Measuring the Congestion Responsiveness of Internet Traffic*

Ravi S. Prasad and Constantine Dovrolis

College of Computing, Georgia Institute of Technology
{ravi,dovrolis}@cc.gatech.edu

Abstract. A TCP flow is congestion responsive because it reduces its send window upon the appearance of congestion. An aggregate of non-persistent TCP flows, however, may not be congestion responsive, depending on whether the flow (or session) arrival process reacts to congestion or not. In this paper, we describe a methodology for the passive estimation of traffic congestion responsiveness. The methodology aims to classify every TCP session as either "open-loop" or "closed-loop". In the closed-loop model, the arrival of a session depends on the completion of the previous session from the same user. When the network is congested, the arrival of a new session from that user is delayed. On the other hand, in the open-loop model, TCP sessions arrive independently of previous sessions from the same user. The aggregate traffic that the open-loop model generates is not congestion responsive, despite the fact that each individual flow in the aggregate is congestion responsive. Our measurements at a dozen of access and core links show that more than 60-80% of the traffic that we could analyze (mostly HTTP traffic) follows the closed-loop model.

1 Introduction

The stable and efficient operation of the Internet is based on the premise that traffic sources reduce their rate upon congestion. If a link has capacity C, then the offered load at that link should not exceed C for any significant period of time (relative to the maximum possible buffering in the bottleneck link). Individual TCP flows are congestion responsive thanks to the well-known TCP congestion control/avoidance algorithms. Can we expect the same, however, for streams of non-persistent TCP flows? This depends on the characteristics of the random process that generates new flows (or sessions of flows). In this paper, we rely on two well-known models to describe the flow/session arrival process at a link $\mathcal{L}$: the "open-loop" and "closed-loop" models. The former does not generate congestion responsive traffic, while the latter does. We use these models to passively estimate the congestion responsiveness of the aggregate traffic at an Internet link.

In the flow-based *open-loop model*, new flows arrive independent of the load at $\mathcal{L}$, for example according to a Poisson process. The average offered load in

* This work was supported by NSF CAREER award ANIR-0347374.

S. Uhlig, K. Papagiannaki, and O. Bonaventure (Eds.): PAM 2007, LNCS 4427, pp. 176–185, 2007.

the open-loop model is given by λS, where λ is the average flow arrival rate and S is the average flow size. The normalized offered load is $\rho_o = \lambda S/C$. If $\rho_o<1$ the link is stable and ρ_o is its average utilization. Otherwise, if $\rho_o \geq 1$, the link becomes unstable since the number of active flows can grow without bound. If users are impatient and abort ongoing flows after a certain time period, then the number of active flows in the underlying system will remain finite, thereby making the system stable. However, aborted flows result in user dissatisfaction and poor performance [1]. Therefore, an open-loop traffic aggregate can cause instability and/or aborted flows, even if all flows use TCP. In other words, *TCP congestion control cannot avoid persistent overload when flows arrive according to the open-loop model*.

In the flow-based *closed-loop model*, we have a fixed number of users N. Each user goes through a cycle of activity, with a flow of average size S followed by an idle period of average length T_i. The average flow arrival rate is given by $\lambda_c = N/(T_t + T_i)$, where T_t is the average flow completion time. The latter depends on the load at the link, as well as on TCP (e.g., on the slow start algorithm). When $\rho_c = \lambda_c S/C \ll 1$, users spend most time thinking (i.e., $T_t \ll T_i$), and the system behaves similar to the open-loop model with arrival rate $\lambda_o = N/T_i$. However, when ρ_c approaches or exceeds 1, the number of active flows in the server increases, reducing the average per-flow throughput and increasing T_t. The increase in T_t reduces the flow arrival rate, keeping the offered load close to the capacity, i.e., $\lambda_c S \approx C$. This means that *the closed-loop traffic model is always stable and it cannot cause overload*.

A direct way to measure the congestion responsiveness of the traffic at an Internet link would be to cause a short-term congestion event, and then examine whether the flow arrival process is affected. Creating congestion events to measure the responsiveness of the traffic, however, is a highly intrusive experiment and it is often not even possible. In Section 3, we describe an indirect procedure to estimate the congestion responsiveness of the aggregate traffic at a link by classifying each session as either open-loop or closed-loop, and then measuring the fraction of traffic that follows the latter. We refer to this fraction as the *Closed-loop Traffic Ratio (CTR)*. A higher CTR value implies more congestion responsive traffic.

