A Comparison of Round-Trip Time Estimation Algorithms

Stephanie Lukin
Loyola University Maryland

Advisor: Katia Obraczka
Gradute Students: Kerry Veenstra, Bruno Nunes
Inter-Networking Research Group
University of California, Santa Cruz

Stephanie Lukin's work was supported by the UCSC SURF-IT 2010 Research Experiences for Undergraduates Site, NSF grant CNS-0852099, <surf-it.soe.ucsc.edu>

I. Introduction

The time it takes for a packet to traverse the internet is a difficult value to calculate. Based upon the Round-Trip Time (RTT), a timer is set. It is initiated when the packet is sent, and is terminated when the sender receives an acknowledgement from the recipient indicating the packet's completed delivery. We can reduce the amount of time wasted during internet communication by closely estimating this travel time. The original estimation algorithm is named after its developer Jacobson, and has been implemented since the birth of online data transfer.

Since then, researchers have been developing their own algorithms to improve this estimation technique. The Inter-Networking Research Group (i-NRG) at the University of California, Santa Cruz, developed the Experts Algorithm and has shown in preliminary trials to be about 40% more accurate than Jacobson's algorithm. As further research, the i-NRG team wanted to compare their algorithm to another in existence named Eifel and developed by a team at the University of California, Berkeley and Ericsson Research. In order to accurately test each algorithm, the i-NRG lab created a module containing the logic for computing Jacobson, Experts, and Eifel RTT values. This module is integrated into the Linux kernel of one of the nodes in our network testbed. Each estimation algorithm calculates its own RTT value and then one value is used in the kernel during runtime. With our module, we plan to run large and "famous" tests over the network to identify the most efficient algorithm and strive towards improving calculation techniques.

II. Background

Transmission Control Protocol / Internet Protocol (TCP/IP) is a five layer protocol suite. It defines the exchange of transmissions across the internet, with the most important protocols being TCP and IP. The protocol most closely associated with this project is TCP. This is the transport layer in the protocol suite and deals with a higher level of data transportation, rather than the bits of data. In this layer, the entire packet of information is taken into consideration. Based upon the size of this packet, a Round-Trip Time (RTT) is estimated. This is the time required for data to go from a source to a destination and then receive acknowledgement of the arrival.

The RTT has many applications to TCP. One of the most important is the Retransmission Time-Out (RTO) timer. This timer is maintained with each connection, and if the sender does not receive acknowledgement upon the expiration of the timer, TCP will retransmit the data and restart the timer. This timer is based upon the size of the packet. Because each packet may be a different size, the timer cannot be a fixed value and therefore relies on an accurate estimation of the RTT.

Another application of RTT is calculating the number of retransmissions of a data packet, which ideally is kept to zero ensuring unbroken communication. The time also helps to measure the throughput and goodput, which are the number of transmissions sent over the network and the number of positive transmissions acknowledged respectively. Thus, it is extremely important that the RTT be calculated precisely to or as close to the true value as possible.

III. Motivation

The Inter-Networking Research Group has set out to develop their own RTT estimation algorithm which uses an on-line machine learning approach. In this way, the RTT would better adapt to the fluctuations observed over an active network. In order to fully validate this algorithm, the team needed to compare it to preexisting RTT estimation algorithms. The first is Jacobson's which is the estimation algorithm currently implemented in TCP and has been since TCP first began in 1988. The other algorithm is called Eifel and was developed by Reiner Ludwig and Keith Sklower in 2000 [2]. Our study conducts a comparison of each estimation algorithm in a live environment, examining not only the closeness of the RTT value to the measured value over the network, but also the repercussions of a closely or poorly approximated time.

IV. Implementation

In order to test each estimation algorithm with the highest amount of precision, a module was constructed. This module contained the logic for how each estimation algorithm calculates the RTT value. There are two methods for conducting tests. In the first, the module is run as a standalone program. The data is input either via a text file or through fixed values hardcoded by the tester. In the experiments with the text file, measured RTT's (mRTT) are collected from previous experiments and feed into the standalone program as input. The algorithms then attempt to predict the next value in the log file based on the previous measurement. This method is used to validate the correctness of the coded Eifel algorithm.

The other testing method is a live test. In this test, the module is synchronized with the Linux kernel of one of the nodes in the i-NRG network testbed. The module is dynamically inserted into the kernel at runtime where the logic of how to calculate RTT values is redirected into the loaded module. When the tests are complete, the module is removed (*figure 1*).

