Gaining access to Internet Quality of Service from an Application(Netbase)

Nicholas Sharples

School of Cognitive and Computing Sciences
University of Sussex, Falmer, Brighton, BN1 9QH, United Kingdom

April 1997

Abstract

This paper is intended as an overview of my recent work toward the construction, of a globally distributed database of WAN connectivity patterns. It is the intent of such a system to provide client applications with prior knowledge as to the Quality of Service(QoS), a path between two host can provide, over a specified period of time.

Using remote objects and native methods now available with the latest release of Java 1.1, a stable framework for development has been constructed. Remote objects provide potential developers with an API into the system, while allowing client applications easy access to the data. Native methods provide access to the finer grained platform dependent timers; required for accurate network performance measures.

The main components of the system have been produced including administration tools, a remote database server and information gathering probes. Having now reached a point from which I can look at the work undertaken, I wish to study it's weakness and decide a strategy for future development.

1 Introduction

The goal of the project is to provide an approximate measure of host to host network connectivity, for specific time periods. It is hoped that over time, continued analysis will reveal patterns in the networks usage which can be used to predicted a base line QoS measure. At this stage in the development of the system, no mechanisms has been established to deal with multiple paths between hosts. Rather, through continual measurement the system should predict the approximate average of the most commonly used paths between the test site and the host.

The following sections decompose the problem into three areas, measuring the network performance, the database used for data storage, and statistical analysis of the obtained measurements.

My primary consideration when designing the structure of the system was to develop, not just a complete application, but an environment for future devel-

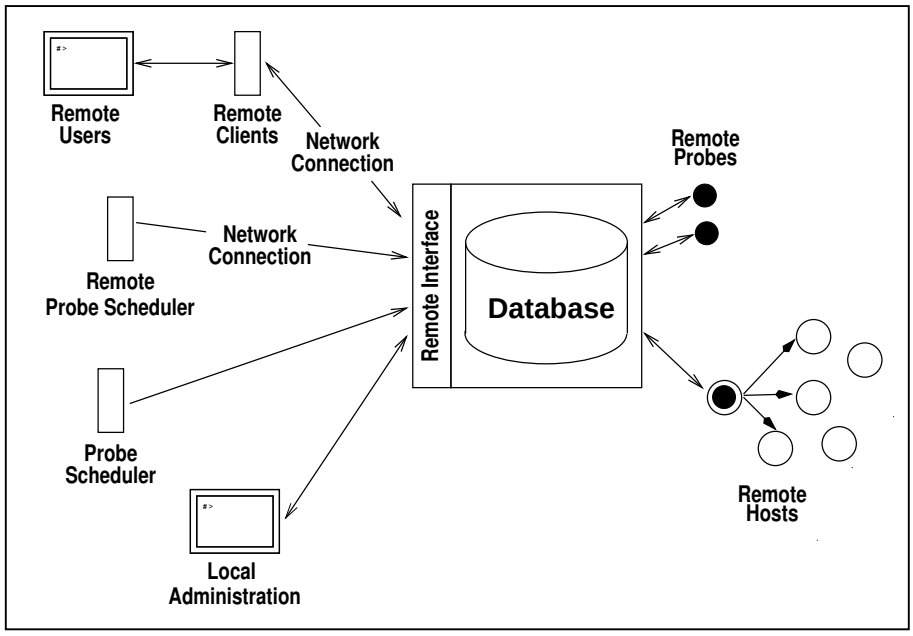

Figure 1: System architecture

opment. To this end, the final system produced has been implemented using remote objects. Using this approach the various information retrieval components and statistical analysis functions of the system can be developed independently. While still passing information back to a central database ready for distribution to peer servers. See Java RMI documentation for a full discussion of Remote method invocation

2 System architecture

The schematic diagram in figure 1 gives an architectural overview of the main system components. The central database provides a remote interface to allow client applications access to it's data across a network. Information is provided to the database via remote probes(remote objects) which can be situated anywhere on the Internet. Allowing independent development of network analysis tools, while still maintaining a central repository. This also offers a light-weight approach to data gathering, remote from the main system.