Our CTR estimation technique is based on the analysis of the interarrivals of packets, flows, and sessions. At this point, the technique is mostly applicable to well understood client-server applications, such as HTTP/HTTPS, FTP, news and email. An extension to peer-to-peer applications is work-in-progress. Therefore, our CTR estimates at this point cover only 30%-80% of the traffic, depending on the measured link. Measurements at a dozen of Internet links show that the CTR is usually between 60-80%. Such high CTR values suggest that a strong reason behind the congestion responsiveness of Internet traffic is that users and applications respond to congestion by slowing down the generation of new flows/sessions. TCP's congestion control and capacity overprovisioning are not the only reasons we do not see significant congestion events in most of the Internet today.

2 Congestion Responsiveness

Traditionally, congestion control has been viewed as a function of the network or transport layer at the OSI stack. The *TCP feedback loop* regulates the offered load (send-window) of a connection, based on the presence of congestion in the network (see Figure 1). The previous view, however, ignores the fact that TCP connections are the result of user and application actions. For example, the TCP connections generated from downloading a Web page, which constitute a "Web session", are the result of a user entering a URL at a web browser or clicking on a link[1]. Such connection arrivals can be consistent with an open-loop model, i.e., users generate new sessions, independent of what happened to their previous sessions and whether the network is congested or not. In other words, even though the transport layer provides congestion responsiveness through TCP, the session layer can be completely unresponsive if it keeps generating new sessions even when the network is congested.

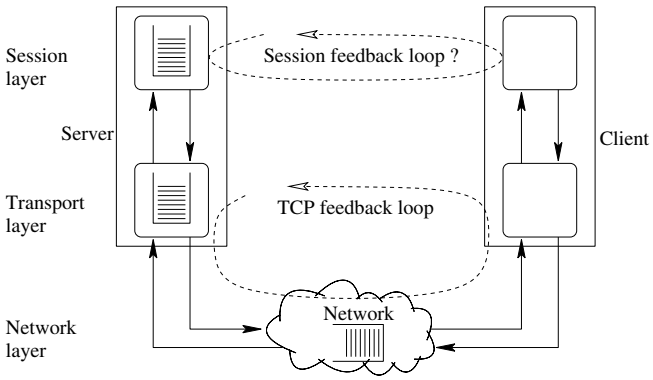

Fig. 1. The TCP feedback loop at the transport layer cannot avoid persistent overload if there is no session layer congestion control

We do not claim that TCP congestion control is not necessary. It is not sufficient, however, to avoid persistent overload. To understand this point, consider the previous example of an open-loop session layer. When the network becomes congested, each active TCP connection backs-off either reducing its send-window by a large factor or getting into a relatively long silence period (retransmission timeout). This means that congestion control pushes the offered load from each connection back to the TCP buffer of the sender. That connection is still active, however, and so it will keep trying to retransmit any lost packets and to increase its window. As the session layer keeps generating new transfers, the number of competing flows will increase, leading to a diminishing per-session goodput.

[1] A single click from a user can create more than one connection to download the embedded objects in a web-page. These connections together constitute one *session* and they can have different source but the same destination IP address.

TCP cannot avoid the emerging persistent overload. Instead, we need a way to tell the session (or application) layer to slow down or stop generating new flows for a while, thus forming a closed-loop system where the arrival of new flows is delayed upon congestion. Presently, this session-layer feedback is sometimes built in the application, or it simply results from the way people react when their applications are slow.

3 CTR Measurements

In this section, we propose a methodology to estimate the Closed-loop Traffic Ratio. The outline of the CTR estimation methodology is as follows. First, we partition the packet trace into a set of sessions initiated by each user. This process is far from simple, and it requires some knowledge of the corresponding application. For example, in the case of HTTP downloads, a "user" is associated with a specific IP destination address. Each session corresponds to a download operation requested by a user and it can consist of multiple "transfers". After we have transformed the packet trace into a "session trace", we then classify each session as open-loop or closed-loop. We assume that the dependency between session arrivals from the same user is related to the arrival time of the new session with respect to the finish time of the previous session from that user. Specifically, a session from a user is considered dependent on her previous session, and is classified as closed-loop, if it starts soon after the completion of the previous session. If the next session from a user starts while the previous session is still active, or after a long time from the completion of the previous session, then we view that new session as independent of the previous session and classify it as open-loop.

