried

// to send at time T-1. If so, reset it. Compute nextTimeToSend for all colliding stations

else if (sending > 1) {

for (i = 0; i < STATIONS; i++) {

if (timeSent[i] == T-1) {

timeSent[i] = 0;

lastAttemptTime[i] = nextTimeToSend[i];

while (nextTimeToSend[i] <= T)

nextTimeToSend[i] = T + computeBackoff(lambda);

}

if (nextTimeToSend[i] == T-1 || nextTimeToSend[i] == T) {

lastAttemptTime[i] = nextTimeToSend[i];

while (nextTimeToSend[i] <= T)

nextTimeToSend[i] = T + computeBackoff(lambda);

}

}

}

// increment time

T++;

}

// find the max timeSent, divide by number of stations to get contention interval

int maximum = 0;

int i;

for (i = 0; i < STATIONS; i++) {

maximum = max(maximum, timeSent[i]);

}

contentionInterval[iteration] = maximum / STATIONS;

//printf("Contention interval is %3d for lambda %2d iteration %3d\n", contentionInterval[iteration], lambda, iteration);

}

// find minimum contention interval

int minimum = 9999;

int sumContention = 0;

int i;

for (i = 0; i < ITERATIONS; i++) {

minimum = min(minimum, contentionInterval[i]);

sumContention += contentionInterval[i];

}

printf("Lambda %2d minimum contention interval = %d\nAverage contention interval = %d\n\n", lambda, minimum, sumContention/ITERATIONS);

}

}