Measuring TCP Performance Metrics with Bro

Leonhard Stemplinger, Simon Bauer*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: leonhard.stemplinger@tum.de, bauersi@net.in.tum.de

Abstract—**The Transmission Control Protocol (TCP) is one of the most widely used networking protocols. The ability to accurately measure the performance of a TCP connection is important for identifying network problems. In this paper, we present an implementation of three TCP performance metrics: the inter-arrival time of acknowledgements, the time-series of retransmissions and the retransmission score. We compute these metrics using the Bro network analysis framework.**

Index Terms—**tcp, bro, retransmissions, network monitoring**

1. Introduction

TCP is the most common transport layer protocol, with over 90% of internet traffic transmitted over TCP [14]. This makes analysing TCP connections important for network operators, as performance problems could impact the majority of users. One indicator of possible network problems is a high rate of retransmissions, as this indicates a high rate of packet loss. We present Bro scripts to measure several TCP performance metrics for both live connections and trace files. These metrics are the inter-arrival times of acknowlegments, a time-series of retransmissions, and the retransmission rate. They were defined by Siekinnen et al. [2] as part of a root cause analysis framework. We extend their work by describing a possible implementation in detail. Additionaly, our implemetation is able to analyse live traffic, while [2] is limited to trace files.

In this paper, we will first summarize important aspects of the TCP protocol and the Bro network monitor in Section 2. We continue by defining the TCP perfomance metrics implemented for this work in Section 3 and listing other works dealing with related topics in Section 4. In Section 5 we describe an implementation of these metrics in Bro scripts and present the results computed by the scripts in a test in Section 6. We show possibilities for further work in Section 7 and provide access to the Bro scripts and conclude the paper with Section 8.

2. Technical Background

2.1. TCP Protocol

TCP is a transport layer protocol. It relies on the underlying Internet Protocol to transfer packets to their destination. TCP aims to provide a reliable connection even over unreliable network infrastructure [9]. To achieve this, a TCP connection must be established using a three-way-handshake, and both endpoints must keep status information over the lifetime of the connection. Additionally, damaged or lost packets must be detected and retransmitted.

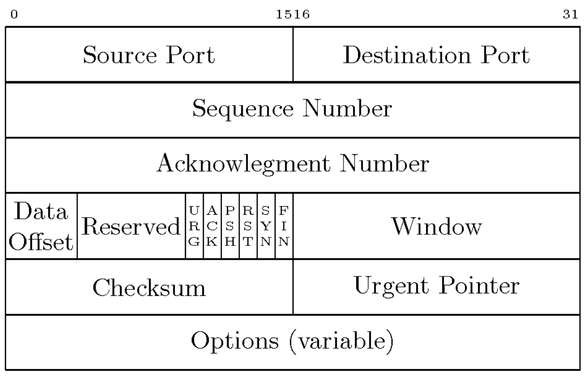

Figure 1. The TCP header [9]

The most relevant part of TCP for this paper is the acknowledgement mechanism. TCP packets carry a sequence number, which is incremented for every byte sent by an endpoint. The receiver acknowledges correctly received data by incrementing the responses acknowledgment number. A packet with set ACK flag and an acknowledgment number n indicates that all data up to and including sequence number $n - 1$ has been received by the connection partner.

Using this mechanism, TCP attempts to detect and retransmit lost packets. Generally, TCP will assume a packet was lost if it is not acknowledged within a certain timespan, or when receiving multiple acknowlegments for previous packets. Details vary between TCP implementations [6]. If a TCP endpoint receives non-contiguous data, meaning data has been lost, but a later packet was received successfully, the selective acknowledgment (SACK) option can be used to notify the sender of the received data and avoid unneccessary retransmissions [10] [12].

2.2. Bro

Bro [8] is a free, open source network monitoring tool. It was originally published by Paxson in 1999 [1]. Bro can process both trace files and live traffic.

Bro is divided into two main components. The Event Engine generates events based on the observed network activity. The Policy Script Interpreter executes scripts in response to the generated events. Bro scripts are written in the Bro scripting language. Each Bro script defines a number of event handlers that are called whenever the corresponding event is triggered. While Bro comes with a large library of scripts for traffic analysis, it can also be extended with custom scripts.

We mainly chose Bro because the abstraction provided by the event system made development of our traffic analysis scripts easier and faster. Bro scripts do not need to perform low level traffic processing such as separating TCP and non-TCP traffic, detecting the TCP connection a packet belongs to or extracting header information from raw packets, as this is handled by the event engine. Additionally they work on live traffic as well as trace files without modifications.