3.1 Definitions and Methodology

We start with a more detailed explanation of the key terms and of the CTR estimation methodology for the case of HTTP/HTTPS downloads. HTTP is not the only protocol/application for which we perform the following analysis. Traffic with other well-known ports is also well understood, in terms of who is the "user", what constitutes a "session", etc, and so we also apply the same methodology for those applications. Unfortunately, a large part of Internet traffic today does not use well-known ports. That traffic is probably generated by peer-to-peer applications. Eventually, we were able to analyze more than 50% of the TCP traffic in half of the traces we analyzed. In some traces the fraction of traffic we could analyze was as high as 78%, but in others was as low as 26%.

Users and sessions: *User* is an entity (typically a person, but it can also be an automated process) that issues Web requests. Each such application-layer request is a *session*. We assume that a user is identified in the packet trace by a destination IP address. An important exception to this rule is when multiple users share the same host (e.g., remote login) or when the addresses of different users are translated somewhere in the network to the same address (e.g., NATs

or proxies). We have devised a heuristic that can identify and ignore multi-user hosts, described in the Appendix.

In HTTP/HTTPS, a user is associated with a destination address, because users typically download traffic. In applications that mostly upload traffic to remote hosts, the user is associated with a source address. A download session, associated with a certain user, can contact a number of different servers, and it can consist of several TCP connections with different destination ports. Consequently, the traffic that belongs to a certain session would have the same IP destination address, but potentially different source addresses and/or destination ports.

Connections and transfers: A TCP connection is identified in the packet trace by a unique 5-tuple field (Source IP address, Destination IP address, Source Port, Destination Port and Protocol). A connection has a start and a finish time, corresponding to the timestamps of the first and last packets in the connection, respectively. We ignore connections that were ongoing at the start or end of the trace. Pure ACKs are packets without payload. A connection with more pure ACKs than data segments is considered an *ACK flow* and it is ignored from the analysis.

HTTP 1.1 introduced persistent connections, meaning that a connection can stay alive for a long time, transferring Web objects that belong to different sessions. We partition a connection into one or more *transfers*, with different transfers being part of different sessions. Figure 2 shows an example of a persistent connection that includes two transfers. The packet interarrivals within the same

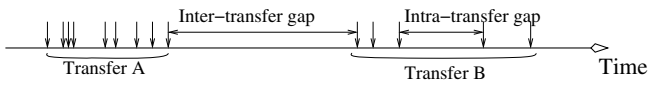

Fig. 2. Packet interarrivals from two transfers within the same connection

transfer are determined by TCP (e.g., self-clocking, retransmission timeouts) or network delays. The packet interarrivals between different transfers, however, are typically determined by the latency of user actions (e.g., clicking at a Web link). Consequently, the inter-transfer interarrivals ("gaps") are usually much longer than the intra-transfer interarrivals. We use this observation to partition a connection into transfers. If a packet interarrival within the same connection is larger than a certain *Silence Threshold* (STH), which represents the maximum intra-transfer gap, then a new transfer starts with that packet. To choose a reasonable value for STH, we examined values in the range 1sec-1min. A very small STH would partition a transfer in smaller chunks, while a very large STH would merge different transfers together. So, we expect that the average transfer size increases with STH. More importantly, we expect that for a certain range of this threshold, when it falls between the larger intra-transfer gaps and the lower inter-transfer gaps, the number of transfers is almost constant. The distribution of transfer sizes as a function of STH for a Georgia Tech inbound packet trace

shows that the median transfer size is roughly constant when the threshold is between 35-45sec. In the following, we set STH to 40sec.

Grouping transfers into sessions: Since a session can consist of several transfers, as shown in Figure 3, we need to identify the transfers (TCP connections or segments of TCP connections) that were generated as a result of the same user action. The key observation here is that the interarrival of two transfers that belong to the same session will typically be much lower than the interarrival of transfers that belong to two different sessions. The latter are separated by the latency of a user action. We expect transfers of different sessions to start at least one second or so from each other, while transfers that belong to the same session are typically generated automatically by the Web browser within tens or hundreds of milliseconds.

