Measuring Distributed Software Server Availability and Response Time via the Socket API and TCP Layers

Peter Clutterbuck and George Mohay
University of Queensland, Brisbane, Australia.
ISRC Queensland University of Technology, Brisbane, Australia.

Abstract

In a commercial Internet environment, the quality of service provided by service owner to service user is increasingly critical to the competitive advantage and survivability of both owner and user. The availability and response time of distributed software servers are central components of this overall quality of service framework. Availability is a measurement of server down time. Response time is a measurement of the time taken from when the service request is made, to when service provision occurs for the user. Deteriorating response time is also a valuable indicator to denial of service attacks. Denial of service attacks continue to pose a significant threat to server availability.

Modern high-performance software servers and user-clients have evolved considerably in efficiency and sophistication. Application designers increasingly look to make more use of operating systems to enable their server and client software to make higher quality and more broadly based performance decisions. General-purpose operating systems, however, provide inadequate support for the monitoring of server availability and response time on either a statistically averaged or individual service request basis. In this paper we propose that server availability measurement must include consideration of server response time. Server response time must be measured on a per-request basis, and must be made available to the prospective client at the earliest opportunity. We describe additions to the Linux Socket API and TCP that provide this important information to client and server.

Index Terms – Availability, Denial of Service, Reliability, Dependability, Response Time

1 Introduction

Distributed software servers are clearly a cornerstone of the ever-expanding use of network computing. Name, file, mail and web servers are clearly members of the set of software services considered essential for efficient network operation. The growing influence of electronic commerce in the Internet environment will quickly expand this base of critical software services. Total quality of service is an increasingly important issue to all service users and providers. It is essential that, within this total quality spectrum, each service quality descriptor has a meaning and measurement that consistently and accurately meet the needs of service users and providers.

Availability and reliability are important service quality descriptors of any hardware or software system [1]. Availability, defined and measured as the probability of service liveness, is a relevant and practical quality of service descriptor. All service stakeholders are concerned about user wait times that depend upon service liveness, service downtime, and service restart time. A legitimate question is, however, whether the stakeholders (ie, owners and users) of modern distributed services perceive this availability definition and measurement to be adequate. Do stakeholders describe a zero-downtime service as highly available even if service provision is extremely slow because of massive service overload or significantly under engineered network links? Are stakeholders satisfied with service availability
ratings that describe liveness probability over an arbitrary time period, or do they require some indicator of service availability on a per-request basis? Do users and stakeholders view service availability as a two element set \{available, not-available\}, or do they view and differentiate availability as many service-level points across a continuum?

This paper describes how pre-existing hardware availability measurement has heavily shaped the existing software availability theory. The paper proposes that service provision time, not just server liveness/uptime, is the basis of server availability, and consequently availability measurement must include server response time. The paper opens with a review of existing availability theory and denial of service. An extended availability definition is then developed. A discussion of implementation issues concludes the paper. This implementation discussion describes how the expanded definition is implemented within the Socket API and TCP within the Linux kernel.

2 Existing Theory

Several themes have relevance in shaping existing availability theory. Firstly, availability is closely linked with hardware reliability theory and consequently software availability management strategies are primarily based on countering any single point of failure within the software service. Secondly, denial of service attacks represent a significant threat to software service availability.

2.1 Availability, Reliability, and Management Approaches

A very early treatment of software availability theory is contained within [2], and more recently within [1] and [3]. The description within [2] begins by describing how software availability represents an application of hardware availability theory. Software availability is introduced as a system function to describe the beneficial features of repair in a system that tolerates shutdown times, caused by either planned or unplanned outages. The availability function, \(A(t)\), is defined as the probability that the system is operating at time \(t\). In contrast, the reliability function \(R(t)\) is the probability that the system has operated over the interval 0 to \(t\). The most important items for consideration are how frequently the system goes down and for how long it stays down. An important parameter, that characterizes this down time is the mean time to (or between) failures (MTTF).

Availability is defined in [1] as one of several reliability metrics. Informally, the reliability of a software system is a measure of how well users think it provides the services that they require. More formally, reliability is usually defined as the probability of failure-free operation for a specified time in a specified environment for a specified purpose. In the main, software reliability metrics have evolved from hardware reliability metrics. Availability takes into account the elapsed repair or restart time when a system failure occurs. If repair or restart time is brief, it is possible to have acceptable availability within a system that displays low reliability. Strategies for achieving high levels of availability are discussed in [3]. The concept of downtime is essential in defining and achieving availability. Downtime is described as ‘if a user cannot get his job done on time, the system is down.’ This strict definition is required because the system is provided for users to work in an efficient and timely way.