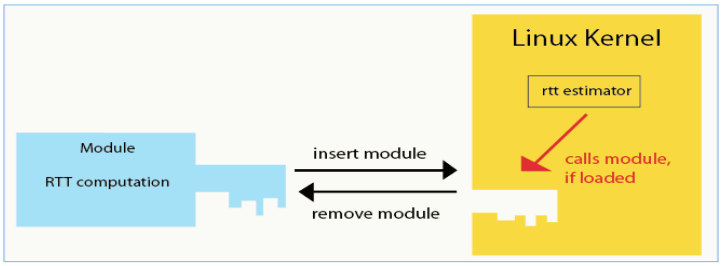

Figure 1: Linux kernel implementation

Every time the module is loaded into the kernel, a test set of three is computed- one test for each algorithm. In order to ensure that the tests have as little variance as possible, the tests are run consecutively. In each test, all three algorithms are computed but only one is used in the computation of the data transportation over the network (*figure 2*).

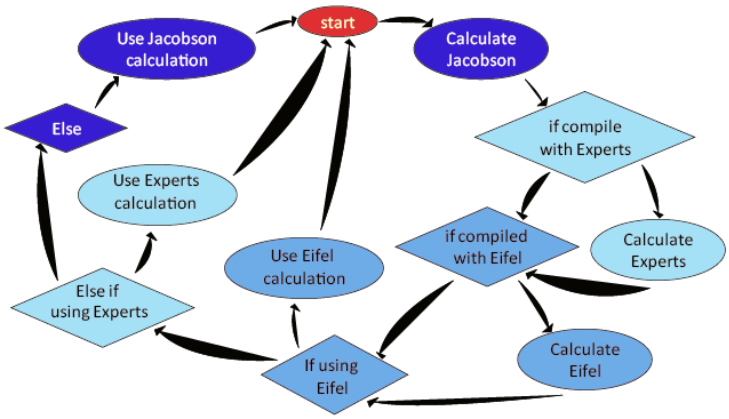

Figure 2: RTT calculation

The integration of the module into the kernel proved to be intricate. Methods, data types, and variables were added into the kernel code so it had knowledge of their use in the module logic. A careful and precise importation and exportation took place for each new addition. Once the kernel had knowledge of these additions, the kernel was rebuilt in order to incorporate these changes. Once rebuilt, the node was rebooted with its new logic in place. At this point, we could easily insert and remove the module in order to perform our aforementioned live tests.

Coding the algorithms as a standalone program was as straightforward as following the computations specified by the formulas (*figures 5 – 7*). At the start of this summer, the logic for Jacobson and Experts already existed in the module. To add the Eifel computation, it was first implemented in the standalone program. Testing was done to ensure that the Eifel program would perform in a predicted manner. After completion of the standalone program, the algorithm was imported into the preexisting module. In order to integrate the standalone program with the Linux kernel, some changes in the storage and representation of data in the standalone program needed to be made. In the kernel space, the only numerical data type allowed is an Integer. An Integer consists of 32 bits, where each bit represents the numerical part of the number. It is important to recognize that Integers do not store or keep track of the fractional part of a number. The following section will address how each algorithm dealt with this integration problem.

V. Three Algorithms

A. Jacobson's Algorithm

The calculation involves a simple four-step process requiring only two input values and fixed weights to calculate the RTT and RTO (displayed in *figure 5*). Simple arithmetic is involved: addition, subtraction, and multiplication.

1. error = measured RTT − prediction
2. new prediction = old prediction + $^1/_8$ × error
   = $^7/_8$ × old prediction + $^1/_8$ × measured RTT
3. new variation = $^3/_4$ × old variation + $^1/_4$ × abs(error)
4. RTO = prediction + 4 × variation

Figure 5: Jacobson's RTT Estimation Algorithm

In the Linux kernel's implementation of RTT, which is Jacobson's estimation algorithm, the numbers are stored such that the left most 29 bits represent the numerical part of the number and the 3 right most bits the fractional part, (*figure 3*). Because the arithmetic in Jacobson's algorithm is simple, it uses this 29.3 bit representation without having to worry about any additional bit shifting or scaling, except for one: the initial measured RTT (mRTT) value is to be shifted to the left by 3 bits (<< 3), i.e. multiplied by 2^3.

