420
10P

Satellite Network Performance Measurements Using Simulated Multi-User Internet Traffic

Hans Kruse
J. Warren McClure School of Communication Systems Management, Ohio University; hkruse1@ohiou.edu

Mark Allman
NASA Lewis Research Center/Sterling Software; mallman@lerc.nasa.gov

Jim Griner,
NASA Lewis Research Center; jgriner@lerc.nasa.gov

Shawn Ostermann
School of Electrical Engineering and Computer Science, Ohio University; ostermann@cs.ohiou.edu

Eric Helvey
Lucent Technologies; helver@alliances.org

Abstract

As a number of diverse satellite systems (both Low Earth Orbit and Geostationary systems) are being designed and deployed, it becomes increasingly important to be able to test these systems under realistic traffic loads. While software simulations can provide valuable input into the system design process, it is crucial that the physical system be tested so that actual network devices can be employed and tuned. These tests need to utilize traffic patterns that closely mirror the expected user load, without the need to actually deploy an end-user network for the test. In this paper, we present trafgen. trafgen uses statistical information about the characteristics of sampled network traffic to emulate the same type of traffic over the test network. This paper compares sampled terrestrial network traffic with emulated satellite network traffic over the NASA ACTS satellite.

Introduction

When designing and deploying new network technologies and infrastructures, designers must be able to test the characteristics of the new system under a realistic load. Although software simulation of the system can provide valuable information about the expected system characteristics and point out problems early in the design process, it's extremely difficult to correctly model all the subtleties of a working network and its usage patterns. A complementary approach that often proves beneficial is to test the new network (or a subset thereof) under actual traffic. Unfortunately, it's often difficult to generate suitably-accurate network traffic to conduct such tests, particularly when that traffic is the product of the interactive behavior of many users.

In this paper we present trafgen [5], a software system capable of creating TCP/IP traffic flows that statistically mirror those of an observed network. Trafgen takes as input the traffic characteristics of a network with usage patterns similar to the ones expected for the network to be tested. trafgen then randomly initiates TCP-based data flows that reproduce the input pattern, traffic types, and connection data sizes of the measured network, subject to an overall scaling factor. We describe here a system which can replicate a network of interest, provided that a suitable description of the expected traffic patterns can be obtained.

This paper describes a preliminary series of experiments run over the NASA ACTS satellite network. We gathered network statistics from an actual Internet Service Provider and built a version of trafgen which emulates this traffic. We then conducted a series of experiments at different multiples of the sampled traffic load over the satellite network to test the behavior of trafgen and verify that it preserves the characteristics of the original terrestrial traffic. The following sections describe the trafgen program, the TCP traffic library tcplib [4] on which it is built, the satellite network that we constructed to test the program, the experiments that we conducted, and an analysis of the results.

How Trafgen Models a Network

A single trafgen program is able to model a network in which TCP data is generated by a computer running trafgen and absorbed by a second computer. A simple example of this is seen in Figure 1.

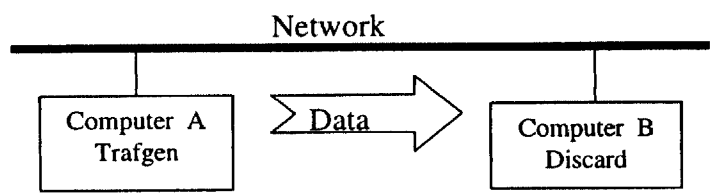

Figure 1: A simple network model using trafgen

trafgen does not directly model bi-directional communications such as the request-response nature of HTTP connections. However, trafgen can model the aggregate behavior of many such conversations by generating data that correctly emulates the characteristics of both the requests and responses. If the network over which trafgen is running is a shared medium (as in Ethernet in which the requests and responses would travel over the same physical channel), a single trafgen program can emulate both the requests and responses. However, in most wide-area network links the traffic flow in one direction uses a separate channel than the traffic in the reverse direction. This is true for the satellite network used in the experiments reported in this paper. For these networks it is necessary to have an instance of trafgen at both ends of the network to emulate requests and responses originating at either end of the media, as shown in figure 2.

