Transport Working Group
Internet Draft
Intended status: Informational
Expires: August 2010

R. Penno
S. Raghunath
Juniper Networks
R. Woundy
Comcast
V. Gurbani
Bell Labs, Alcatel-Lucent
J. Touch
USC/ISI
February 26, 2010

LEDBAT Practices and Recommendations for Managing Multiple Concurrent TCP Connections
draft-ietf-ledbat-practices-recommendations-00.txt

Status of this Memo

Copyright Notice

Abstract

Applications routinely open multiple TCP connections. For example, P2P applications maintain connections to a number of different peers and web browsers perform concurrent download from the same web server. Application designers pursue different goals when doing so: P2P apps need to maintain a well-connected mesh in the swarm while web browsers mainly use multiple connections to parallelize requests that involve application latency on the web server side. However this practice also has impacts to the host and the network as a whole. For example, an application can obtain a larger fraction of the bottleneck than if it had used fewer connections. Although capacity is the most commonly considered bottleneck resource, middlebox state table entries are also an important resource for an end system communication.

This document clarifies the current practices of application design involving concurrent TCP connections and reasons behind them, and discusses the tradeoffs surrounding their use, whether to one destination or to different destinations. Other resource types may exist, and the guidelines are expected to comprehensively discuss them.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119.

Table of Contents

1. Introduction

The use of P2P protocols by end users is widespread. These protocols are meant to exchange, replicate, stream or download files with little human intervention, trying at the same time to minimize the download time of the files requested by any single peer. This is done by opening several connections to multiple peers and downloading one or more chunks of the file from each one, selecting faster peers, amongst others.

If we assume that in any file transfer the bottleneck is on the uploading peer or server side, end users that utilize P2P clients in general download the file faster and consume more bandwidth within a specific timeframe than traditional client-server applications. P2P clients can overcome the server side bottleneck by opening multiple connections to different peers. Users of P2P applications also consume bandwidth throughout the whole day since even after a file is fully downloaded it will continue to be shared with others users increasing the upstream bandwidth.

We can see then that the advantages of P2P applications come from the fact that they open multiple TCP connection to different peers in

order to download multiple pieces of a file in parallel, and that they always look for faster peers.

But the use of multiple TCP connections by an application is not new. Web Browsers have been doing it for a decade. But these are usually short-lived connections as opposed to long-lived connections. A long-lived connection in this document should be interpreted as strictly defined, i.e., a TCP connection that is simply in the established state, but not necessarily continuously transferring data. In the case of P2P protocols, e.g. BitTorrent, at any point in time only a fraction of these connections is actually sending or receiving data, while the others are idle or exchange occasional control information.

With the popularity of P2P applications, which maintain hundreds of long-lived TCP connections to multiple hosts, the issue of applications making use of multiple TCP connections has been gaining attention.

This document clarifies the current practices of application design and reasons behind them, and discusses the tradeoffs surrounding the use of many concurrent TCP connections to one destination and/or to different destinations. Other resource types may exist, and the guidelines are expected to comprehensively discuss them.

2. Terminology

Bandwidth: A measure of the amount of data that can be transferred within a time period, often expressed in bits per second. Bit rate prefixes are expressed in decimal, so 1 kilobit per second is 1,000 bits per second, and 1 megabit per second is 1,000,000 bits per second. So, if one million bits are transferred within one second, the average bandwidth consumption during the transfer would be 1 megabit per second (1 Mbps). If the same amount of data were transferred within a day, the bandwidth would be approximately 11.574 bits per second.

Volume: The total number of bytes (or octets) transferred during a time period. Byte prefixes are expressed in binary, so 1 kilobyte is 1,024 bytes, and 1 megabyte is 1,024 * 1,024 = 1048576 bytes. In both examples above the volume within a day would have been 125,000 bytes or about 122.07 kilobytes (122.07 KB).

Capacity: The maximum bandwidth a link can sustain continuously.

Long-lived connection: A TCP connection that is in the established state but not necessarily continuously transferring data.

