TCPMoon: Monitoring the Diffusion of TCP Congestion Control Variants in the Internet

Gianni Casagrande, Fabrizio Granelli
DISI, University of Trento
Via Sommarive 14, 38123 - Povo, Trento (Italy)
email: g.casagrande.2@studenti.unitn.it
granelli@disi.unitn.it

Daniele Miorandi
CREATE-NET
Via alla Cascata 56/D, 38123 - Povo, Trento (Italy)
email: daniele.miorandi@create-net.org

Abstract—TCP congestion control mechanism has a critical impact on the Internet stability and performance. In relatively recent times, a number of novel TCP congestion control variants, such as TCP BIC, CUBIC and Compound, started to be deployed in modern operating systems. While risks and benefits of these recent developments are subject of debate, the research and engineering communities need to monitor the extent to which each of these novel TCP congestion control variants is actually used in regulating Internet traffic, and this information, due to the complex, diverse, and decentralized nature of the Internet, is very difficult to obtain.

As a first step towards the collection of this important piece of information, we have developed a tool, called tcpmoon, which allows us to identify the TCP variants employed in monitored TCP sessions. Our approach consists in passively monitoring packets and ACKs exchanged in TCP sessions, estimating the evolution of congestion window, and matching a set of features against those typical of the various TCP variants considered. Our identification tool has been validated through emulation and real Internet experiments, showing promising results in the identification of NewReno, BIC, Cubic and Compound TCP variants.

I. INTRODUCTION

Congestion control mechanisms are at the very heart of the Internet stability and efficiency. Introduced in the late 80s following a number of collapses of the rising Internet, they have followed a rather stable evolution path until few years ago, moving from TCP Tahoe to Reno, then to NewReno with the successive adoption of the selective acknowledgment option. Within such a path, there was basically one “globally accepted” standard TCP version, defined by IETF RFCs and implemented, with some variations, by all computing systems.

In the last few years, however, we have been witnessing the mushrooming of a number of TCP variants [1]. While the vast majority of such TCP variants has found interest limited to the scientific community, some of them have made their way into widely adopted operating systems protocol stacks. This is the case of TCP BIC [2] (adopted as default by Linux starting from kernel 2.6.13), TCP CUBIC [3] (default by Linux from kernel 2.6.19) and TCP Compound [4] (included in Windows Server 2008 and Windows Vista). Monitoring the adoption of such protocols (and the amount of traffic they control) is key to understanding and studying the global stability of the Internet.

The goal of this work is to devise a suitable framework for the passive measurement of the adoption of different TCP variants in wide area networks. The problem can therefore be reduced to the ability of identifying, in a real-world context, the congestion control algorithm driving the behavior of a TCP flow. In particular, we focus on four TCP variants: NewReno, BIC, CUBIC and Compound. Existing tools are not adequate, as they either do not support/recognize the aforementioned protocols (e.g.,[5], [6]) or are able to distinguish only between two types of protocol at a time [7].

In this paper, we introduce tcpmoon, a monitoring and identification framework we have developed to address the issue of classifying TCP-controlled flows in wide area networks according to the congestion control algorithm used. tcpmoon uses standard passive monitoring tools, at any point within the communication path, to estimate the behavior of the congestion window parameter at the TCP sender. It splits the trace obtained into periods, for each of them a number of features is evaluated and used to infer the likelihood that it has been generated by one of the considered TCP variants.

The main contributions of our work can be summarized as follows:

- We introduce a framework for the classification of TCP variants in the Internet. The framework, based on passive monitoring, does not generate additional traffic and is designed to work at any intermediate point in the network.
- We present a set of algorithms which are used by the framework to (i) estimate the congestion window evolution at the sender (ii) extract some important features from the congestion window trace (iii) classify flows into the TCP variants currently supported.
- We validate the framework and algorithms by presenting numerical results obtained by means of emulation and real-life Internet testing, using a prototypical implementation of the proposed approach.