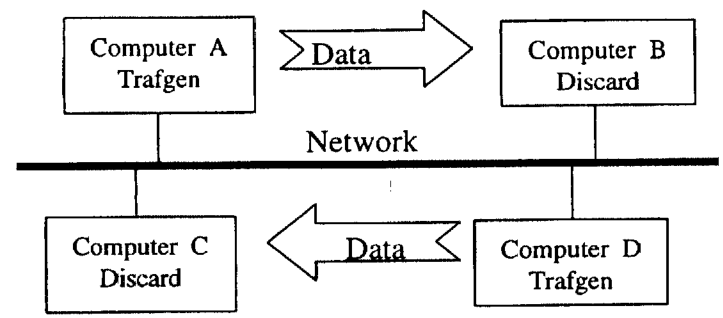

Figure 2: Using trafgen to independently model both sides of a bidirectional traffic flow

The Tcplib Library

The first step in using trafgen is to determine what network conditions are to be emulated. The input to trafgen is a set of traffic characteristic histograms for tcplib [4]. The original tcplib library gathered statistics about the application protocols FTP[10], NNTP[6], SMTP[11], TELNET[9], and PHONE (the application protocol PHONE is no longer widely used and was not modeled). Missing from this list, but a large presence on modern networks, was HTTP[1], which we added to the library. We also added information on connection interarrival times. Our modified version of tcplib uses the following network characteristics:

conv.conv_time	Conversation Interarrival Time, the time from the beginning of one connection to the beginning of the next
breakdown	Percentage of connections from each of the applications
ftp.ctlsize	The total number of bytes in an FTP control connection
ftp.itemsize	The total number of bytes in a single FTP file transfer
ftp.nitems	The number of file transfers initiated from a single FTP control connection
http.itemsize	The number of bytes in an HTTP connection
nntp.itemsize	The number of bytes in an NNTP connection
nntp.nitems	The number of transfers in an NNTP connection
smtp.itemsize	The number of bytes in an SMTP connection
telnet.duration	The length of time that the TELNET connection exists
telnet.pktsize	The size of a single TELNET packet
telnet.interarrival	The time between successive TELNET packets

Table 1

Each of these data sets is modeled as a cumulative probability table. For example, a portion of the telnet duration table is given in Table 2; 17% of the connections (35 connections) had duration between 100ms and 500ms, 24% of the connections (49 connections) had a duration between 100ms and 600ms, etc. The longest connection lasted 4823.1 seconds.

Duration (ms)	% Conversations	Running Sum	Counts
100	0.1724	35	35
500	0.2414	49	14
600	0.3498	71	22
700	0.4089	83	12
800	0.4138	84	1
1000	0.4187	85	1
...			
4800300	0.9951	202	1
4823100	1.0000	203	1

Table 2

A portion of the TCPLIB input distribution for Telnet conversation lengths.

Using a pseudo-random number generator, the tcplib library provides representative samples from this distribution using the probabilities given.

Obtaining Tcplib Source Data

Historical tcplib-style data is widely available. However, because the purpose of our experiments was to model current networks, we needed current statistics. To accomplish this in our work, we identified a local Internet Service Provider that was willing to allow us to collect packet headers.

The network that was emulated for the experiments presented in this paper was NewWave Internet in South Charleston, West Virginia, which provided us with several large tcpdump-format trace files from various times of the day. For this paper, we used a trace file collected starting at 11:15pm (a typically busy time of day for this ISP) on Feb 4th, 1998. The file contained 2.9 million packets spanning an hour and 20 minutes. This data file was divided into two data sets: data flowing into the ISP from the Internet and data flowing out of the ISP to the Internet. The "incoming" data was used to drive the trafgen emulation at one end of the satellite link and the "outgoing" data was used to drive the other end of the link.

We captured network packets using tcpdump[7] at various times of the day and then analyzed those packet traces to generate tcplib-style data tables. To generate tcplib data tables from the source data, we wrote a module for tcptrace[8] that can quickly analyze large packet traces and generate the necessary data. For example, from the packet trace mentioned above, the subset of 27,000 connections initiated from inside the ISP can be converted to tcplib data in about 50 seconds on a current-generation Sun Sparc Ultra-2 workstation.