3. Multiple control versus data connections

The traditional model of applications interacting with each other using TCP started off as a single socket opened between a client and a server for data communications. Control signaling was usually passed on the same channel as well. Telnet [RFC854] and rlogin protocols [RFC1282] are good examples of this approach. File Transfer Protocol [RFc959] was one of the first known protocols that used more than one connection between a client and a server. In FTP, the client in the normal client-server fashion opens the control connection. This connection is used for commands from the client to the server and replies from the server to the client. Distinguishing FTP from other protocols was its use of a second data connection. The client initiates this data connection passively, and the port number is sent to the server. The server subsequently establishes an active connection to this port. A data connection is created each time a file is transferred between the client and the server. However, unlike the control connection, it does not persist for the duration of the FTP session.

These early protocols limited TCP connections between a pair of machines. This changed with the advent of the Hypertext Transfer Protocol [RFC2616]. In HTTP, a client (browser) downloads a document from a server and analyses it to render the document on a display device of some sort. As part of the analysis, the browser may open one or more connections to either the same host from which the original document was downloaded, or to different hosts that serve other content referenced in the document. However, these connections were usually short lived (the current phenomenon of "long polling" notwithstanding). Here, the client (browser) opens up multiple TCP connections to possibly multiple servers simultaneously. The Session Initiation Protocol [RFC3261] can use TCP connection in the same vein as HTTP did, namely to contact multiple servers simultaneously. Generally -- although there are exceptions -- in SIP just like HTTP, these connections are typically short-lived.

More recent protocols like Skype (http://www.skype.com) and BitTorrent (http://www.bittorrent.com) have a much different view on the number of TCP connections they are willing to open and manage. While earlier protocols were parsimonious with connections, the modern peer-to-peer protocols do not appear to be wary of this to the same degree. Part of the reason why this is the case is the assumption that the older protocols (Telnet, rlogin, HTTP) were operating under was that relatively few bytes will be transferred from the client to the server while many more bytes will be

transferred in the opposite direction. With current peer-to-peer protocols, where the resource to be accessed is distributed among the peers, a requesting peer has to open multiple TCP connections to more than one peer in order to efficiently download the data represented by that resource.

In summary, trying to establish a boundary between data connections and control connections is something of a fool's errand. Protocols evolve to match the capabilities and characteristics of the network. While early protocols may have opened up a pair of connections to communicate, more recent protocols are not inhibited in the same manner.

Similarly, while earlier protocols may have established different control channel from a data channel, this was not a design rule that was carried forward faithfully. While SIP falls in the former camp of a control channel that is distinct from a data channel, HTTP falls in the latter camp (i.e., same TCP connection serves to send control messages and the data itself.) BitTorrent and Skype perform control and data communications over the very same TCP connection as well. BitTorrent, in particular, attempts to open multiple connections to many peers, even though only a small subset of these connections are involved in the actual data transfer. In Skype, a peer does not open multiple connections to access a resource; rather multiple connections are opened and maintained to a discrete set of neighbors to help in routing of subsequent messages [Skype-analysis].

4. Multiple TCP Connections Advantages

There are good reasons for an application to use multiple TCP connections. P2P apps need to maintain a well-connected mesh in the swarm while web browsers mainly use multiple connections to parallelize requests that involve application latency on the web server side

But from a P2P standpoint multiple TCP connections are at the heart of its functionality. Multiple connections allow for multiple simultaneous downloads, which improve reliability and speed. Multiple connections also allow more effective discovery of new peers, and effective peer-to-peer communication, which allows exchange of information such as which pieces of a file a client has and is available.

4.1. Avoiding head-of-line blocking

Web browsers started using multiple TCP connections partly because of this reason [STEVENS]. This is especially true when the multiple TCP

connections are between a pair of hosts. Originally, individual HTTP transactions each used their own TCP connection, because HTTP lacked a response length marker. The client sent a request to the server, and the server's response to the client was completed when the TCP connection was closed, i.e., CLOSE was interpreted as ''end of transaction''. This caused numerous problems, notably the buildup of connections in the TIME-WAIT state at the server [Fab1999]. HTTP added persistent connections to v1.0 (they were later included in the 1.1 spec), and they became the default. In persistent connections, transactions complete but the connection remains open for subsequent responses. Responses are pipelined, not interleaved, however, resulting in Head-of-Line (HOL) blocking.