Availability management is primarily based on countering any single point of service failure via software replication. Replicated server availability has been described in [4], [5], [6], [7], [8], [9], and [10]. Replicated server availability, as it is presented in [5], assumes an asynchronous distributed system consisting of multiple host machines or nodes linked together into an arbitrary network topology. This service-hosting network in turn has some arbitrary link with the service user community. The primary entities within the model include the distributed service, a service group, an availability policy, and a management service. The model manages distributed service availability by ensuring that operational
primary and backup implementations exist within the distributed system at all times. The only threat to service availability is total, simultaneous failure of all nodes. The model makes no distinction between network level communication faults, hardware or operating system faults at a node level, or service implementation (process) failure.

2.2 Denial of Service

Denial of service attacks represent a significant threat to software service availability. Denial of service theory was initially developed in [11], and subsequently expanded in [12], [13], [14], [15], [16], [17], [18], and [19]. Denial of service attack strategies, and their impact, are discussed in [20].

Denial of service is defined in [13] as ‘a group of otherwise-authorized users of a specific service is said to deny service to another group of otherwise-authorized users if the former group makes a service unavailable to the latter group for a period of time that exceeds the intended (and advertised) waiting time.’ [20] describes the denial of service attack goal as being ‘completely deny service to legitimate users, networks, systems, or other resources.’ Much of the literature describes denial of service as a subset of availability theory. Indeed [14] uses denial of service concepts in describing availability as the ability to access a specific resource or service within a specific time frame as defined within the product specification. The specific time frame for access to the service can be defined in terms of a maximum waiting time (MWT) or finite waiting time (FWT).

Denial of service protection guarantees service access and service provision within acceptable, well defined and advertised wait times. [15] provides a resource allocation model for denial of service protection. In this model, service access and provision guarantees are managed by a Denial of Service Protection Base (DPB) that forms part of, or works closely with, the Trusted Computing Base (TCB) or host operating system. The DPB consists of waiting time policies, user agreements, an allocation system, and a resource allocation parameters that quantify the service access.

3 An Expanded View of Availability

What does availability mean to service user? A concise yet powerful user view is expressed in [21]. A system ‘that the service is there’ and ‘can get the job done’. A service user is also reflected in how [3] define downtime: ‘If a user cannot get his on time, the system is down’. Statements reflect a real world truth: service must operationally exist if rated by users as available. This characteristic (existence, or liveness, is the cornerstone of availability definitions presented in the previous section. The same is however, also reflected in a second characteristic – a service must define response timeliness for access at provision if it is to be rated by available. The importance of response time to the development of effective and clear purview is reported in [22] and [23] regard an unresponsive site as “not available”. This timeliness access and service provision parameter within the availability definition of the previous section. Access and timeliness, however, do feature in denial of service research. [11] unacceptable service access time as a definitive result of denial of service conditions. [15] goes even further that both access and progress towards provision are the essential goals of service protection. An availability based on service liveness is necessary but not sufficient. An availability definitions based solely on service liveness at provision response time.

The term 'service liveness process' (i.e., the service program in
liveness, a concept that has been treated in numerous papers, including [11]. A distributed service, however, consists of the several cooperating software entities: application process(es), an operating system, and network communication software and API. The networking software and API will be referred to as a network interface for the purposes of this paper.

The term 'service response time' - from a client’s perspective - measures a timeline that begins and ends within the client’s network interface. The calculation of service response time begins when the service request is sent by the client’s network interface. The calculation ends when service provision arrives within the client’s network interface.

An expanded, user-driven definition of distributed service availability incorporates service liveness levels together with service response time. Such a definition may be stated as follows:

Distributed service availability is a measurement of user wait time for service commencement. This user wait time consists of two discrete, but complementary, measurements:

- Liveness probability (AVAIL_liveness) AND service response time (AVAIL_response).

Liveness probability is calculated as follows:

\[
\text{AVAIL}_\text{liveness} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}
\]

where \( \text{MTBF} \) is the mean time between failures of the service process and its network interface, and \( \text{MTTR} \) is the maximum time to repair/restart the service process and its network interface.

Service response time is calculated as follows:

\[
\text{AVAIL}_\text{response} = T_{\text{commencement}} - T_{\text{requested}}
\]

where \( T_{\text{commencement}} \) is the time at which the commencement of service provision occurs within the user's network interface and \( T_{\text{requested}} \) is the time at which the service request is dispatched by the user's network interface.

AVAIL_response must measure server wait time caused by server processing load (T_processing) and network latency (T_latency) between server and client. AVAIL_response must be provided to the prospective client at the time the service request is first made by the prospective client and must be verified by a trusted computing base or operating system [24].