3. Metrics

This paper describes a solution to measure three of the metrics defined in [2] using Bro scripts. This section describes the selected metrics. All metrics are measured separately for each direction of a TCP connection.

3.1. Inter-Arrival Times of Acknowledgements

Each endpoint of a TCP connection acknowleges correctly received packets. For every acknowlegement, we record its' arrival time, the number of acknowledged bytes and the interval since the last observed acknowledgment. Packets that do not advance the acknowledgement number are not recorded. The inter-arrival times can be used to estimate the capacity of the connections' network path [2].

3.2. Retransmission Metrics

TCP detects lost packets by monitoring acknowledgments and retransmits those packets. For this paper, retransmitted packets are defined as packets with a sequence number lower than or equal to the highest previous sequence number. The definition in [2] additionally demands an IPID higher than all previous packets to eliminate false positives caused by out of order packets. However the IPv4 specification has since been updated to allow arbitrary ID values for non-fragmented packets [3]. Furthermore, non-fragmented IPv6 packets do not carry any ID value [16]. Packets without a payload (i.e. pure ACKs) are not included in the analysis, as they do not advance the sequence number.

We measure two metrics to analyse retransmissions. The timestamp and payload size of each retransmitted packet are recorded to create a time-series of retransmissions. Additionaly, we keep track of the amount of data retransmitted, and the total amount of data transmitted. The ratio of retransmitted data to total data is the retransmission score. A high retransmission score generally correlates to a high rate of packet loss. If this occurs frequently, it can indicate a network problem.

4. Related work

The TCP performance metrics implemented for this paper are a subset of those described by Siekinnen et al. [2]. They define several other metrics, as well as a procedure to determine limiting factors for a connections' throughput based on these metrics. However, they do not provide implementation details.

There have been many other works studying TCP retransmissions. Examples include Pentikousis et als. [5] analysis of aggregate retransmission rates among a large number of connections. Rewaskar et al. [6] and Jaiswal et al. [7] present methodologies to classify out-of-order packets into retransmissions and reordered packets.

The TCPRS Bro plugin by Swaro [13] extends Bro with additional events for reordered and retransmitted packets. While the implementation for this paper is written entirely as Bro scripts, TCPRS adds a new analyzer to the Bro event engine.

Most publications related to Bro deal with network security topics, such as intrusion detection, and not network performance. One exception is Sargent and Allmans analysis [15] of the limiting factors for very high bandwith (1 Gb/s) residential fiber connections. They report that current TCP implementations do not use such a connection efficiently, as receiver window sizes limit data transmission to a far lower rate.

5. Implementation

This section describes the Bro scripts that implement the metrics described above.

5.1. Common components

This section describes some patterns that occur in both scripts.

5.1.1. State Information. Both scripts need to keep information about previous packets observed in each direction of each connection. For this purpose, both scripts include two tables to store this information, one for each direction of a connection. A Bro script table is a key-value store. In this case, the keys are the unique IDs (uids) generated by bro for each connection, and the values are records, a data type similar to C structs, that hold information about the connection.

Records are added when observing a packet that does not belong to a previous connection. They are deleted when Bro deletes the connection by handling the connection_state_remove event.

5.1.2. Analyzing new TCP packets. To monitor TCP connections, both scripts register a handler for the tcp_packet event, which is triggered for every TCP packet . The handler receives a connection record, a flag indicating by which endpoint the packet was sent and the values of various TCP header fields. The connection record has a large number of fields which hold information about aspects of the connection. For our scripts, only the connection uid is needed.

5.1.3. Logging. Both scripts record their results in Bro log files (acks.log, retransmission_series.log and retransmission_scores.log). While the result format is different, the first three items of each entry are the same in all logs:

- Connection UID: For cross-referencing with other Bro logs
- Timestamp
- from_orig: A boolean flag that indicates which direction of the connection the entry concerns.

5.2. Inter-arrival times of Acknowledgements

The acknowledgment script stores two numbers for each direction of a connection: The highest ack number of the observed packets, and the timestamp of that packet. For each new packet, the script first checks wether the packet is relevant for the time-series. Packets without a set ACK flag, as well as packages whose acknowledgment number is lower than or equal to the highest number recorded for the same direction are filtered out. If the packet passes this filter, the stored information about the corresponding connection is updated, and a log entry is created. In addition to the fields listed in section 5.1.3, it contains the packets' acknowledgment number, the number of bytes acknowledged (i.e. the difference between the new ack number and the previous one), and the interval since the last acknowldegment.