HTTP clients currently open 4-8 connections to each endpoint. This partly avoids HOL blocking, but also allows increased performance. Separate connections open independently, increasing bandwidth, and also can use separate endpoint processes, increasing computational parallelism that maps more effectively to multiprocessors and multi-core systems.

4.2. Logical partitioning at application level

Some applications such as FTP use a separate connection for control and data transfers. The advantage is that this allows a model where the data transfer is actually happening between hosts that are not local (see [RFC959], sections 2.3 & 5.2).

4.3. Multiple streams with different properties

The application may need different properties on multiple streams of data (e.g., Nagle's algorithm, socket buffer sizes etc).

4.4. Signaling application layer request completion

If the application assumes that connection close indicates the completion of a request, it becomes necessary to have new connections for multiple requests. This was a reason for multiple connections in HTTP 1.0.

4.5. High bandwidth-delay links

In the presence of a large bandwidth-delay product, the 16-bit window size parameter in TCP header does not allow the application to fully utilize the link. In such situations, the current practice is to negotiate the Window Scale Option [RFC1323]. In addition multiple TCP connections can allow the application to achieve an effectively larger window size so that it can better utilize a link with high

bandwidth-delay product (e.g. iSCSI [SCSIREF]), although this can result in mutual escalation, where TCP fairness is ensured only for endpoints opening multiple connections.

4.6. Error resiliency and reliability

When multiple connections are used to download a single file or webpage, for instance, there is lesser chance of a single failure on one connection having a negative impact on the whole download. Especially with P2P applications, this makes the network robust to failures and churn in participants.

4.7. Leveraging multiple processors in a system

With multiple processor systems, there can be higher performance with parallelism and multiple connections spread over different processors.

This presumes that the kernel is parallelized; the potential for TCP parallelism is limited (http://www.isi.edu/touch/pubs/pfhsn94.html)

4.8. Overcoming TCP Limitations

The performance of a single TCP connection given a certain link is well understood today [PARATCPSCK], [FF99], [PFTK98] and the general rule of thumb is that a TCP connection can utilize 75% of its capacity. This means more than one connection to one or more servers would be needed to saturate the link.

5. Multiple TCP connections Disadvantages

Every connected application on the Internet competes for resources. This is not specific to applications that open multiple TCP connections. The use of multiple TCP connections just amplifies the issue. In the following sections we discuss these resources and how they are amplified by an application opening multiple connections.

5.1. Additional connection setup overhead

The TCP's mechanisms for starting up the connection and then probing the available bandwidth have to be repeated for each new connection. So there may be lesser leverage of network information. There is also the overhead of additional control traffic that may have been avoided.

5.2. Memory Space

Each TCP connection needs a TCP control block (TCB) or equivalent to keep state about its connection. In operating systems where the TCP stack is part of the kernel, this would come from the kernel memory space, otherwise from userland memory.

But irrespective of where the memory comes from a TCP control block requires a significant amount of memory. This is significant issue for devices that terminate TCP connections from multiple end hosts to provide functions such as Load-Balancing, Gateway and Tunneling.

Some proposals have been put forward to reduce the amount of memory occupied by each TCP control block [RFC2140], but the issue remains significant and is amplified by applications that use multiple TCP connections.

5.3. Link Bandwidth

The bottlenecks for these N multiple connections could be shared or separate. If separate, there's no specific bottleneck where the connections are hogging bandwidth. But from a network resource point of view, the application download still gets multiple shares.

If some/all bottlenecks are shared, then two possibilities exist for shared bottleneck:

o the bottleneck is a last-hop link (user traffic dominates link), OR

o the bottleneck is an in-network wide-area link (background traffic dominates link)

If bottleneck is the last-hop, then n transport connections compete with each other and share link bandwidth.

Although these connections might impact delay-sensitive traffic and increase delay, in the last hop they only affect the end-user, which is in control of which applications run on its host. In this case the user has the option of manually choosing when to run each application, configuring the end host, amongst other choices. Alternatively, or in conjunction with the above, the application can be enhanced to use Diffserv and new delay sensitive congestion mechanisms.

If the shared bottleneck is in-network, then the application gets an unfair share of bottleneck bandwidth. This impacts flows belonging to

other users in general, and most importantly may impact delay-sensitive traffic.