Specifically, if the interarrival between two successive transfers is larger than a certain parameter, referred to as *Minimum Session Interarrival* (MSI), then the second transfer starts a new session. We examine the robustness of the CTR estimate to the exact choice of the MSI in Section 3.2.

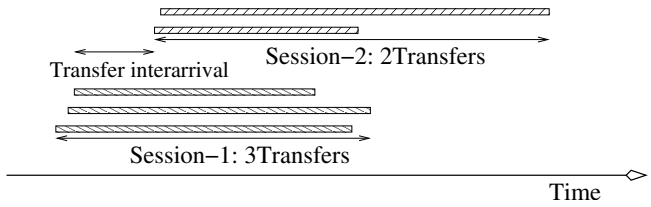

Fig. 3. Timeline of two sessions. Each session consists of several transfers.

Classifying sessions as open-loop or closed-loop: After having transformed the packet trace into a "session trace", we classify each session as open-loop or closed-loop. Recall that the key difference between these two is that in the open-loop model sessions arrive independent of the progress of previous sessions from the same user.

The first session from a user is considered open-loop, given that that session has no dependencies to previous sessions. If that user generates a new session after her previous session finishes and no later than the *Maximum Think Time* (MTT), then the arrival of this new session is considered dependent on the progress of the previous session. So, we classify that session as closed-loop. If, however, the new session arrived during her previous session or much later, more than MTT, we assume that the user either does not wait for her previous session's completion, or she was inactive for some time and now returns to the network without any "memory" of previous sessions. Thus, we classify that session as open-loop. These different cases are shown in Figure 4. We examine the robustness of the CTR estimate to the exact choice of the MTT in Section 3.2.

Other traffic with well-known ports: For other traffic, not generated by HTTP/HTTPS, we followed the convention that the transfer is an upload if the

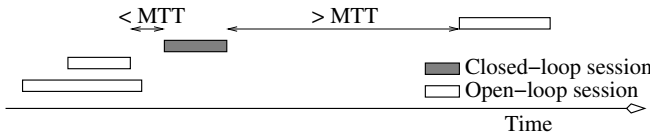

Fig. 4. Classification of different sessions from the same user as open-loop or closed-loop

destination port is well-known, and a download if the source port is well-known. The rest of the estimation methodology is the same as in Web traffic.

CTR calculation: Once we have classified sessions as open-loop or closed-loop, we then calculate the CTR as the fraction of bytes from closed-loop sessions. If the CTR is close to zero we expect that most traffic is congestion unresponsive (open-loop model), while if the CTR is close to one we expect that most traffic would reduce its flow arrival rate upon congestion (closed-loop model).

Limitations: The main assumption in the previous methodology is that the timing between the arrivals of successive sessions from the same user can reveal whether sessions arrive independently of each other. This is not always the case of course. For example, a user can obtain a URL through an ongoing web-page download (session X) and then start downloading that URL in another window (session Y), while session X is still active. In this case, the arrival of session Y depends on the progress of session X. However, session Y starts before X completes, and so we would classify the former as open-loop. On the other hand, in a fast and uncongested network, even though a user can start a sequence of independent sessions, some of the sessions can complete before a subsequent session starts. In this case, the latter will be incorrectly classified as a closed-loop session. We expect that such "deviations" are not common compared to the more common Web browsing behavior, which is to first download a page (open-loop session arrival), spend some time reading or viewing it, and then either download another page (closed-loop session arrival) or leave the system for a while.

3.2 Robustness of Estimation Methodology

The CTR estimate depends on the following parameters: STH, MSI, and MTT. We have already mentioned that a robust value for STH is around 40sec. In this section, we investigate the optimal range for MSI and MTT, and examine the CTR's robustness to the choice of these two parameters.

The MSI is used to merge together different transfers of the same session. As these transfers are typically machine-generated, they start almost simultaneously. In the packet trace, however, they can appear with slightly longer spacings due to network delays. A reasonable range for this parameter is between 0.5-1sec. The MTT, on the other hand, represents the longest "thinking" time for a user during Web browsing, before we assume that that user left the system. A reasonable range for this parameter is between 5-30min.