Once a set of tcplib data is obtained, a version of the tcplib library is compiled against this data and then linked against the trafgen program to emulate a particular set of network conditions.

How Trafgen Models Network Traffic

The trafgen main program loop is an infinite loop as shown below:

```
trafgen:
    loop forever {
      /* ask what type of connection to run next */
        switch (next_connection_type()) {
              /* create a thread to make one connection
*/
              ftp:    MakeThread(doFTP());
              http:   MakeThread(doHTTP());
              nntp:   MakeThread(doNNTP());
              smtp:   MakeThread(doSMTP());
              telnet: MakeThread(doTELNET());
        }

        /* get a conversation interarrival time sample
and wait */
        sleep(conv.conv_time());
    }
```

trafgen repeatedly asks tcplib for the next type of connection to emulate. It then creates a thread[1] to handle that new connection. Finally, it sleeps (delays) for an amount of time determined by tcplib to be an appropriate conversation interarrival time.

trafgen uses the same model for both SMTP and HTTP. Each connection consists of a single burst of data. The TCP protocol sends the data as quickly as possible and then closes the connection. Note that this behavior only emulates HTTP version 1.0 without persistent connections. The algorithms are as follows:

```
doHTTP:
      Send(http_itemsize());
      exit;

doSMTP:
      Send(smtp_itemsize());
```

---

[1] Under NetBSD we simulate threads using multiple processes.

```
        exit;
```

The algorithm to emulate an NNTP connection is only slightly more complex than HTTP and SMTP. The NNTP protocol allows the sender to deliver several bursts of data, which tcplib and trafgen emulate using the distributions nntp_nitems() and nntp_itemsize() as follows:

```
doNNTP:
        for (item = 1 .. nntp_nitems());
                Send(nntp_itemsize());
        exit;
```

Since TELNET is used primarily as an interactive protocol, it is emulated by tcplib and trafgen by alternately sleeping and sending a (usually small) burst of data until the connection duration has been reached. The algorithm is as follows:

```
doTELNET:
        duration = telnet_duration();
        while (time < duration) {
                Send(telnet_pktsize());
                sleep(telnet_interarrival());
        }
        exit;
```

FTP is the most complex of the protocols emulated by trafgen. Recall that FTP uses a control connection to initiate directory listings and file transfers and then a separate TCP connection for each transfer of data. The tcplib library models the FTP control connection as a telnet connection (with the same data bursts and quiet time) with a fixed length in bytes. After that control interval has completed, a file transfer is initiated on a new TCP connection whose size is determined by tcplib.

The algorithm is as follows:

```
doFTP:
        for (item = 1 .. ftp_nitems());
                num_ctl_bytes = ftp_ctlsize();
                while (num_ctl_bytes > 0) {
                        len = telnet_pktsize();
                        Send(len);
                        sleep(telnet_interarrival());
                        num_ctl_bytes -= len;
                }
                Send(ftp_itemsize();
        }
```

Once trafgen has been compiled to emulate a particular traffic pattern, the amount of traffic generated can be controlled with a single run-time parameter, BRK. The BRK (affectionately referred to as the 'Big Red Knob') is a multiplier that is applied to each connection interarrival time. When the BRK is set to 0.5, for example, trafgen multiplies each connection interarrival time generated by tcplib by 0.5, causing new connections to be started at twice the rate of the source data, effectively doubling the amount of data generated (subject to the limits of the networks and hardware involved). Likewise, a BRK value greater than 1 will produce fewer connections per unit time than the source data and correspondingly less data.

Accepting Trafgen Data

While the trafgen program accomplishes the traffic generation, the experiments require a second computer to accept the data. Recall that trafgen models the network by generating TCP connections containing data that only flow in one direction (outbound from trafgen). The computer at the other end of the connections needs only to accept and discard the data. Most Unix platforms already have a standard TCP discard server. Unfortunately, this server is not appropriate for these experiments for at least two reasons. First, the built-in discard server typically uses a small receive window [2] to limit data throughput, making it inappropriate for emulating applications which transfer large volumes of data. Furthermore, many of these built-in servers contain a security feature that causes them to stop accepting new connections if the frequency of incoming connections is too high.