To qualify as a remote probe, a class must implement the following Java remote interface.

```
public interface Probe extends Remote {
  public TimeSeries probe(InetAddress internetAddress,
                          short packetSize,
                          short seriesLength,
                          short failedProbes) throws RemoteException;
}
```

The probe method must return a valid TimeSeries object which contains the results of *seriesLength* successive probes with packets of *packetSize*, see the TimeSeries class 4.4.1. If *failedProbes* probe attemps fail to return packets, the method returns the timeSeries as is i.e. with the number collected or an empty TimeSeries object. The *internetAddress* parameter specifies the destination end of the path to be measured, the sending end of the path set by the location of the probe object itself.

3 Measuring network performance

Before discussing the method used to analyse network load, it is important to clearly define the information a user will require, and thus the information a network analysis tool must provide. However, this is rather circular as the completed system will allow its users to customize the analysis functions to extract information particular to their needs. To allow for this we must separate the performance measures from the process of numerical analysis. Any measurement tool developed, must provide the raw data rather the processed information. This scheme provides a front end for statistical analysis and allows for a flexible and extend-able environment for the end user. As a baseline goal, to begin development, we have set ourselves the task of constructing a system capable of calculating a QoS figure composed of three values throughput(bandwidth), round trip time(rtt) and packet loss. When a user requests information on a given host, along with the address of the host requested they must also provide the start time and length of time they want the estimation for.

3.0.1 Bandwidth

Bandwidth(measured in bits per second) is the number of bits a 'connection' can transmit in a single second. This definition is by know means complete, a connection is made up of an arbitrary number of links each having varying amounts of available bandwidth. It is the overall route bandwidth we require, which is depended on the link with the least available bandwidth, known as the bottle neck bandwidth.

3.0.2 Round trip time

The time taken for a packet to complete the journey from the local host to the remote host in milliseconds. This can be used to prime the retransmition timers used in adaptive protocols.

3.0.3 Packet loss

The percentage chance that a packet will be lost, either through corruption or dropped by a router.

3.1 Network probes

To attain the required QoS elements a simple tool was developed using the Packet-Pair technique developed by Keshev see [Kes95]. The original technique involves sending two packets back to back, and measuring the delay between the

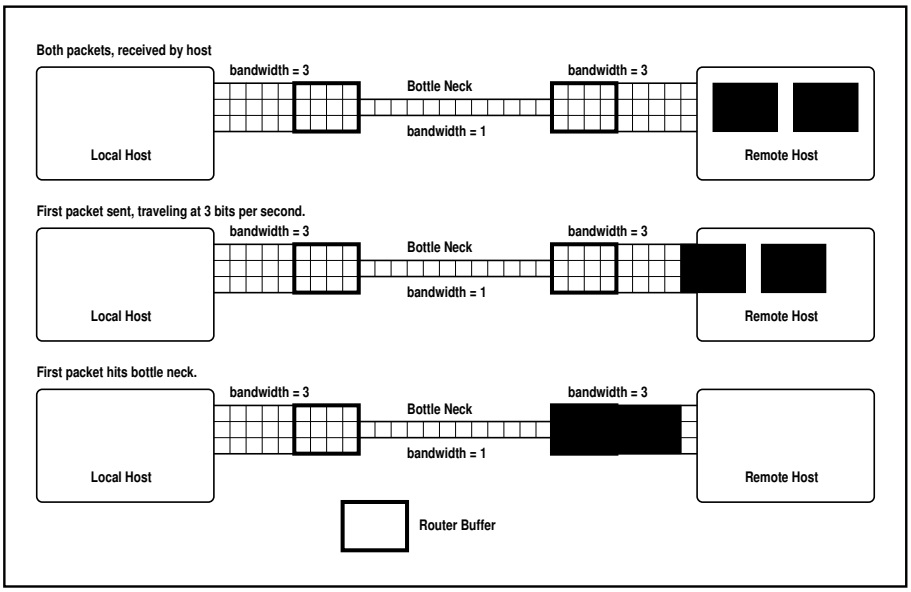

Figure 2: The remote host returns the two UDP packets. For this example the packets are 12 bits.

returning acknowledgments. The inter-packet delay is the time taken to write the second packet through the smallest bottleneck connection, on its return journey. All delay sustained on the outward journey is negated when the first packet hits the bottleneck in its return path. This technique can be simplified with the use of ICMP ping packets, which every Internet host must return to the sender. Using these we can send two packets of arbitrary size and measure the inter-packet delay as the packets are return to their source. For a full description of the ICMP protocol see [W.S94].