We examined the CTR variations for a Georgia Tech trace with different MSI and MTT values. The main finding is that the CTR does not significantly depend on these parameters (it varies in a small range between 0.6-0.72) as long as the MSI and the MTT fall between 0.5-2sec and 5-25min, respectively. To further examine the robustness of CTR against these two thresholds, we also performed two-factor analysis of variance. The null hypothesis is that the CTR is independent of these two parameters. We can reject this hypothesis at a significance level of 0.05. However, the hypothesis that the CTR is independent of the interaction of these two parameters cannot be rejected at the same significance level. We found that the slope of the CTR with respect to MSI is 0.0232/sec, and with respect to MTT it is 0.0020/min. Therefore, we concluded that the CTR is practically insensitive to these parameters in the ranges that we consider. In the rest of the analysis, we set MSI=1sec and MTT=15min.

Table 1. CTR of the analyzed TCP traffic at various links

					TCP traffic		
					Total	Well-known ports	
Trace_ID	Direction	Collection time	Link location	Duration	GB (%)	bytes	CTR
GaTech-in	In	07-Jan-05	GaTech	2Hr.	129.74 (97.15)	63.50%	0.71
GaTech-out	Out	07-Jan-05	GaTech	2Hr.	208.06 (98.92)	47.90%	0.57
Los-Nettos	Core	03-Feb-04	Los-Nettos, CA	1Hr.	59.37 (94.96)	65.59%	0.77
UNC_em0	Out	29-Apr-03	UNC	1Hr.	153.19 (97.33)	35.78%	0.61
UNC_em1	In	29-Apr-03	UNC	1Hr.	41.51 (87.76)	26.59%	0.76
UNC_em1_2	In	24-Apr-03	UNC	1Hr.	55.25 (85.67)	44.88%	0.78
MFN_0	Core	14-Aug-02	MFN, San Jose	1Hr.	151.38 (96.31)	61.09%	0.69
MFN_1	Core	14-Aug-02	MFN, San Jose	1Hr.	186.93 (97.75)	71.83%	0.62
IPLS_0	Core	14-Aug-02	Abilene	1Hr.	172.22 (96.40)	41.93%	0.70
IPLS_1	Core	14-Aug-02	Abilene	1Hr.	177.99 (85.00)	47.27%	0.64
Auckland_0	In	06-Nov-01	Auckland, NZ	6Hr.	0.58 (94.83)	72.99%	0.73
Auckland_1	Out	06-Nov-01	Auckland, NZ	6Hr.	1.44 (98.43)	77.99%	0.67

3.3 Results

Table 1 summarizes the metadata of the traces we analyzed in this paper. The traces are from university access links, commercial access links and backbone links, and they were collected between 2001-2005. It also shows the CTR estimates for Web and other well-known port traffic in these 12 Internet traces.

An important observation is that the CTR for access links is always higher in the inbound direction than in the outbound direction. This can be explained by the fact that users that initiate sessions in the inbound direction belong to the limited population of users within that campus network. On the other hand, users that initiate sessions in the outbound direction come from all over the Internet and they belong to a much larger population. Consequently, the fraction of open-loop traffic in the latter is higher (lower CTR).

The most important observation, however, is that the CTR for almost all traces is high, typically between 60-80%. Even the backbone links, where we would expect more open-loop traffic due to the large number of users, have a high CTR. This suggests that a major reason for the congestion responsiveness of Internet

traffic may be that most applications follow the closed-loop model, and so they are responsive to congestion at the session generation layer.

4 Related Work

The notion of congestion responsiveness is related to the issue of TCP-friendliness [2]. The latter, however, focuses on non-TCP traffic. Previous studies with open-loop flow arrivals concluded that admission control is necessary to avoid persistent overload [3,4]. Berger and Kogan [5], as well as Bonald et al. [6], used a closed-loop model to design capacity provisioning rules for meeting throughput-related QoS objectives. Heyman et al. [7] used a closed-loop model to analyze the performance of Web-like traffic over TCP, showing that the session goodput and fraction of time the system has a given number of active sessions are insensitive to the distribution of session sizes and think times. In a recent paper, Schroeder et al. [8] compare open-loop and closed-loop arrival models and highlight their differences in terms of mean job completion time and response to different scheduling policies.