5.4. Middleboxes

Middleboxes are defined as any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host [RFC3234]. Middleboxes can be stand-alone or integrated in another device such as a router or modem.

The functions that are relevant to this discussion are those that require the middlebox to keep per session state, sometimes referred as transformation services. Some of these functions are, for example, NAT, Intrusion Detection and Load-Balancing.

It is easy to see that the more sessions a host initiates, the more state the middlebox will have to keep. The relationship is at least 1:1 but due asymmetric traffic, routing changes and other considerations, this can be 1:N.

Although application traffic from most broadband subscribers today go through at least one middlebox (as a stand-alone device in the home network, or integrated into the broadband modem), it can traverse other middleboxes that reside within the ISP's network or close to the destination. These middleboxes aggregate traffic from multiple subscribers, and state tables within these devices can become a premium.

6. Conclusion and Recommendations

6.1. Diffserv

REC-1: Applications involved in bulk data transfer with low priority in time could mark their packets according with the guidelines of RFC 3662 [RFC3662].

6.2. Window scale negotiation

REC-2: Where appropriate, sender & receiver window should be scaled using RFC1323 based negotiation in order to make the best use of network resources. Recommendations to adjust window size are not new and have been recommended in networks where the BDP (Bandwidth Delay Product) is large [RFC3481].

6.3. Number of Connections

Multiple connections to the same or different servers provide a significant speedup as compared to a single connection. The motivation to use multiple connections is to achieve throughput efficiency and the cause for such deficiency could be head of line blocking, slow servers, server availability or simply overcoming TCP throughput limitations. [DYNPARACON],[PARATCPSCK].

In the case of multiple parallel connection homogeneous connection sharing a link of capacity c, it was found that 6 connections are sufficient to reach 95% download utilization [PARATCPSCK]. Interestingly this is comparable to the number of configured active transfers (5) of the most used BitTorrent client [UTORRENT]. It is worth noticing that during a large file transfer BitTorrent clients will prefer peers that provide the largest upload rate, thus theoretically saturating its download link. In reality, packet drops, upload caps and others transient effects would require clients to have more than 5 connections in order to saturate the link, but the overall effect of these issues in terms of bandwidth decrease is something already measured by BitTorrent clients and used in the (optimistic) unchoke/choke algorithm. Therefore the number of active connections should not be much higher than 5 since the idea is to saturate the link by choosing the best connections and not necessarily more connections.

REC-3: Applications should only open more than 6 connections to download the same object if the first hop link is not saturated.

6.3.1. HTTP

The case of web browsing (HTTP) is quite different from P2P. One could argue that the number of active connections used by HTTP is much higher than that used by BitTorrent, but the scenarios are quite different. In the case of dynamic pages, different objects are downloaded from (and exclusively available from) certain locations. Moreover, time is of the essence since there is an expectation that a page is downloaded and rendered within a few seconds. Finally, objects in a webpage are quite small, with the majority (75%) below 6KB [HTTPDATA]; therefore many connections are needed to saturate the link since TCP congestion avoidance never has time to ramp up to its maximum bandwidth. If multiple small HTTP objects can be retrieved from the same server, the use of HTTP/1.1 Pipelining is recommended since it can dramatically reduce the number of packets and connection overhead between client and server [HTTPPERF].

REC-4: HTTP based applications should use HTTP/1.1 pipelining when transferring multiple small objects from the same server.

6.4. Bi-Directional HTTP

Recent frameworks like Ajax allow application developers to write applications that allow a delay between when the HTTP server receives a request and sends the corresponding response a response. This technique, called "long polling", works by having the HTTP server delay sending the response to a request back until it has some additional data to sent to the client. HTTP streaming is a technique where the server keeps the connection open indefinitely by using chunked Transfer-Encoding mechanism to send incremental responses spread over time.

Both these techniques originated as a counter mechanism to the normal manner of polling for events in HTTP: sending multiple requests where the inter-request frequency is fairly small. Such a polling mechanism tends to overwhelm the server if the polling frequency is set too low.