The remainder of this paper is organized as follows. Sec. II surveys the related work and compares our approach to the state-of-art. Sec. III describes the algorithms employed by our approach, detailing the various steps and how they can be implemented. Sec. IV reports on the experiments carried out in order to validate the ability of our approach to correctly classify TCP flows. Sec. V concludes the paper pointing out promising directions for future extensions.
II. RELATED WORK

This work on the identification of TCP congestion control variants in the Internet is motivated by a major evolution in Internet-level congestion control: the adoption of new high-speed TCP variants by commonly-used operating systems. We can trace this back to 2005, when Linux kernel 2.6.13 was first released, with TCP BIC [2] replacing TCP NewReno [8] as the default congestion control mechanism.

Most of the works present in the literature on the passive identification of TCP variants were developed earlier on, and were focusing on the analysis of the different implementations of a given congestion control protocol, in order to identify violations of the protocol specification. Examples of such line of research include tcpanaly [5] and tcpflows [6], developed in 1997 and 2004, respectively.

tcpanaly [5] employs detailed knowledge of 11 TCP congestion control implementations (referring to different operating systems), which at the time of its development constituted almost all of the observable TCP implementation in the Internet. These implementations employ slightly different versions of the two classic congestion control algorithms available at the time: TCP Reno [9] and TCP Tahoe [10]. Identification is performed by tracking packets and ACKs throughout a TCP session. For each TCP implementation embedded in the tool, a replica of the congestion control mechanism in the monitored host is generated, each one being fed with the monitored ACK stream. The resulting congestion window trace output is matched against the monitored packet flow. A matching algorithm is then employed to estimate which protocol is most likely to be used at the sender. This replica–based approach is very effective in identifying TCP algorithms, but it requires detailed knowledge (at the level of code implementation) of all the possible congestion control algorithms to be identified, as well as accurate information on the packet and ACK streams observed at the monitored host.

In particular, the latter requirement implies that, in order for the tool to be effective, packets and ACKs should be captured in the immediate vicinity of the TCP sender. Besides not supporting the TCP protocols we are interested in, the replica-based approach of tcpanaly is not suitable to our problem, as it requires (i) detailed knowledge on the implementation of different TCP variants in various operating systems (ii) packet flows to be monitored very close to the sender.

tcpflows [6] is an identification tool aimed at the collection of global statistics on the diffusion of congestion control algorithms, which applies and extends tcpanaly’s replica–based approach towards remote identification (i.e., far from the sender host). Monitoring at a location which is far from the host under analysis significantly complicates the identification task, as the monitoring node needs to compensate for what may happen (delay, packet losses, reordering etc.) in the portion of the network separating it from the sender node. The variety of TCP implementations embedded in tcpanaly is not present in tcpflows, which instead embeds only one single version of the three main classic TCP algorithms: Tahoe, Reno and NewReno. The fact of using a replica–based approach still limits the flexibility of the tool (in terms of TCP variants it can identify) and its ability to deal with the uncertainty arising due to the remote location of the monitoring agent. Validation of tcpflows is only performed through emulation using BSD machines, embedding the dominant TCP Reno and NewReno implementations [6]. Real Internet experiments are also proposed, but without ground truth references allowing to actually assess the accuracy of the results.

A recent work in congestion control identification, considering NewReno, BIC, CUBIC and Compound is reported in [7]. We will refer to the tool there presented there as tcpcluster in the remainder of this paper. The proposed identification algorithm is focused on minimal computational complexity and aims to be employed in real-time applications such as traffic policing in network access routers. tcpcluster performs a rough round-trip-time and congestion window estimation in order to extrapolate a set of simple features from the monitored flows. The extracted features are then clustered, using standard machine-learning approaches, into two groups, each one referring to a possible TCP variant. Although applicable to a very broad range of TCP algorithms and computationally inexpensive, tcpcluster can only be used to discern between two given TCP variants, rather than among the variety of algorithms observable in the Internet.

tcpmoon similarly to [6], aims to be applied in the collection of global Internet statistics on the usage of today’s most diffused congestion control algorithms. Our overall identification approach is closer to [7], but it manages to overcome tcpcluster limitations and can be directly applied to the identification of TCP variants in real Internet traffic.