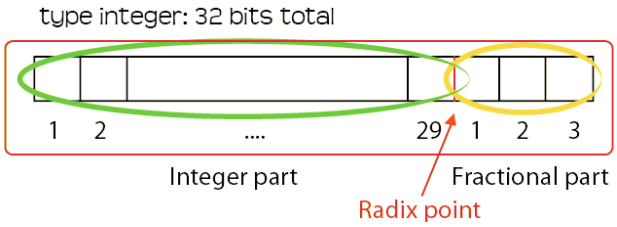

Figure 3: Integer storage

B. Experts Framework

The Experts Framework was developed by the i-NRG lab over the previous year [1]. The process follows a similar overall structure to Jacobson's but differs in steps 1 and 2 (*figure 6*). Some of the modified computations were so extensive that an entire mathematics library was created to accommodate the weighted averages and exponential operations. Although the computations are more complex than Jacobson's algorithm, no further bit shifting is required except for the initial mRTT value.

1. for each expert:
   error = measured RTT − prediction
2. for each expert:
   preliminary weight = old weight × exp(-η × error²)
3. compute average preliminary weight
4. for each expert:
   new weight = (1 − α) × preliminary weight
   + α × average preliminary weight
5. new prediction = weighted average of experts' predictions
6. new variation = 3/4 × old variation + 1/4 × abs(error)
7. RTO = prediction + 4 × variation

Figure 6: Experts Framework RTT Estimation Algorithm

C. Eifel Algorithm and Fixed Point Arithmetic

The Eifel algorithm was devised by two researchers from California and Germany [2]. There are many computational differences in this algorithm from Jacobson's algorithm (*figure 7*).

1. error $DELTA_E = RTT_{Sample} - SRTT_E$
2. flight $FLIGHT_E = \mathrm{MAX}\left(SSTHRESH, \frac{CWND}{2}\right) + 1$
3. gain $GAIN_E = \begin{cases} \frac{1}{FLIGHT_E}, & \text{if RTT Sampling Rate} = 1 \\ \frac{2}{FLIGHT_E}, & \text{if RTT Sampling Rate} = \frac{1}{2} \\ \frac{1}{3}, & \text{if 1 RTT Sample per RTT} \end{cases}$
4. inverse gain $\overline{GAIN_E} = \begin{cases} GAIN_E, & \text{if } (DELTA_E - RTTVAR_E) \geq 0 \\ GAIN_E^2, & \text{if } (DELTA_E - RTTVAR_E) < 0 \end{cases}$
5. new prediction $SRTT_E = SRTT_E + GAIN_E \times DELTA_E$
6. new variation $RTTVAR_E = \begin{cases} RTTVAR_E + \overline{GAIN_E} \times (DELTA_E - RTTVAR_E), & \text{if } DELTA_E \geq 0 \\ RTTVAR_E, & \text{if } DELTA_E < 0 \end{cases}$
7. RTO $RTO_E = \mathrm{MAX}\left(\left(SRTT_E + \frac{1}{GAIN_E} \times RTTVAR_E\right), RTT_{Sample} + (2 \times tickz)\right)$

Figure 7: Eifel RTT Estimation Algorithm

The most important difference is the presence of division, and especially division by 3. The number 1/3 cannot be represented in binary by a finite series of bits. In order to preserve as many of the numbers to the right of the radix point as possible, we need to move the radix point itself and store the numbers in a different way, called a shifted_t. A shifted_t is a representation of a number such that the first 16 bits are the numerical part and the 16 bits to the right are the fractional part. By assigning more bits to the fraction, we ensure more precision in our calculations.

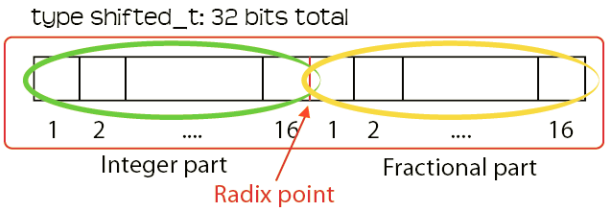

Figure 4: shifted_t storage

VI. Results

When the module and synchronization with the kernel were complete, it was time to begin testing. Two test suites were conducted: a small suite with 17 measurements and a large suite containing 3000 measurements. Three tests were performed for each test suite: one test implemented Jacobson's estimation, another Experts', and the last Eifel's. For each test, n measurements were performed, where n=17 for the small suite and n=3000 for the large suite. We began our analysis of results with the small test suite (*figures 8-11*). The graphs show the measured RTT of a specific data packet and the three estimates for that data packet.