Shown in figure 2 are two ping packets on their return journey to the local host. As can be seen from the diagram the bandwidth is 3 bits per second, so each packet moves 1 vertical column each second. In figure 3 the first packet has hit the bottle neck, its progress is now reduced to 1 bit per second. The second packet is still travelling at 3 bits per second, and has to wait for the first packet to go through the bottle neck before it can proceed any further. In the final figure 4, the first packet is through the bottle neck and moves away at 3 bits per second. While the second packet is transmitted across the slower link at 1 bit per second. For the remainder of this paper I shall refer to this as *probing*. This form of probing is imprecise as are many network analysis techniques, however it is hoped the imprecision can be reduced through continually probing over time.

3.2 Probe scheduling

An important consideration is the amount of network traffic caused by the probes as they analyse the connection to a host. An arbitrary maximum of one probe per minute, was decided upon early in the project. It should be stressed

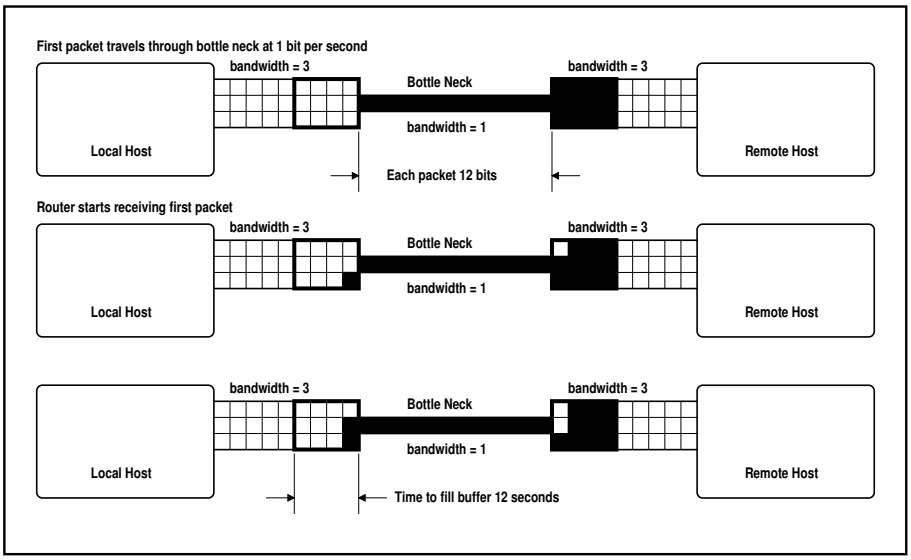

Figure 3: First packet transmitted across slower link

that any network performance measure which adds to the network traffic will in some way effect the measure taken. However, it is hoped that restricting the probes to only once a minute will cause little excess pressure on the path.

This raised a problematic issue, how to decide which hosts to probe and when. As time progresses new hosts added to the system would require a larger number of probes than hosts which had been on the database for some time. To avoid this issue, simple scheduling classes were produced with the aim of suppling a potential Netbase developer with a simple extendible class. This postpones the construction of complex scheduling algorithms in favour of light weight customizable classes.

4 The Database

The primary role of the database is to provide persistence across all the data objects it maintains. It's secondary function is to satisfy queries from its users, during which it must efficiently extract and analysis specific records within its system. Thirdly, once the database has been developed, it must be initiated with suitable data.

For the purposes of this project, it is enough to produce a system which maintains a single repository, or database, of information. Later work, if undertaken, will aim at a globally distributed database much like the architecture employed in the Domain Naming Service.