When designing a new discard server for these experiments, we also needed to insure that the server would be able to accept new connections at a rate approaching 10s or 100s of new connections per second. This seemed to preclude the possibility of using a new process for each new connection, as done by the standard discard server. The discard server that we built runs as a single process and has been able to absorb any amount of traffic that we've been able to generate (over a 10Mb Ethernet link) without placing a significant load on the computer.

Results

For our experiment, trafgen was deployed using the NASA ACTS satellite to provide a T1 link between two routers. Attached to each router was an Ethernet segment with a traffic generator and a discard server. The generated packet flow is captured (using tcpdump) on each Ethernet segment. We refer to the sampled traffic described in the previous section as the "source" traffic. The traffic generated and recorded in the experiment is referred as "observed" traffic.

The same analysis code used to create the source histograms is used to analyze the observed traffic. Figure 3 shows the cumulative probabilities for observing FTP item sizes, for both the observed and the source traffic. The two distributions clearly track each other.

In order to compare a large number of these histograms, we compute the first

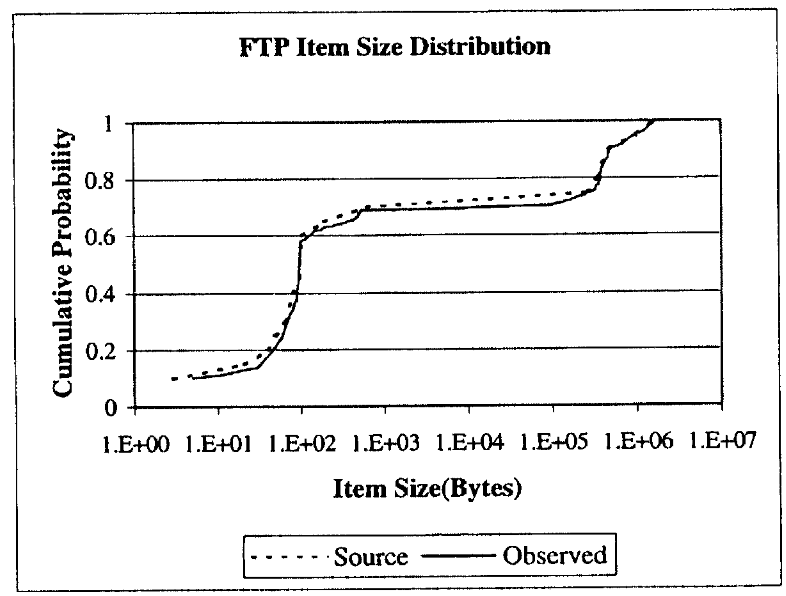

Figure 3: A comparison of source and observed tcplib distributions for FTP item sizes.

few moments of the distributions. For these computations, we start with the distributions shown in Table 1. The distributions contain the observable variables, x_i, and the frequency, f_i, with which each value x_i was found in the recorded traffic. The normalized frequency,

$$\overline{f_i} = \frac{f_i}{\sum_i f_i}$$

represents the probability of tcplib returning an item size between x_{i-1} and x_i. For this paper we report on the averages represented by each distribution, based on this interpretation of the probabilities. In figure 4 we show the average values for a number of the observed distributions, plotted against the averages obtained from the source distributions. The results for both traffic directions and four separate experiments of 2 hours each are shown.