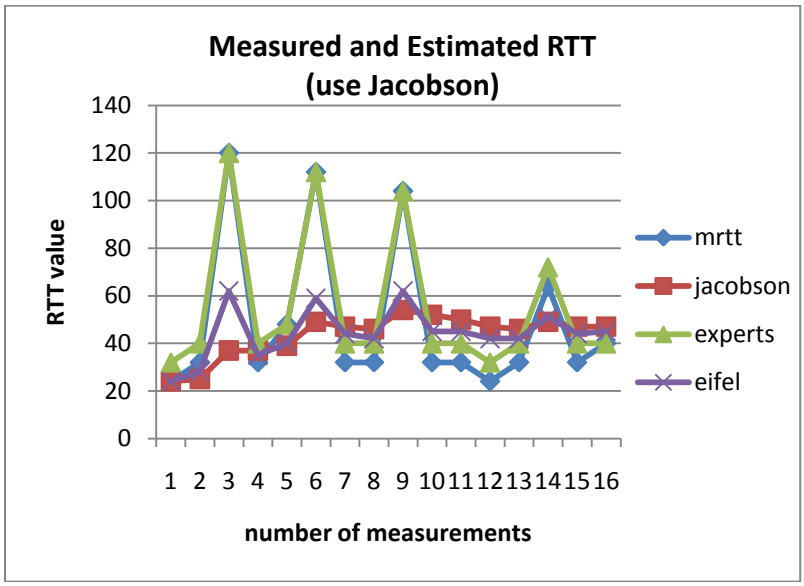

Figure 8: RTT estimation values and the measured RTT value implementing Jacobson

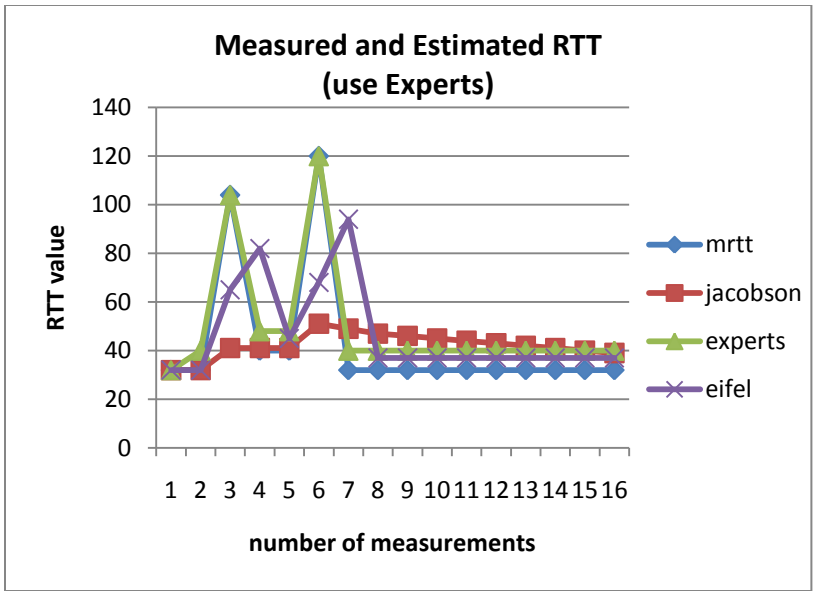

Figure 9: RTT estimation values and the measured RTT value implementing Experts

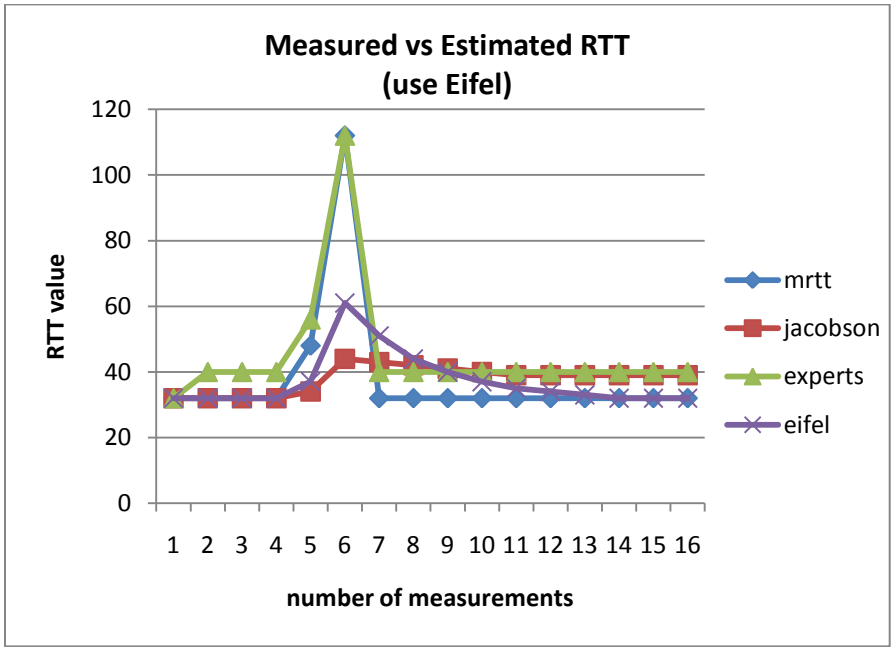

Figure 10: RTT estimation values and the measured RTT value implementing Eifel

It is not enough to simply look at the graphed results, especially when dealing with a test suite as large as 3000 measurements. A metric was developed which encapsulated the data seen in the graphs. First, the difference between the measured value and the approximation value is taken. If this difference is a negative number, then it is an over estimation (OE). In this situation, the approximated RTT indicates to TCP that there is more time than necessary to send the packet, and bandwidth is wasted waiting for the packet to be supposedly transferred. In the case where the difference is a positive number the estimation is an under estimation (UE). The approximated RTT is lower than the

actual time the packet would have taken to be sent, so TCP thinks the packet needs to be retransmitted when in fact it did not have to be. Bandwidth is wasted retransmitting in this case.

The UE and OE for the small test suite are recorded in *tables 1 - 3*. The absolute value has been taken for the over estimations. We see that in all three tests, Experts never under estimates the RTT value so that it never has to retransmit the data. It is also interesting to see that regardless of the mRTT, Experts always over estimates it by 8, which can be seen explicitly in *figures 8-10*. When looking at the nonzero UE, Jacobson's is higher in each test than Eifel, which leads to the assumption that Eifel better under estimates Jacobson, and while under estimating is not necessarily good, it is reduced from the original. Examining the OE, the number seems to be significantly closer to zero than the UE range. Still, Experts is the lowest, followed by Eifel and finally Jacobson. From these preliminary tests, it appears that Experts is the best RTT estimator algorithm, but many more tests and further examination must be performed before such a statement can be declared with strong conviction.