4 Implementation Issues

The implementation of AVAIL_response, as outlined in this paper, has been completed within the Linux kernel. Kernel implementation ensures the necessary timing calculations are completed within a trusted computing base and provided to the client at the earliest possible time. The implementation has required additions to the data structures and functions within the networking code and socket API, together with the design of an additional option within the Transport Control Protocol (TCP). TCP was initially described within RFC793. An operational treatment is provided within [25].

Linux networking is implemented via a two-layer architecture, the BSD layer and the INET layer. The outer layer (BSD) uses the BSD socket API to provide the specific communication protocol access required by the user. Function calls to the API calls are contained within the standard C library. BSD functionality supporting the API is initialised at kernel boot time. All application calls to API functions are then interpreted via one function (sys_socketcall) within the kernel’s BSD code. With this single access point the Linux networking implementation mirrors the SVR4 IPC. The kernel function sys_socketcall is accessed via code in the file entry.S, an assembler file that is run immediately after all software interrupts. The
file entry. S then uses a 'call table' to invoke the appropriate API function such as socket, bind, listen. These functions then invoke the lower layer (INET) functionality as specified by the user's protocol choice.

TCP is implemented within the Linux INET layer. The TCP functions and data structures within the INET layer have been reshaped to calculate and record the necessary timing data required for AVAIL response. A TCP option has been developed to distribute timing data used in the calculation of AVAIL response. TCP has provision for optional header fields identified by an option kind field. Options 0 and 1 are exactly one octet which is their kind field. All other options have their one octet kind field, followed by a one-octet length field, followed by length-2 octets of option data [26]. Current TCP options span 0 to 18 inclusive (notwithstanding 6 and 7 are obsoleted by option 8). AVAIL response has therefore used option kind 19 as shown in figure 1.

<table>
<thead>
<tr>
<th>Kind: 19</th>
<th>Length: 4 bytes</th>
<th>Server Processing Wait Time (T_processing) data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 0</td>
<td>Bit 7</td>
<td>Bit 15</td>
</tr>
</tbody>
</table>

**Figure 1**

The 16 bits (bit 16 to bit 31 inclusive) are used to hold the server processing wait time (T_processing) in milliseconds. This measurement is the time from when the server receives the service request within its INET layer to when the server forwards the initial results of the service request to its (i.e., the server's) socket layer. In our implementation, T_processing is calculated as a rolling average calculation. That is, an average is made for all ESTABLISHED connections from when the initial SYN is received to when service delivery commences in the send buffer of that socket. T_processing is an absolute measurement in milliseconds. T_processing is always recorded as 0 ms for each server as it boots, and subsequently updated after each event pair {request receipt, service commencement}.

Option 19, utilising T_processing and T latency, is exchanged between server and client within the initial setup stages (i.e., SYN, SYN, ACK) of the communication session as displayed in figure 2. The implementation outlined in figure 2 accurately implements AVAIL response as specified in the expanded availability definition. The resulting data is represented in absolute terms and is made available to both server and client via the extended API calls. This enables the client to assess overall availability on a per request basis at the earliest time from a trusted source (i.e., operating system). This same data enables the server to assess request load and network link load – two very good indicators to denial of service conditions. The calculation of server processing wait time (T_processing) is done with respect to the last most recently completed service request. This represents an up to date guide as to the prevailing server load conditions at the time the very next request is received. As stated in RFC 1185, the bandwidth and processing overheads for a TCP option are not likely to cause a performance concern. Opening a TCP connection requires the execution of significant special-case code, and the processing of options is unlikely to increase that cost significantly.

1 The existing TCP timestamp option was considered for use in implementing T_response. It was decided, however, that a new option kind would produce a cleaner result.
Figure 2

Our implementation also records AVAIL liveness within the BSD layer. At socket creation time, the kernel creates a file named via a concatenation of server name, date and time. This file will contain three lines of data. The first line contains server-name, date, and time- commenced. The second line contains a heartbeat indicator written to the file every five seconds over a maximum time of two minutes (ie, overwritten at each two minute interval). The third line is the current date and time - written at two-minute intervals. This data may be used by the server (or other suitable application) to gauge historical server liveness within a precision of five seconds.

5 Conclusion

As outlined by [21], availability means "that the service is there" and also "that it can get the job done". Availability must indicate if the service exists and when service provision can begin. This paper has described an availability definition that expands upon the traditional view by measuring availability in terms of service liveness and the timeliness of service commencement. This expansion produces an availability measurement that is more meaningful to service users and also resolves an inconsistency between the traditional availability approach and denial of service research. The expanded definition has been implemented within the Linux kernel so that service providers and service users may quantify and interpret service availability on a per request basis.

6 References