# CS 670 - HW 1

March 12, 2008

# 1 Exercise 1.3

## 1.1 Implementation of a Chi-Square Test Statistic for IP Packets

The chi-square statistic can be used to find if the overall set of observed character frequencies in a received IP Packet are unusually different (as compared to normal random variation) from the expected character frequencies. This is a more sophisticated test, statistically speaking, than the simple threshold detector used in the warm-up example. Assume that the thresholds represent the expected frequencies. The statistic is computed by

$$\chi^2 = \sum_{i=1}^{n} \frac{\left(f_e\left[i\right] - f_o\left[i\right]\right)^2}{f_e\left[i\right]} \tag{1}$$

where $\mathbf{n}$ is the number of character types, $\mathbf{f_e}$ is the expected frequency of the indexed character type, $\mathbf{f_0}$ is the observed frequency of the indexed character type, and $\boldsymbol{\chi^2}$ is the test statistic.

Design a hardware chip that will alarm if the observed characters in an IP packet results in a Chi-Square test statistic that is above a specified threshold $\mathbf{T}$. Design the chip in a way to efficiently implement the statistic assuming that the length of the packet is known only after processing the entire packet.

## 1.2 Initial Design Thoughts and Assumptions

The overall length of the data portion of the IP packet is straightforward to implement; a register is incremented for each data byte processed. Therefore, we will focus on the problem of processing the test statistic at wire speed after the data is received (during the receipt of the header of the following IP packet). It is unreasonable to believe that a floating point calculation of the test statistic can be calculated in this time. For each character type (256 charcters for the ASCII character set) an observed frequency must be calculated (which requires a division), the square of the difference from the expeceted frequency must be found, and the result must be divided by the expected frequency. The sum of the 256 calculated values must be found and compared to the threshold. This is an enormous amount of processing to do in such a short time period. Therefore the convienient solution will not suffice.

Let's start our initial alalysis by expanding the test statistic equation (1) which results in the following expression for the 256 element ASCII character set

$$\chi^2 = \sum_{i=1}^{256} \frac{f_e[i]^2 - 2f_e[i]\,f_o[i] + f_o[i]^2}{f_e[i]} \tag{2}$$

which can be simplified as

$$\chi^2 = \sum_{i=1}^{256} \left( f_e[i] - 2f_o[i] + \frac{f_o[i]^2}{f_e[i]} \right). \tag{3}$$

Noting that $f_o[i] = \frac{C[i]}{L}$ where $\mathbf{C}[\mathbf{i}]$ is the indexed character occurance count within the IP packet and $\mathbf{L}$ is the length of the IP packet in characters (bytes in the ASCII case), we can express equation (3) as

$$\chi^2 = \sum_{i=1}^{256} \left( f_e[i] - 2\frac{C[i]}{L} + \frac{C[i]^2}{L^2 f_e[i]} \right). \tag{4}$$

Distributing the summation throughout the expression, equation (4) can be written as follows

$$\chi^2 = \sum_{i=1}^{256} f_e[i] - \frac{2}{L}\sum_{i=1}^{256} C[i] + \sum_{i=1}^{256} \frac{C[i]^2}{L^2 f_e[i]}. \tag{5}$$

Equation (5) can be greatly simplified if one notes that $\sum f_e = 1$. This is a result of the obvious; we expect to get a character each time we receive a character. This design "requirement" along with the fact that $\sum C[i] = L(for i = 1 \ldots 256)$ allows equation (5) to be further simplified into the following expression

$$\chi^2 = 1 - 2 + \frac{1}{L^2}\sum_{i=1}^{256} \frac{C[i]^2}{f_e[i]}. \tag{6}$$

Using this form of the test statistic, we can state that the threshold alarm should be fired if

$$-1 + \frac{1}{L^2}\sum_{i=1}^{256} \frac{C[i]^2}{f_e[i]} > T \tag{7}$$

or

$$\frac{1}{L^2}\sum_{i=1}^{256} \frac{C[i]^2}{f_e[i]} > T + 1 \tag{8}$$

Letting $T_f = T + 1$ results in the final inequality

$$\frac{1}{L^2}\sum_{i=1}^{256}\frac{C\left[i\right]^2}{f_e\left[i\right]} > T_f \tag{9}$$

Using equation (9) for the test statistic allows the value $T_f$ to be stored in the threshold register for alarm detection; and in addition, it allows the final division by $L^2$ to be delayed until all the data has been received. ***Note:*** *This is the only use of the packet length L.* In spite of its advantages over equation (1), it still requires 256 multiplications and 256 divisions to achieve the value for the sum in equation (8). As a result, the summation must be simplified or calculated as the data is received in order to efficiently evaualte the test statistic at line speed.

