wtracert: An optimal timer based traceroute implementation for wireless networks

Ani Taggu
Computer Science and Engineering Department
Rajiv Gandhi University
Itanagar, India
anitaggu@gmail.com

Amar Taggu
Deptt. of Computer Science and Engineering
NERIST
Nirjuli, India
ataggu@gmail.com

***Abstract*—Traceroute program is extensively used in wired networks to track the route packets take on their way to a destination. However, traceroute is often less effective in wireless networks where broken routes are too common. In this paper, the design of wtracert is presented that employs optimal timeouts to provide a faster implementation of the traditional traceroute. Optimal timeout period of timers is based on exponential weighted moving average (EWMA) of round-trip-times of UDP packets. Our linux based implementation of wtracert shows marked improvement over standard traceroute especially when the target is not reachable.**

***Index Terms*—UDP, ICMP, traceroute, EWMA, TTL**

I. Introduction

Traceroute [1] remains a critical piece of software in the toolbox of a network administrator. It has proven its worth as a diagnostic tool over the years. By default, traceroute uses UDP probing to discover routes.

However, it is less effective in wireless networks. Experiments show that the traceroute program, especially in wireless environments, takes too long to detect if a host is not reachable. In such environments, broken routes are frequent. This occurs because of the lossy wireless links and mobility of the nodes.

In this paper, a new implementation of traceroute is proposed for such wireless networks. The primary idea is to terminate traceroute if a given hop is not reachable. The key then, is to find an optimal timeout interval before reporting a broken route. Obviously, the timeout should be larger than a connection's round-trip time (RTT). How much larger should it be is the key question.

The proposed tool, named wtracert, can also be used as a drop-in replacement to traceroute in wired networks. It will simply be a resource-optimized version of traceroute.

The contribution of this paper is the application of well-known Exponential Weighted Moving Average(EWMA) based formulas used in TCP design to create a wireless version of the standard traceroute program. It also implements early detection of a broken route and termination of traceroute.

The remainder of this paper is organized as follows: In Section II, we review some of the related work done. Section III analyzes the classic traceroute algorithm and explains our wtracert implementation in detail. Section IV describes the comparative results obtained for wtracert in wireless networks. Section V endeavors to identify the future directions. Finally, Section VI concludes this paper.

II. Related Works

Van Jacobson, one of the pioneers of TCP congestion control algorithm, is the original author of the traceroute program. Many studies have focused on the limitations of the traditional traceroute. Paris traceroute [2] is focused on the effect of load-balancing routers on the traditional traceroute. Parallel probing using simultaneous probes are explored in [3], [4]. The issues caused by asymmetric links in a network has been discussed in [5], [6]. AS-level route discovery is studied in [7]. Modern traceroute implementations provide multiple methods to track a route including ICMP, UDP, TCP and raw packets as seen in LFT [8] and Linux implementations like [9].

Computation of Retransmission TimeOut (RTO) using exponential aging algorithms in TCP timers are described in [10] and IETF recommendations for managing TCP Timers [11]. These well-known formulas have been used extensively across different TCP implementations.

III. wtracert

Traceroute [1] sends out probes to unlikely destination UDP ports. Whenever a probe returns, it contains this target port. Thus, a history of the packets are maintained. Then, the TTL is increased and the same process gets repeated again.

By default, the maximum number of hops supported by traceroute is 30. In each TTL, the total number of probes is 3 by default. Since the average hop count in Internet is found to be 15 [13], traceroute can, on an average, take 225 seconds or more.

This worst case situation can arise for traceroute when hops along the route are reachable but do not send back any replies to the source host. This could happen, e.g. when packet filters along the route do not respond to UDP queries.

Thus, the standard traceroute takes an inordinately long time under two conditions:

1) *Broken Routes:* Even when route to a host is not detected by traceroute, it continues probing until all 30 hops are over.
2) *Expiry of Polling:* By default, traceroute waits for up-to 5 seconds for a reply to a probe sent by it. However,

978-1-4673-4404-3/12/$31.00 ©2012 IEEE

if a node is reachable within a few milliseconds, it is wasteful to wait for the default timeout of 5 seconds.

Based on the above observations, it makes sense to optimize traceroute by preventing the conditions that cause it to delay. These modifications are as follows:

A. Broken Route Identification

If all the probes in a particular hop expire without receiving a reply, it clearly indicates a broken route. Under such conditions, it is preferable especially in wireless networks to stop tracerouting rather than attempting to probe all the 30 hops. To identify the "Route broken" case, it is simple to check if at least one of the 3 probes in a hop has been replied. We use a single variable, $nprobes\_ret$, to count the number of probes replied per hop. If $nprobes\_ret = 3$ after the polling process for one hop is over, the route is identified as broken.

B. Optimal Polling period

While polling for response to probes, usually 5.0 seconds are used. However, timeout need not really be that long. For instance, if in a particular hop, the first probe returns a RTT of 100 ms, the second and third probes for that given hop is expected to be near the 100 ms range. However, with the network scenario being dynamic, the very next probe might require a timeout of 2 seconds for the probe reply to arrive.