III. APPROACH AND ALGORITHMS

The starting point in our framework is the availability of a log file tracing all packets flowing through a given host. This can be obtained using tcpdump1 or similar tools.

The approach employed by tcpmoon for identifying TCP variants consists then of three stages. First, it computes, using the log file, an estimate of the congestion window evolution at the sender side. Then, the obtained estimate is analyzed in order to identify relevant events and to filter out undesired noise. Last, a number of features is computed and matched against the characteristics of the TCP variants being considered.

A. Congestion Window and Round-Trip Time Estimation

tcpmoon estimates the congestion window by counting the number of packet transmissions performed within a Round-Trip Time (RTT). This is based on the fact that the TCP congestion window is a limit imposed by congestion control algorithms on the number of packets sent and not yet acknowledged (also called flight-size [11]). This requires the round-trip time to be estimated throughout the analyzed TCP session.

The RTT estimation is performed in tcpmoon employing a mechanism originally introduced in [12], which requires

1http://www.tcpdump.org/
to trace both ACKs and data packets. Let us consider a monitoring agent located anywhere in the path between the sender and the receiver. The round-trip time can be computed as the sum of two time intervals: a sender–to–monitor round-trip time and a monitor–to–receiver round-trip time. The first interval separates the detection of an ACK from the detection of the corresponding triggered packet transmission, while the second interval separates the detection of a packet from the detection of its corresponding ACK.

Coupling packets with their corresponding ACK is simple, as each ACK carries the sequence number of the packet it acknowledges. Coupling ACKs with the triggered packet transmissions is, in general, non-trivial. In tcpmoon we decided to rely on the widespread adoption of the TCP timestamps option [13] in modern TCP stacks in order to perform it. In TCP sessions where the timestamps option is used, each TCP data packet carries both the of its transmission by the sender and a copy of the time-stamp carried by the ACK that triggered its transmission. By recording the timestamps of ACK packets, and by mining the timestamps field of TCP data packets, it is possible to accurately estimate both components of the round-trip time.

B. Congestion Epoch Detection and Filtering

Once a first estimate of the congestion window is obtained by means of the procedure described in the previous section, the next step is to process it in order to enable an effective template matching. Three factors mainly influence the accuracy of estimated congestion window: loss recovery mechanisms, round-trip time variability and estimation noise.

1) Loss Recovery Detection: Congestion control algorithms present recovery mechanisms, which are triggered when a packet loss has been detected, and take control over the congestion window evolution until the loss has been recovered. In tcpmoon we remove from the congestion window estimate the portions which refer to loss recovery.

tcpmoon analyzes the data packet stream in order to detect loss events. In particular, it scans for the reaction by the sender to a packet loss. Monitoring packets at a midpoint between a TCP sender and its receiver, loss is observable in two different forms, depending on whether data packets are lost before or after the monitoring host. In the latter case, packets with the same sequence number shall appear multiple times. In the former case, instead, when packets are lost before the packet monitor, a packet sequence gap (i.e., a missing block of data) will be detected in the monitored packet stream.

Sequence gaps do not necessarily imply packet losses, as packets can be out-of-order at the monitoring location. As the TCP sender cannot detect a lost packet before one RTT from its transmission, tcpmoon considers a detected sequence gap to be associated to a packet loss if and only if the missing packet does not appear within a RTT interval on the data packets stream. Recovery periods begin when retransmission is performed, and, as just described, these retransmissions are detected in the form of multiple packets with same sequence number or in the form of late sequence gap filling.

The duration of a recovery period depends on the specific recovery algorithm implemented by the TCP data sender and on the specific congestion event considered. In tcpmoon, recovery from congestion events is considered over when an ACK is monitored, which acknowledges all of the packets transmitted by the sender before the monitored retransmission.

This approach is consistent with NewReno and Reno SACK recovery algorithm for 3-dupack congestion events and with Retransmission Timeout congestion handling in Linux TCP [14].