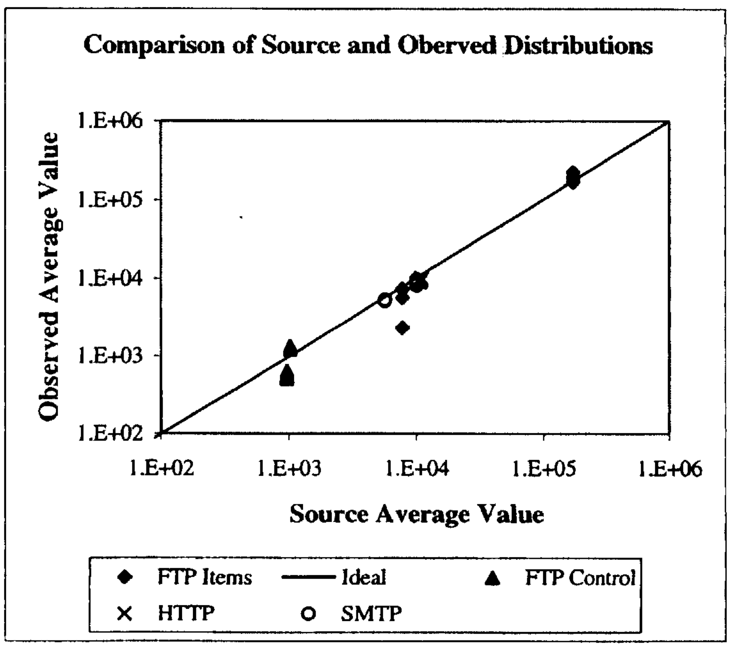

Figure 4: The observed average values for various tcplib distributions, plotted as a function of the averages obtained from the source distributions.

If the observed distribution were to reproduce the source distribution perfectly, the data points in Figure 4 would fall on the line labeled 'Ideal" in the figure.

Due to limitations in the NetBSD Unix operating system used in these experiments, our implementation of the simulated Telnet sessions is rather complicated; and as of yet, incomplete. We plan to include simulated Telnet sessions in future experiments.

We find that the observed and source averages in most cases are no more than 10% apart. Notable exceptions are the FTP control and item sizes. We have examined the distributions in detail, and find that the FTP distributions for the sampled traffic are very sparse in some regions. The FTP itemsize distribution in figure 3 shows this behavior for items sizes in the range between 5000 bytes and 800,000 bytes. In this area the cumulative distribution is flat because there no item sizes in this range appeared in the source traffic. As indicated above, tcplib chooses values in such a sparse region with the probability assigned to the item size at the top of the sparse region, leading to a large statistical spread of item sizes. We are currently examining this behavior to decide if this is indeed a desirable feature of the tcplib mechanism, or if the FTP data needs to be augmented to better determine the source distribution for tcplib.

As described earlier, trafgen has the ability to scale the conversation interarrival times to allow the creation of different traffic amounts. In the experiment, we have operated trafgen at four values of BRK, between 1.0 (which reproduces the source traffic) and 0.5 (which requests twice the source

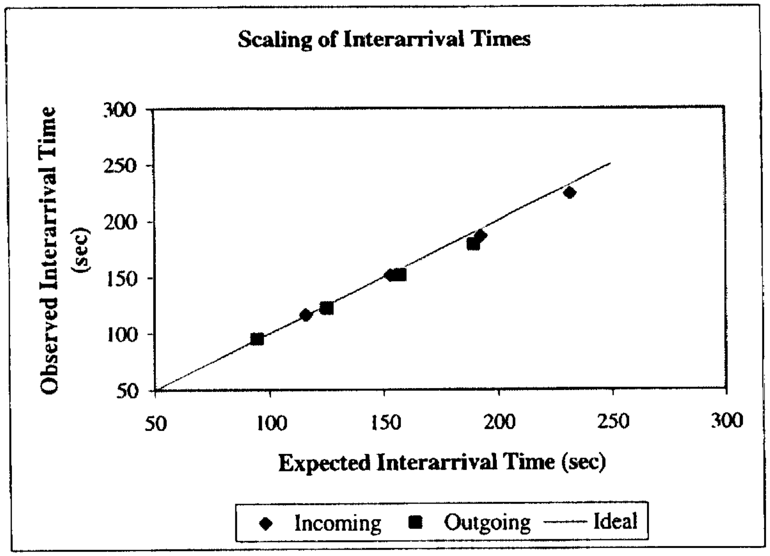

Figure 5: This figure shows the observed average conversation interarrival times against the expected values based on the BRK setting.

traffic). In figure 5, we plot the observed average conversation interarrival times against the expected values, obtained by multiplying the average interarrival time from the source distribution by the scaling factor.