4.1 Persistence

Using Java, object persistence is easily achievable by serializing the object as a stream of bytes, and writing these to a file. This does have the disadvantage

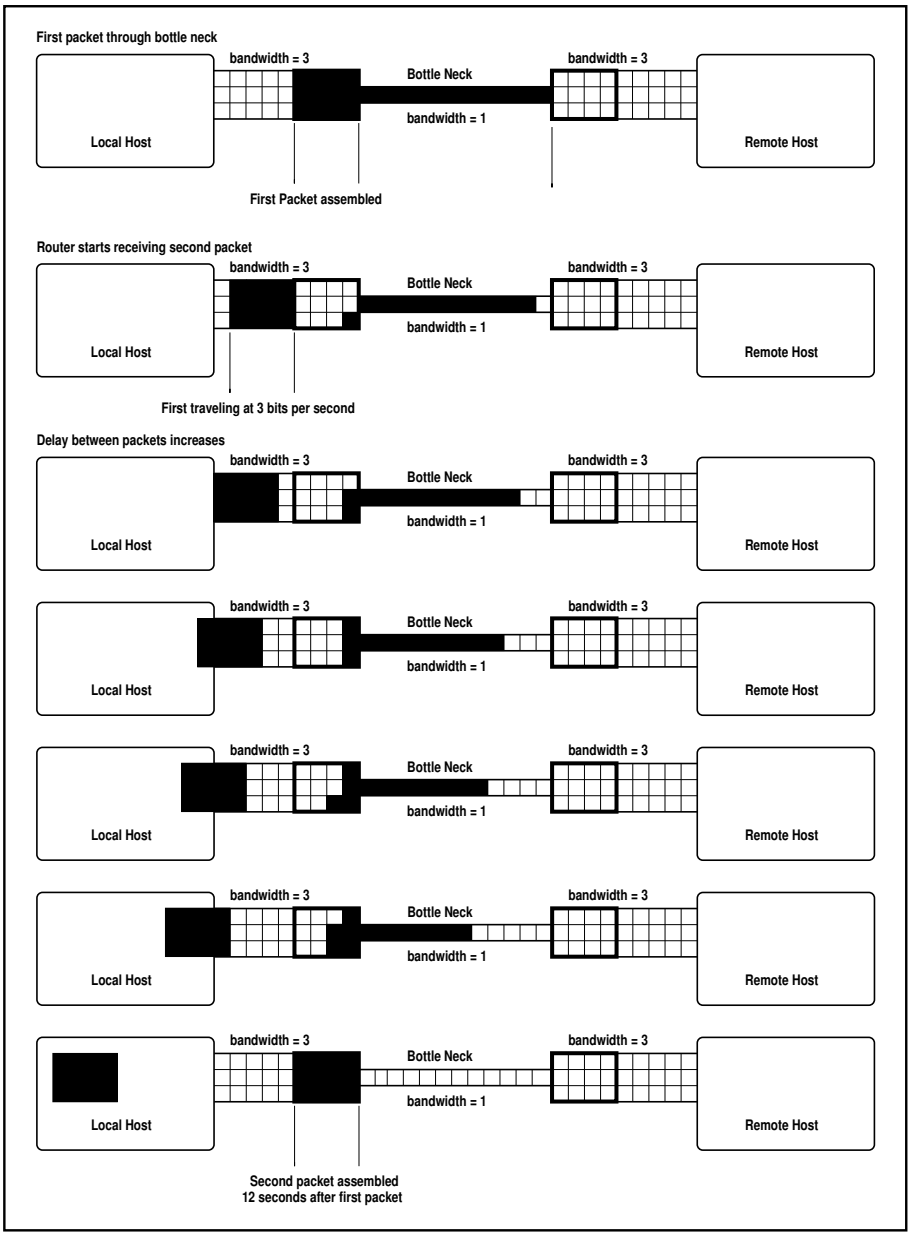

Figure 4: As the second packet is being transmitted across the slower link, the first packet travels away faster introducing a time delay between the packets. $Bottleneck = \frac{r_1 - r_2}{b}$

that if the class is altered the objects cannot be de-serialized. So, object serialisation should only be considered once the system is reasonable stable, and an upgrade solution has been established. However, during development, the database stored object state in simple text files. These files, designed more for readability than performance, became the preferred method for data storage. Allowing easy manipulation via text editors for test purposes and construction of reports.