Thus, a mechanism for computing timeout using measured RTTs (MRTT) is required. More precisely, a running average of the RTTs seen so far can be used to compute how long traceroute should wait. This mechanism is already used to compute TCP retransmission timeout (RTO) as described in [10], [12]. We propose to reuse the same EWMA equations used to compute estimated RTT or "smoothed" RTT (SRTT) as in TCP, but use these equations for traceroute also.

$$SRTT \leftarrow (1-\alpha) \times SRTT + \alpha \times MRTT \tag{1}$$

$$EDEV \leftarrow (1-\beta) \times EDEV + \beta \times |MRTT - SRTT|$$

EDEV stands for estimated deviation. α and β are constants with typical values of 1/4 and 1/8 respectively. Finally, the timeout for a probe is computed using the equation

$$Timeout \leftarrow SRTT + 4 \times EDEV \tag{2}$$

C. wtracert Algorithm

The strategies discussed in the previous sections are incorporated into the standard traceroute as shown in Algorithm 1.

Unlike the classic traceroute, the modified Algorithm 1 can exit the $while$ statement prematurely if the variable $final$ becomes true. This condition becomes true if either the destination is reached *or* all the probes in a hop are unreachable. Thus, premature termination is trivially implemented.

When wtracert starts, it sends 3 probes per hop. Pre-emptive probes to hosts at a distance greater than one TTL can be dispatched but is has the potential to cause broadcast storms especially in the opposite direction [9]. After the probes have been sent, polling starts with a given timeout. Whenever a reply to a probe arrives, the sequence number is used to identify the precise probe for which the corresponding reply is received. On receipt of a reply, the $nprobe\_ret$ variable is incremented.

Then, $MRTT$ is computed and printed. Using the measured RTT, the smoothed RTT and the corresponding timeout value is computed using the equations 1 and 2. $srtt\_estimate()$ procedure is a straightforward implementation of the Van Jacobson equations.

Algorithm 1 wtracert(dst)

Require: $dst = hostname \vee dst = hostIP$

```
1:  ttl ← 1
2:  while ttl ≤ 30 ∧ ¬final do
3:    for probe = 1 → 3 do
4:      send_probe(dst, seq, ttl, currtime)
5:    end for
6:    for n = 1 → 3 do
7:      probe ⇐ poll(timeout)
8:      nprobes_ret ⇐ nprobes_ret + 1
9:      MRTT ⇐ compute_mrtt()
10:     print(MRTT)
11:     srtt_estimate(MRTT)
12:     if dst reached then
13:       final ← 1
14:     end if
15:   end for
16:   if nexthop ≠ reachable then
17:     final ← 1
18:   end if
19:   ttl ← ttl + 1
20: end while
```

IV. Experimental Results

Algorithm 1 was implemented and executed in Fedora 13 (2.6.34.9-69). The reference traceroute version used is 1.4a12 with -n option.

To execute wtracert in a wireless network, a Linux-based laptop was connected to our Institutional LAN via the WiMax wireless network.

The first top 100 websites were selected from [14]. Both traceroute and wtracert accept either a hostname or an IP address. Since the time required for DNS name lookup for websites could be variable, all the hosts were probed using their IP addresses. If multiple IP addresses were available for a website, one IP address was selected at random.

For each IP address, wtracert program is executed, immediately followed by the standard traceroute for the same IP address. wtracert, on an average, completes tracerouting within a few seconds. Therefore, the network topology for both the programs remain roughly the same Such a setup enables a meaningful comparison of the relative performance of the two traceroute versions.