5.3. Retransmission Metrics

For each direction of a connection the retransmission metrics script stores the maximum sequence number, the number of bytes previously retransmitted and the total number of bytes transmitted, including retransmissions. It also stores wether these values have changed since the last time the retransmission score was computed. Whenever a new packet with a payload is received, the script updates the amount of transmitted data. It also determines wether the packet is a retransmission by comparing its' sequence number to that stored for the connection. If it is not a retransmission, the new maximum sequence number is saved. If it is, the number of retransmitted bytes is updated, and an entry is added to retransmission_series.log.

The information gathered this way is used to calculate the retransmission score. The script defines a new event score_log_trigger and schedules it to trigger every 0.1 seconds. In a handler for this event, the script updates retransmission_score.log. For each direction of each connection, a new log entry is created, if there was a packet observed in this direction since the last log entry.

6. Evaluation

The scripts were tested on a packet capture of the download of part of the Bro documentation. Both Figure 2 and Figure 3 show results for the same connection.

Figure 2 shows that over this connection, data was sent almost exclusively from the responder to the originator. The only packets acknowledged by the responder are those necessary for the TCP and TLS handshakes and for the TCP teardown. After the data transfer, there was a period of inactivity for about two seconds before the connection closed. As seen in Figure 3, the connection experienced a spike in retransmissions after approximately two seconds without retransmissions. Afterwards, the retransmission score decreased again as more data was transmitted.

```
type ack_time: record{
        ack: count;
        timestamp: time;
};

global last_orig_acks: table[string] of
↪  ack_time;
global last_resp_acks: table[string] of
↪  ack_time;

event tcp_packet(c: connection, is_orig:bool,
↪  flags: string, seq: count, ack: count,
↪  len: count, payload: string){
        if ("A" !in flags){
                return;
        }
        local timestamp=network_time();
        local first_ack: bool;
        local last_acks=last_resp_acks;
        if (is_orig){
                last_acks=last_orig_acks;
        }
        first_ack=(c$uid !in last_acks);
        if (first_ack){
                last_acks[c$uid]=ack_time($ack=ack,
                ↪  $timestamp=timestamp);
        }
        if (ack <= last_acks[c$uid]$ack &&
        ↪  !first_ack){
                        return;
        }
        handle_ack(last_acks[c$uid], ack,
        ↪  timestamp, is_orig, c, first_ack);
        last_acks[c$uid]=ack_time($ack=ack,
        ↪  $timestamp=timestamp);
}
```

Script 1: The acknowledgement scripts' tcp_packet handler

```
event score_log_trigger(){
        local ts=network_time();
        log_scores(orig_info, T, ts);
        log_scores(resp_info, F, ts);
        schedule 0.1sec {score_log_trigger()};
}
```

Script 2: Retransmission score logging

7. Future Work

Currently, our script uses a very simple retransmission detection mechanism. The precision of of the retransmission metrics could be improved by a more sophisticated mechanism that is able to separate true retransmissions and reordered packages, removing false positives.

As mentioned in section 4, not all of the metrics described in [2] were implemented for this work. The remaining metrics could be implemented in similar Bro scripts. This would enable the use of the root cause analysis procedure described in [2].

However, Bro scripts might not be the best tool for calculating these metrics, in particular for higher bandwidth connections. The Bro documentation [4] warns of

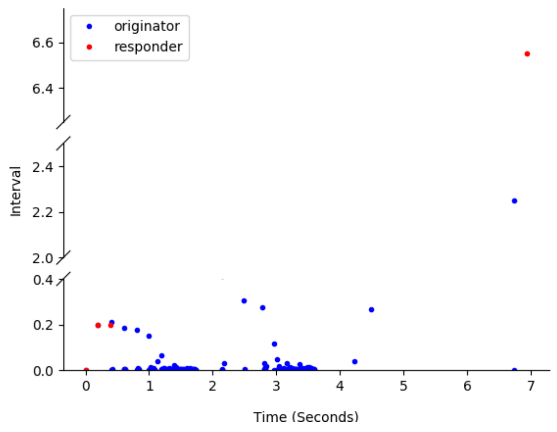

Figure 2. A plot of the intervals between acknowledgments over the duration of a connection.