4.2 Data structure

To efficiently handle a query, the information must be stored in an optimal structure. For the purposes of this system, a query will relate to a single host over a time period i.e. Extract and analyse all the information for host x, from 9am till 10am.

The dynamical nature of the Internet makes accurate load predication extremely difficult. However, it is not the intent of this system to provide highly accurate information, rather rough estimates provided by information taken over extended periods of time. To minimise the in-accuracy a single result will consist of n consecutive samples, compiled into a single time series. Each time series represents the performance of the path for a single second of a 24 hour period. Any subsequent tests performed at that second are added to those already gathered. Each time series is assigned a minute key, by which it is grouped into minutes, and minute summary values calculated. A host record on the database contains a vector of minute summary values, it is these values which are extract during a query.

4.3 Priming the database

To initiate the system a number of host address records must be established on the database. It was my original intention to extract the address records from the network access logs maintained by my local host. However, a more flexible approach adopted later in the project was to perform a zone transfer from the DNS using nslookup or DIG. For information on performing DNS zone transfer see [AL97] page 229. Additional hosts will be added to the system as user queries are unsatisfied. Specifically the following information is stored per host.

4.4 The Host class

As well as the host information, an object of this class also acts as a container for all the network performance data relating to it.

- The IP address of the host.
- The date the host entered the system.
- The date and time the host was last probed.

4.4.1 The TimeSeries class

This class acts as a container class for individual probe results. It also maintains totals for all the ProbeResult objects it contains.

- The IP address of the host.
- The time and date the probe took place.
- A vector of probe results
- Number of probes which make up the series.

4.5 The ProbeResult class

This class represents the performance measure of the path from the probe to the host.

- The IP address of the host.
- The time and date the probe took place.
- The size of the packet used.
- The measured bandwidth.
- The measured round trip time.
- The number of packets lost.

5 Database analysis

Users of the system will on the whole access the database across a network, using remote objects. This offers two significant advantages. Client side analysis tools can be developed independently, and the CPU intensive analysis can carried out by the client. For development and testing some simple schemes were constructed, which through object inheritance can easily be extended by future developers and users.

For each probe the bottleneck bandwidth can be calculated using the equation given in 1 where Q_b is the queueing delay at the bottleneck and ρB is the bandwidth at the bottleneck. The bottleneck estimate can contain an amount of noise, due to fluctuations in network traffic, to overcome this an average is taken over several successive probes, and the bandwidth recalculated using the equation 2. To calculate the average round trip time we use the equation 3 where r_n is the round trip time for packet n. Using this method does have problems in that a lost packet has 0 round trip time, and cannot therefore be included in the averaging calculations. The percentage chance of packet loss can be calculated using 4. For a full discussion of the packet pair estimation technique see [Jac88]

$$Q_b = \frac{b}{\rho B} \tag{1}$$

$$Q_b = \frac{1}{n}\sum_{i=1}^{n} Q_b n \tag{2}$$

$$R = \frac{1}{n}\sum_{i=1}^{n} r_1 - r_2 n \tag{3}$$

$$L = \frac{l}{n} \tag{4}$$

5.1 User Interface

Although the system was primarily build to allow applications access to data, during construction it became advantageous to have an overview of the data collected. This requirement lead to development of a graphical front end which provides a graph like display showing the information collected per host. This feature has survived through out development and now provides database administration as well as simple client access to the current database.

References

[AL97] Paul Albitz and Cricket Liu. *DNS and BIND.* O'Reilly and Associates Ltd, 1997.

[Jac88] Van Jacobson. Congestion avoidance and control. In *Proc. ACM SIGCOMM 88*, University of California, August 1988. Sigcomm.

[Kes95] Srinivasan Keshev. A control-theoretic approach to flow control. In *Computer Communication Review*, pages 188–201. ACM press, 1995.

[W.S94] Richard W.Stevens. *TCP/IP Illustrated Volume 1.* Addison-Wesly Publishers Ltd, 1994.