	UE	OE
Jacobson	37.83	14.56
Experts	0	8
Eifel	29.67	10.67

Table 1: Test 1, implementing Jacobson

	UE	OE
Jacobson	66	9.83
Experts	0	8
Eifel	45.5	12.84

Table 2: Test 2, implementing Experts

	UE	OE
Jacobson	41	8.89
Experts	0	8
Eifel	31	8.34

Table 3: Test 3, implementing Eifel

While the method for creating and analyzing the metrics has been developed, the process has not been automated. Therefore, we were unable to examine the metrics for the large test suite at this time. Furthermore, the repercussions of the approximated RTT's on the network have not been examined, such as throughput, goodput, and congestion window.

VII. Conclusion and Future Work

By the end of the summer research program, a module had been built which incorporates and calculates all three proposed RTT estimation algorithms: Jacobson's, Experts Framework, and Eifel. Jacobson's is the original RTT approximator algorithm first incorporated into the TCP protocols; The Experts Framework was developed by the i-NRG team over the past year which uses machine learning to compute RTT values; The Eifel Algorithm is a unique way of computing RTT values whose technique strays from the previously seen techniques of computation.

Due to the complex and abstract nature of the problem, preliminary testing took place only during the final week of the research session. However, the i-NRG team now has a fully functional module for computing three unique RTT values. In the future, they hope to run more in depth and well known test scenarios. Furthermore the team will observe other important metrics in data transfer such as throughput, goodput, and the congestion window. The results of these tests will be analyzed and used to enhance the research paper they submitted last year, seeking a publication.

VIII. References

[1] B. Nunes, K. Veenstra, W. Ballenthin, and K. Obrazcka, *A Machine Learning Approach to End-to-End Round-Trip Time Estimation and Its Application to TCP*, 2009.

[2] R. Ludwig and K. Sklower, *The Eifel Retransmission Timer*, 2000.