Once recovery periods are detected, the congestion window trace can be separated in congestion epochs. These represent portions of the congestion window trace that are not affected by packet losses. In fact, all of the following steps in the identification algorithm operate on such obtained congestion epochs rather than on the entire congestion window trace.

2) Round-Trip Time Normalization: The most common congestion control variants (except for TCP CUBIC) define a congestion window update mechanism that is directly influenced by the round-trip time. For example NewReno increases the congestion window of one packet per RTT when in congestion avoidance, and of one packet per received ACK (i.e. doubles its congestion window every RTT) when in slow-start. Normalizing the timescale of congestion window samples with respect to round trip time enables tcpmoon to detect some of these TCP variant-specific window update metrics, which would otherwise be corrupted by RTT fluctuations. We define the normalized sampling time as:

$$\tau_i = \begin{cases} 0 & i = 1 \\ \tau_{i-1} + \frac{t_i-t_{i-1}}{RTT_i} & \forall i > 1 \end{cases}$$

where $t_i$ is the sampling time for congestion window sample $cwnd_i$, $RTT_i$ is the round-trip time sample associated to $cwnd_i$, and $\tau_i$ is the corresponding normalized sampling time.

3) Filtering and Re-sampling: Estimated congestion window traces may be affected by a considerable amount of distortions and noise. To compensate for these, tcpmoon filters the congestion window samples in each of the congestion window traces by applying a robust local regression smoothing [15]. First, for each RTT a first-order polynomial approximation of the samples obtained is computed using standard linear-regression methods (based on minimum square error). A local weighting function is used in order to give samples which are located closer to the interval center more relevance in the linear fitting process. The difference between original and the fitted sample values is called residual, and is used to compute a second set of robust weights [15]. Robust weight values are higher for samples showing lower residuals, and robust weighting is used in order to minimize the impact of isolated outliers on the overall smoothing process. Residuals of this robustly smoothed signal can be iteratively used to compute new robust weights.

In our matlab implementation of tcpmoon, we use mslowess function to perform local regression smoothing, and robust weight calculation is iterated three times before the obtained robustly smoothed signal is used as the final
smoothed congestion window. An example of the result of this filtering process is reported in Fig. 1.

C. Feature Extraction, Template Matching and Classification

Let the \( w_i \) be the average congestion window size for round-trip time \( i \) within a given congestion epoch, and let the per round-trip time congestion window increment for round-trip time \( i \) be defined as \( d_i = 0.5 \times (w_{i+1} - w_{i-1}) \).

In tcpmoon, each congestion epoch is characterized by the following features based on the statistical properties of \( d_i \):

- **LARGE**: number of large-increment samples, i.e. number of \( d_i \) samples in a congestion epoch where \( d_i > 1.25 \);
- **SMALL**: number of small-increment samples, i.e. number of \( d_i \) samples in a congestion epoch where \( d_i \leq 1.25 \);
- **MED**: statistical median among small-increment samples;
- **DEV**: deviation from median, computed as the maximum absolute difference between MED and the 80% and 20% quantiles of small-increment samples.

We associate values taken by the aforementioned parameters to different TCP variants by means of templates. Templates define the intervals within which the observed values of features MED and DEV are expected to lay, when a TCP flow is controlled by a specific window update mechanism. The templates defined in tcpmoon are listed in Table I.

![Fig. 1. Example of denoised congestion window evolution for TCP CUBIC.](image)

<table>
<thead>
<tr>
<th>TCP mechanism</th>
<th>MEDmin</th>
<th>MEDmax</th>
<th>DEVmin</th>
<th>DEVmax</th>
</tr>
</thead>
<tbody>
<tr>
<td>NewReno: Cong. Avoid.</td>
<td>0.45</td>
<td>1.1</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>BIC: Flat Region 0</td>
<td>0.5</td>
<td>0.1</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>CUBIC: Inflection Point</td>
<td>0.1</td>
<td>1.25</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>CUBIC: TCP-friendly</td>
<td>0.25</td>
<td>0.4</td>
<td>0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

New Reno template captures its congestion avoidance behavior, where the congestion window is constantly incremented of one packet per RTT. SMALL window increments are observed in the inflection point of TCP BIC and CUBIC window update mechanism, but BIC is characterized by a central flat region showing lower congestion window increments while CUBIC’s inflection point exhibits wider variability in window increments. TCP CUBIC additionally specifies a TCP friendly behavior for low-delay high-speed TCP sessions, where the congestion window is constantly incremented of one packet every three round-trip times.