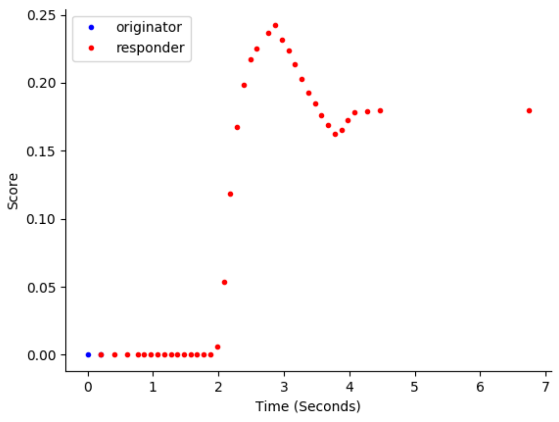

Figure 3. A plot of the retransmission score over the duration of a connection

the performance impact of handling the tcp_packet event. While no performance problems were observed in our tests, these were done with fairly small trace files and results could change in higher load situations. Additionally, the tcp_packet event does not expose the size of the TCP window, which is needed for some metrics. To access this value, a script needs to use the more general new_packet event. This could add to to any performance problems, as new_packet is triggered more frequently than tcp_packet. One solution to possible performance problems could be moving packet classification from the tcp_packet handler to a custom Bro traffic analyzer. As Bros' traffic analysis layer is built in C++, it would likely run far faster than the interpreted scripts.

8. Conclusion

In this paper, we summarized the TCP acknowlegment and loss recovery mechanism and introduced three performance metrics based on this mechanism: the inter-arrival times of acknowledgments, the time-series of retransmissions and the retransmission score. We described Bro scripts that compute these metrics for both trace files and live traffic and tested them on real-world internet traffic.

Both of our scripts are available at [11]. For instructions on how to set up Bro and run custom Bro scripts, please refer to the Bro documentation [8].

The implemented metrics are of limited use on their own. However, this work shows the feasibility of using Bro for TCP performance analysis. Combined with scripts for other metrics, the Bro scripts shown here could form part of a more sophisticated measurement toolkit.

References

[1] Vern Paxson, Bro: A System for Detecting Network Intruders in Real-Time, Computer Networks, 31(23-24), pp. 2435-2463, 1999

[2] Matti Siekkinnen, Guillaume Urvoy-Keller, Ernst W. Biersack, Denis Collange, A root cause analysis toolkit for TCP, Computer Networks, Volume 52, Issue 9, Pages 1846-1858, 2008

[3] J. Touch, Updated Specification of the IPv4 ID Field, RFC 6864, 2013

[4] Documentation of the Bro_TCP.events script, www.bro.org/sphinx/scripts/base/bif/plugins/Bro_TCP.events.bif.bro.html

[5] Kostas Pentikousis, Hussein Badr, Asha Andrade, A Comparative Study of Aggregate TCP Retransmission Rates, International Journal of Computers and Applications, 32:4, 435-441, 2010

[6] Sushant Rewaskar, Jasleen Kaur, F. Donelson Smith, A Passive State-Machine Approach for Accurate Analysis of TCP Out-of-Sequence Segments, ACM SIGCOMM Computer Communication Review, Volume 36 Issue 3, Pages 51-64, 2006

[7] Sharad Jaiswal, Gianluca Iannacone, Cristophe Diot, Jim Kurose, Don Townsley, Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone, IEEE/ACM Transactions on Networking, Volume 15 Issue 1, Pages 54-66, 2007

[8] The Bro Network Security Monitor, www.bro.org

[9] Transmission Control Protocol, RFC 793, 1981

[10] M. Mathis, J. Mahdavi, S. Floyd, A. Romanov, TCP Selective Acknowledgement Options, RFC 2018, 1996

[11] github.com/lstemplinger/bro-tcp, Commit c4b6dc9

[12] E. Blanton, M. Allman, L. Wang, I. Jarvinen, M. Kojo, Y. Nishida, A Conservative Loss Recovery Algorithm Based on Selective Acknowledgement (SACK) for TCP, RFC 6675, 2012

[13] James Swaro, TCP Retransmission and State Analyzer plugin for the Bro-IDS framework, github.com/jswaro/tcprs

[14] DongJin Lee, Brian E. Carpenter, Nevil Brownlee, Media Streaming Observations: Trends in UDP to TCP Ratio, International Journal on Advances in Systems and Measurements, vol 3 no 3&4, 2010

[15] Matthew Sargent, Mark Allman, Performance within a fiber-to-the-home network, ACM SIGCOMM Computer Communication Review, Volume 44, Issue 3, pages 22-30, 2014

[16] S. Deering, R.Hinden, Internet Protocol Version 6 (IPv6) Specification, RFC 8200, 2017