Understanding user and application behavior, as well as their impact on network traffic, has been the subject of some previous work [9,10,11,12]. These papers focus on HTTP, mostly because the application characteristics of Web browsing are well understood. Casilari et al. [9] present models for different levels of HTTP analysis, namely at the packet, connection, page or session levels. Smith et al. attempt to reconstruct HTTP sessions from unidirectional TCP header traces [11]. Feldmann [10] correlates bidirectional IP traces with HTTP header traces. Tran et al. [12] use payload information to identify related HTTP requests/responses and to study the effect of congestion on user behavior (probability of aborted and retried sessions).

5 Discussion and Future Work

This paper focused on a simple question: how does Internet traffic react to congestion? This is an important question towards a better understanding of the Internet traffic characteristics. Additionally, the issue of congestion responsiveness has several practical implications. To avoid overload conditions, applications need to be "congestion-aware" at the session layer. One way to do so is to delay generating a new session to a server if the previous has not yet completed or if it is stalling. If the traffic at a certain link is mostly open-loop, then the operator has two options. One is to increase the link capacity above the given offered load. The second is to do some sort of admission control at that link. With closed-loop traffic, on the other hand, these two options may not be necessary because the offered load is self-regulating. We are currently extending this work to estimate the CTR for peer-to-peer traffic, which often accounts for most of the traffic today.

References

1. Yang, S.C., de Veciana, G.: Bandwidth Sharing: The Role of User Impatience. In: Proceedings IEEE GLOBECOM. (2001) 2258–2262
2. Floyd, S., Handley, M., Padhye, J., Widmer, J.: Equation-Based Congestion Control for Unicast Applications. In: SIGCOMM. (2000)
3. de Veciana, G., Lee, T., Konstantopoulos, T.: Stabily and Performance Analysis of Networks Supporting Services. IEEE/ACM Trans. on Networking **9** (2001)
4. Fredj, S.B., Bonald, T., Proutiere, A., Regnie, G., Roberts, J.W.: Statistical Bandwidth Sharing: A Study of Congestion at Flow Level. In: Proceedings of ACM SIGCOMM. (2001)
5. Berger, A., Kogan, Y.: Dimensioning Bandwidth for Elastic Traffic in High-Speed Data Networks. IEEE/ACM Transactions on Networking **8** (2000) 643–654
6. Bonald, T., Olivier, P., Roberts, J.: Dimensioning High Speed IP Access Networks. In: 18th International Teletraffic Congress. (2003)
7. Heyman, D., T.V.Lakshman, Neidhardt, A.L.: A New Method for Analysis Feedback-Based Protocols with Applications to Engineering Web Traffic over the Internet. In: ACM SIGMETRICS. (1997)
8. Schroeder, B., Wierman, A., Harchol-Balter, M.: Closed Versus Open System Models and their Impact on Performance and Scheduling. In: Symposium on Networked Systems Design and Implementation (NSDI). (2006)
9. Casilari, E., Reyes-Lecuona, A., Diaz-Estella, A., Sandoval, F.: Characterization of Web Traffic. In: IEEE Globecom. (2001)
10. Feldmann, A.: BLT: Bi-Layer Tracing of HTTP and TCP/IP. Computer networks **33** (2000) 321–335
11. Smith, F., Campos, F., Jeffay, K., Ott, D.: What TCP/IP protocol headers can tell us about the Web. In: ACM SIGMETRICS. (2001)
12. Tran, D.N., Ooi, W.T., Tay, Y.C.: SAX: A Tool for Studying Congestion-Induced Surfer Behavior. In: PAM. (2006)

Appendix: Multi-user Host Detection

We describe a heuristic to distinguish between single-user and multi-user hosts (such as NATs, proxies, rlogin servers etc). A host is identified by a unique IP address in the packet trace. In a multi-user host, sessions generated by different users share the same destination address, making it impossible to distinguish sessions from different users. In multi-user hosts, however, the number of transfers per session would typically be much larger than in single-user hosts. This large difference is the key criterion to detect multi-user hosts. For instance, in one of the Georgia Tech inbound packet traces about 99% of hosts have less than 20 transfers per session. However, there are also 100 hosts (<1%) that generate up to a few thousands of transfers per session; it is likely that they are multi-user hosts. We examined the DNS names of those hosts, and several of them indicate proxies and firewalls. So we chose a threshold of 10 transfers per session, on the average, to distinguish single-user from multi-user hosts. It turns out that the final CTR estimate (which is what we care about) is robust to the previous threshold, as long as the latter is more than 5-6 and less than about 100 sessions.