TCP Compound defines a delay based window update mechanism, which is very difficult to be synthesized into a template. Although several solutions have been experimented, the identification Compound did not lead to satisfying results and will be addressed in future work.

Features SMALL and LARGE as well contribute in the identification process: NewReno’s Slow Start and BIC’s logarithmic-exponential update mechanisms exhibit very aggressive window increments which are observed for small portions of a congestion epoch. Consequently, when a congestion epoch shows a majority of LARGE window increments, the epoch is directly associated to CUBIC.

The overall identification algorithm adopted by tcpmoon is summarized in Fig. 2. Features extracted from congestion epochs in a TCP flow are matched with the templates for NewReno, BIC and CUBIC. Epochs which are found compatible with a TCP variant template, contribute to the cumulative probability (or score) of the TCP flow to be controlled by the associated TCP variant. The TCP variant controlling the flow (TCPvar) is finally identified as the variant accumulating highest score in the template matching process.

IV. Performance Evaluation

In this section we present a validation of the framework and algorithms introduced above. The validation has been carried out in two steps, including a computer emulation (using interconnected virtual machines) and real Internet measurements. The two settings have been used to obtain a number of traces, which have then been processed using a prototypical implementation (developed in Matlab) of tcpmoon.

All tests were performed by running a number of TCP sessions with a known congestion control mechanism, logging
packets at the sender/receiver host and using tcpmoon to assess its ability to correctly identify the TCP variant employed.

A. Emulation

1) Tests Setup: We first tested tcpmoon using a simple emulation testbed. We used a single host (dual-core processor) running Linux 2.6.31 kernel. The machine emulates an Internet path and acts as a bridge and for two other virtual machines, one acting as Sender and running Linux 2.6.33 kernel and the other acting as Receiver and running a Linux 2.6.31 kernel. We use the Linux netem tool to emulate link delay and bandwidth limitation.

The emulation setup is sketched in Fig. 3. Ten scenarios have been emulated, the settings used are reported in Table II. For each scenario, 5 runs have been performed, each run consisting of a TCP-controlled transfer for each of the three TCP variants considered: NewReno, BIC and CUBIC. Due to the limited number of runs and the lack of emulated cross-traffic, these emulation experiments do not provide a thorough validation of our approach, but are instead only meant to provide insight into the robustness of the proposed approach to different network conditions.

![Fig. 3. Emulation testbed set-up.](image)

A log was generated at the sender and at the receiver for each performed test. In this way we could test the ability of tcpmoon to operate consistently when employed close to the sender or close to the receiver (the latter case being more challenging due to the effect of the forward path on the congestion window estimation process).

2) Results: Table III reports the correct identification rate achieved by tcpmoon in our emulation tests. As shown, tcpmoon is very effective in identifying NewReno, BIC and CUBIC in all high bandwidth scenario, while for lower bandwidth paths identification is more problematic.

The main cause of the observed identification problems on low bandwidth scenarios lays in the high average round-trip time experienced and in the consequently low number of observed congestion window samples. As packets accumulate behind a buffer at the entry of an emulated bottleneck link, these are delayed by an amount of time which depends on the bandwidth of the link. When packets are queued behind a full 80Kbyte buffer with 100kb/s bottleneck bandwidth, these undergo a queuing delay as high as 6.4 seconds.