One design decision is to require all expected frequencies to take on a value of the form $\frac{1}{2^x}$. If this design decision still satisfies user requirements, it can simplify the computations required in equation (9). The use of this simplification allows equation (9) to be expressed as follows

$$\frac{1}{L^2}\sum_{i=1}^{256}2^{x[i]}C\left[i\right]^2 > T_f. \tag{10}$$

By replacing the division operation with a multiplication of a power of two, we can replace the 256 division operations with 256 shift operations. This speeds up the statistic evaluation dramitacally, but still leaves 256 multiplications and 256 additions in order to generate the sum of the square of the individual character counts. In spite of the success in simplifying the expression, we must now try to find a way to calculate the summation in inequality (10) as characters arrive in order to have a chance at line speed processing for the desired test statistic.

To try and find an incremental approach to build the sum in inequality (10) we must first look at how the value of the summation changes when an individual character count is incremented. When a character count is incremented the summation increases in value according to the following expression

$$\Delta_\Sigma = 2^{x[i]}\left(C\left[i\right]+1\right)^2 - 2^{x[i]}C\left[i\right]^2. \tag{11}$$

Expanding the expression gives

$$\Delta_\Sigma = 2^{x[i]}\left(C\left[i\right]^2 + 2C\left[i\right]+1\right) - 2^{x[i]}C\left[i\right]^2, \tag{12}$$

and collecting terms yields

$$\Delta_\Sigma = 2^{x[i]+1} C\,[i] + 2^{x[i]}. \tag{13}$$

Equation (13) is an incremental expression that will allow the value of the summation in equation (10) to be calculated as incoming packet characters are being counted and processed. Equation (13) is a very simple and fast expression which can be evaluated by shifting the previous character count according to the character's stored expected frequency (since it is a multiplication of a power of two). We simply need a hardware register to store the incremental result until all characters have been received for the packet in order to use it for the final statistic calculation.

Since the summation in equation (10) is calculated as characters arrive, we only need to divide the result by $L^2$ to calculate the statistic. This should be workable in the available time. The result can be compared to the threshold $T_f$ in order to determine of an alarm should be raised. It is noteworthy that by incrementaly calculating the summation of equation (10) we reduce post data calculations to one division operation, which is a substantial improvement from the original calculation requirements.

### 1.3 Final Design Thoughts

The memory footprint for the above design would consist of an array of 256 32-bit words having the following data boundaries

- 3-bit generation number
- 13-bit expected frequency $f_e$ shift value $\mathbf{x}$, where $f_e = \frac{1}{2^x}$
- 16-bit character counter

The generation number would be used in the same fashion as the previous threshold example to help elliminate initialization. The expected frequency would hold the shift value associated with the indexed character. The character count would hold the current occurance count for the indexed character for the packet being processed. In addition to the array, a few other registers would be needed in order to implement the algorithm

- Current generator number
- Current summation value

- Current packet character count
- Alarm threshold value

This results in an estimted footprint of 260 32-bit words which is quite small for the posed problem.

### 1.4 Pseudo Code

```
Length = 0
GenNumber = 0
Sum = 0
Threshold = Tf
NUM_CHAR_TYPES = 256

for(uint i=0; i<NUM_CHAR_TYPES; i++)
    charRecord[i].ExpectedFreqShift = DESIRED_VALUE

while (RunProcess)
{
    if ( (GenNum mod 8) == 0)
    {
        for(uint i=0; i<NUM_CHAR_TYPES; i++)
        {
            charRecord[i].Count = 0
            charRecord[i].GenNumber = 0
        }

        GenNum = 0
    }

    Sum = 0
    Length = 0

    while(DataToBeProcessedInCurrentPacket())
    {
        character = ReadCharacter()

        if(character)
        {
```

```
            Length += 1

            if(charRecord[character].GenNum != GenNum)
            {
                charRecord[character].GenNum = GenNum
                charRecord[character].Count = 1

                Sum += (1 << charRecord[character].ExpectedFreqShift)
            }
            else
            {
                Sum += (charRecord[character].Count <<
 charRecord[character].ExpectedFreqShift)
                        + (1 << charRecord[character].ExpectedFreqShift)

                charRecord[character].Count += 1
            }
        }
    }

    if(Sum/(Length*Length) > Threshold)
        Alarm()
}
```