Both long polling and HTTP streaming affect the number of TCP connections open over a period of time and the network in the following way:

- o Reducing the overhead of opening/closing connections

- o Increasing memory consumption in both clients and servers

7. Security Considerations

None at this time

8. IANA Considerations

None at this time

9. Acknowledgments

J. Iyengar was one of the presenters on the first BOF and worked on the original version of this document.

10. References

10.1. Normative References

[RFC959] J. Postel and J. Reynolds, "File Transfer Protocol (FTP)", RFC 959 (1985).

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, December 1998.

10.2. Informative References

[RFC1323] V. Jacobson, B. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, May 1992

[RFC2140] J. Touch, TCP Control Block Interdependence, RFC 2140 (1997)

[RFC2616] R. Fielding et al, Hypertext Transfer Protocol HTTP/1.1, RFC 2616 (1999)

[RFC3481] Inamura, H., Montenegro, G., Ludwig, R., Gurtov, A., and F. Khafizov, "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks", BCP 71, RFC 3481, February 2003.

[RFC3234] Carpenter, B. and S. Brim, "Middleboxes: Taxonomy and Issues", RFC 3234, February 2002.

[RFC3662] Bless, R., Nichols, K., and K. Wehrle, "A Lower Effort Per-Domain Behavior (PDB) for Differentiated Services", RFC 3662, December 2003.

[SCSIREF] K.Z. Meth, J. Satran, Design of the iSCSI protocol, Storage Conference (2003)

[STEVENS] W. Richard Stevens et al, ''Unix Network Programming, The Sockets Networking API'', Volume 1, Third Edition (2003), section 10.5, page 293.

[HOLBLCK] Head-of-line Blocking in TCP and SCTP: Analysis and Measurements

[EXPPARA] S. Philopoulos and M. Maheswaran. Experimental Study of Parallel Downloading Schemes for Internet Mirror Sites. In Thirteenth IASTED International Conference on Parallel and Distributed Computing Systems (PDCS '01), Aug. 2001.

[HTTPPERF] Network Performance Effects of HTTP/1.1, CSS1, and PNG. http://www.w3.org/Protocols/HTTP/Performance/Pipeline

[DYNPARACON] P. Rodriguez and E. W. Biersack, ''Dynamic Parallel-Access to Replicated Content in the Internet'', IEEE/ACM Transactions on Networking, August 2002

[UTORRENT] http://www.utorrent.com

[PARATCPSCK] Altman, E., Barman, D., Tu.n, B., Vojnovic, M.Parallel TCP Sockets: Simple Model, Throughput and Validation. In: IEEE INFOCOM 2006, Barcelona, Spain (2006)

[HTTPDATA] Y. C. Chehadeh, A. Z. Hatahet, A. E. Agamy, M. A. Bamakhrama, and S. A. Banawan, "Investigating distribution of data of http traffic: An empirical study," in Innovations in Information Technology, 2006.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. ''Modeling TCP Throughput: A Simple Model and its Empirical Validation''. SIGCOMM Symposium on Communications Architectures and Protocols, Aug. 1998.

[FF99] S. Floyd and K. Fall. ''Promoting the Use of End-to-End Congestion Control in the Internet''. IEEE/ACM Transactions on Networking, Aug. 1999.

[Fab1999] Faber, T., Touch, J. and W. Yue, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proc. Infocom 1999 pp. 1573-1583.

[Skype-analysis] S. Baset and H. Schulzrinne, "An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol". IEEE Infocom", Apr. 2006

Author's Addresses

Reinaldo Penno
Juniper Networks
1194 N Mathilda Aveue
Sunnyvale, CA

Email: rpenno@juniper.net

Satish Raghunath
Juniper Networks
1194 N Mathilda Aveue
Sunnyvale, CA

Email: satishr@juniper.net

Vijay K. Gurbani
Bell Labs, Alcatel-Lucent
1960 Lucent Lane
Room 9C-533
Naperville, IL
60566
USA

Email: vkg@bell-labs.com

Richard Woundy
Comcast Cable Communications
27 Industrial Avenue
Chelmsford, MA 01824
US

Email: richard_woundy@cable.comcast.com
URI: http://www.comcast.com

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695
U.S.A.

Email: touch@isi.edu
URL: http://www.isi.edu/touch