The limited number of samples observed in each congestion epoch results in: (i) non-identification of TCP flows, as our identification algorithm discards congestion epochs with less than 5 congestion window samples; or (ii) inflation of DEV feature due to the high impact of outliers in a small sample set, misleading identification towards templates specifying higher DEV values (Reno and BIC identified as CUBIC). This last effect is mitigated when packets are logged at the receiver. Packets which are transmitted in bursts, are then more uniformly spaced by the effect of crossing a bottleneck link [10]. Packet spacing renders receiver side estimated congestion window traces significantly smoother than their sender side counterpart, and thus leading to lower DEV values which compensate for the identification problems described above.

### TABLE III

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Bottleneck Bandwidth</th>
<th>Path Delay</th>
<th>Access Link Bandwidth</th>
<th>Transfer Size</th>
<th>Bottleneck Buffer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100 kb/s</td>
<td>40ms</td>
<td>1 mb/s</td>
<td>4mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>2</td>
<td>100 kb/s</td>
<td>150ms</td>
<td>1 mb/s</td>
<td>2mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>3</td>
<td>300 kb/s</td>
<td>40ms</td>
<td>1 mb/s</td>
<td>4mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>4</td>
<td>300 kb/s</td>
<td>150ms</td>
<td>1 mb/s</td>
<td>4mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>5</td>
<td>1 mb/s</td>
<td>40ms</td>
<td>10 mb/s</td>
<td>8mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>6</td>
<td>1 mb/s</td>
<td>150ms</td>
<td>10 mb/s</td>
<td>8mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>7</td>
<td>3 mb/s</td>
<td>40ms</td>
<td>10 mb/s</td>
<td>15mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>8</td>
<td>3 mb/s</td>
<td>150ms</td>
<td>10 mb/s</td>
<td>15mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>9</td>
<td>10 mb/s</td>
<td>40ms</td>
<td>100 mb/s</td>
<td>30mbyte</td>
<td>80Kbyte</td>
</tr>
<tr>
<td>10</td>
<td>10 mb/s</td>
<td>150ms</td>
<td>100 mb/s</td>
<td>30mbyte</td>
<td>80Kbyte</td>
</tr>
</tbody>
</table>

B. Internet Measurements

1) Tests Setup: The effectiveness of tcpmoon has been validated over two real Internet paths: the first connecting a local residential area with our Research Lab (∼15 km distance), and the second connecting our research lab in northern Italy with a server cluster hosted at the University of Luxembourg.

In the first experiment (∼15 km distance), we performed a set of 40 runs, each one consisting in transferring (for each of the TCP variants considered) a 10MByte file. Experiments have been performed with a Sending host running Linux 2.6.31 kernel and logging packets at both ends of the TCP flow.

In the second experiment (Italy-Luxembourg), we performed a set of 100 runs, each one consisting in transferring (for each of the TCP variants considered), a 10MByte file. The sender settings were the same ones as specified above. For this experiment, sender-side measurements only were used.
2) Results: Figure 4 reports how \texttt{tcpmoon} identifies NewReno, BIC and CUBIC flows in our first real Internet experiment. As shown, the identification of NewReno is very accurate, while several CUBIC flows, when monitored at the sender, are erroneously associated with BIC.

![Identification outcome for NewReno, BIC and CUBIC flows in first real Internet experiment.](image)

The framework builds on a number of logical steps, starting from a packet log file and returning, for each TCP session monitored, a likelihood estimation that the flow was controlled by NewReno, BIC or CUBIC. Validation, performed using emulation and real-life Internet measurements, shows that \texttt{tcpmoon} can effectively distinguish the aforementioned TCP variants, with an average correct classification probability of 92% for both emulation and real Internet experiments.

We identify two main direction of future work. Firstly, in order to achieve better identification accuracy and to enable identification of Compound TCP, our feature extraction approach can be extended to exploit additional pieces of information regarding the congestion window evolution, such as the statistical properties of \textbf{LARGE} window increments, the events of window reduction or the evolution of window increments in time. Secondly, a real-time implementation of \texttt{tcpmoon} can be developed and applied to monitoring the adoption of different TCP variants in the Internet.

**Acknowledgements**

We would like to thank Dzmitry Kliazovich and the University of Luxembourg for granting us access to a remote machine for Internet file transfer experiments.

**References**