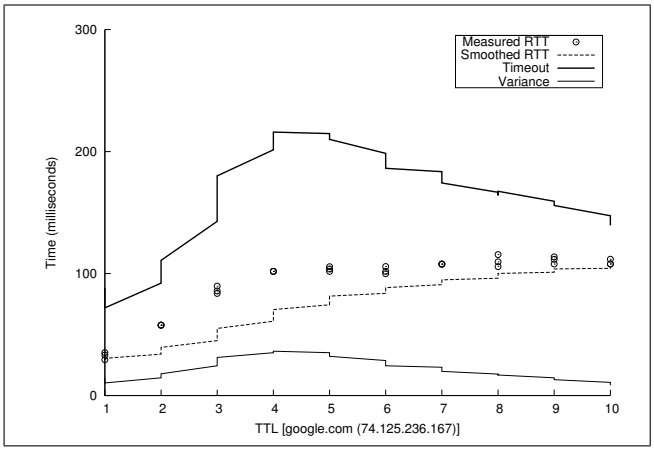

Fig. 1. Optimal Timeout Interval for a successful wtracert session

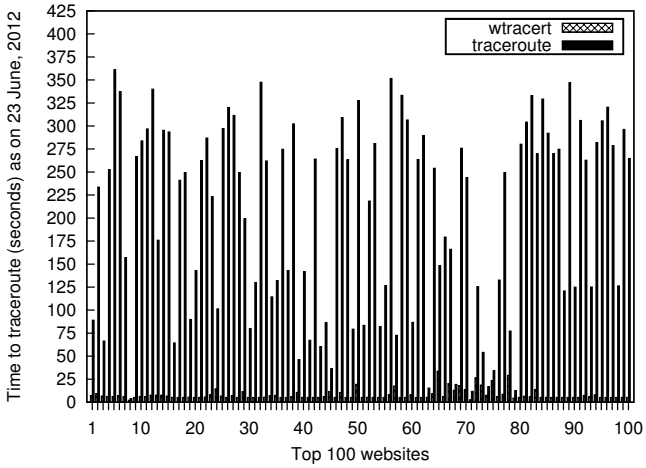

Fig. 2. Comparative Performance in WiMax Network

A. Estimated RTT and Timeout

As seen in Figure 1, the measured RTT, marked by $\odot$, swings variably. The smoothed RTT, on the other hand, shows less perturbations. The variance, or deviation, of the measured RTT from the estimated RTT is also shown. Thus, the effect of equation 1 is clearly seen here.

Now, the plot of timeout shows that when measured RTT varies mildly compared to estimated RTT, the timeout is quite close to the Estimated RTT. However, with larger swings in measured RTT, the algorithm reacts by increasing the timeout quickly. This prevents premature timeouts; after all, if the timeout is too low, it will expire even before the probe reply arrives back to the host. The timeout increases very quickly when the variance of the measured RTT is very high. Essentially, this behavior is the realization of the equation 2.

B. Performance of wtracert

For each IP address, wtracert and traceroute were executed turn by turn. The elapsed real time between invocation and termination of the wtracert and traceroute program were recorded for comparison. The result so obtained for the wireless network is shown in Figure 2. The wtracert performs far better than the standard traceroute.

One primary reason for the sluggish performance of classic traceroute is the presence of packet filters and firewalls in today's Internet which are typically configured to ignore connection requests to unlikely UDP ports. No valid route can thus be found using UDP probes.

V. Future Directions

Since the firewalls and filters are increasingly blocking UDP probes in today's Internet, TCP based probing can be implemented for wtracert. However, since TCP has a number of issues with wireless networks, other probe methods need to be investigated for implementation in wtracert.

VI. Conclusion

The classic traceroute is a critical tool in network administration and design. However, when a route is broken, it takes an inordinate amount of time to detect this condition. Especially in wireless networks where broken routes are common, classic traceroute becomes less effective.

In this paper, a modification to the basic traceroute algorithm has been proposed. Exponential aging algorithms from TCP congestion control are used in a novel way to compute optimal timeouts for traceroute instead. Coupled with early detection of broken route, wtracert overcomes suboptimal latency associated with the classic traceroute.

The experimental results clearly demonstrate the superiority of wtracert over classic traceroute both for wired and wireless networks.

References

[1] Van Jacobson, *traceroute*, ftp://ftp.ee.lbl.gov/traceroute-1.4a12.tar.gz.
[2] Augustin B., Cuvellier X., Orgogozo B., Viger F., Friedman T., Latapy M., Magnien C. and Teixeira R., *Avoiding traceroute anomalies with Paris traceroute*, Proc. ACM SIGCOMM IMC, Pages 153-158, 2006.
[3] *NAGOG traceroute*, ftp://ftp.login.com/pub/software/traceroute.
[4] Moors T., *Streamlining traceroute by estimating path lengths*, Proc. ACM SIGCOMM 2006, Pages 153-158, 2006.
[5] Paxson V., *End-to-end routing behavior in the Internet*, Proc. ACM SIGCOMM, Pages 25-38, 1996.
[6] Katz-Bassett E., Madhyastha H.V., Adhikari V.J., Scott C., Sherry J., Wesep P.V., Anderson T. and Krishnamurthy A.,*Reverse traceroute*, Proc. of 7th NSDI USENIX conference, 2010.
[7] Mao Z., Rexford J., Wang J. and Katz R.,*Towards an accurate AS-level traceroute tool*, Proc. SIGCOMM, pp 365-78, Aug 2003.
[8] McCarthy N., Oppleman V., Antsilevitch E., Kondryukov S., Kanner Z., Davis L., Ballard R., Andrei F.and McKim J., *Layer Four traceroute*, http://pwhois.org/lft/index.who last downloaded 20th June 2012.
[9] Linux traceroute, *traceroute(8)*, Linux manual page for traceroute(8), 11 October 2006.
[10] Jacobson V., *Congestion Avoidance and Control*, ACM SIGCOMM'88, vol. 18, no. 4, August 1988.
[11] Paxon V., Allman M., Chu J., and Sargent M.,*Computing TCP's Retransmission Timer*, RFC 6298, June 2011.
[12] Karns P. and Partridge C., *Improving Round Trip Time Estimates in Reliable Transport Protocols*, ACM SIGCOMM'87, vol. 17, no. 5, August 1987.
[13] F. Begtasevic and P.V. Mieghem, *Measurements of the Hopcount in Internet*, Proceedings of Passive and Active Measurements Workshop, April 2001.
[14] *Top 1000 Websites*, http://www.google.com/adplanner/static/top1000, April 2011, last downloaded on 20th June, 2012.