It is apparent from this figure that trafgen correctly changes the scale of the interarrival times. An early object of these experiments was to determine if the scaling of conversation interarrival times would translate into a comparable increase in observed network traffic. This connection is not automatic, since trafgen hands each object to the TCP layer for transfer. TCP's congestion control [2,3] combined with the channel bandwidth and the router queues will determine the rate of packet flow. For these experiments, the routers used FIFO queuing, with bottleneck queues of 70 packets, which is approximately one delay bandwidth product at a link MTU of 1500 bytes.

During each experiment, the routers connecting the Ethernet segments to the satellite T1 circuit are queried via SNMP for basic interface statistics. For our comparison we determine, at various points in time, the number of octects received on the T1 interface since the start of the experiment. We convert this number of octets into a cumulative data rate by dividing by the time elapsed since the start of the experiment. Figure 6 shows the results for 3 different settings of the scaling factor, BRK. Over the region of circuit utilization explored by our experiment, we conclude that the trafgen scaling mechanism not only correctly scales the arrival of connections, but also increases the network traffic by the same scaling factor. These results indicate that a test network using trafgen can be placed under a predictable load using the scaling described in this paper. Clearly, we expect that this scaling behavior will break down when the generated traffic flow approaches the capacity of the network. A study of this scaling behavior is part of the ongoing experiments.

Conclusions

In this paper we have described the implementation of trafgen, and shown how it relates to the existing tcplib data base. We have demonstrated that the network traffic created by trafgen reproduces the characteristics of the tcplib input data. Finally, we have introduced a scaling mechanism which allows the creation of various traffic volumes with otherwise similar characteristic. Our

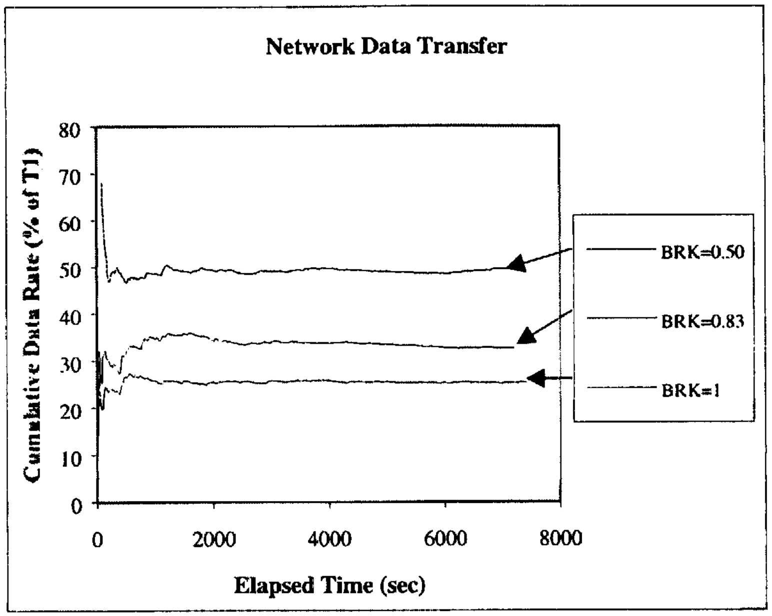

Figure 6: Cumulative data rates observed by the network routers for different BRK settings.

experimental data shows how the network traffic scales with different levels of generated traffic.

Acknowledgements

The work reported in this paper has been supported in part through NASA grant NCC 3-430. Access to the NASA ACTS satellite system has been granted under experiment number 140.

The authors gratefully acknowledge NewWave Internet which made the source traffic data available to us.

References

[1] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol - HTTP/1.0, May 1996. RFC 1945.

[2] Douglas E. Comer. Internetworking with TCP/IP Volume I, Principles, Protocols, and Architecture. Prentice-Hall, Englewood Cliffs, New Jersey, third edition, 1995.

[3] Douglas E. Comer and David L. Stevens. Internetworking with TCP/IP Volume II, Design, Implementation, and Internals. Prentice-Hall, Englewood Cliffs, New Jersey, third edition, 1999.

[4] Peter B. Danzig and Sugih Jamin. tcplib: A library of TCP/IP traffic characteristics. USC Networking and Distributed Systems Laboratory TR CS-SYS-91-01, October, 1991. ftp://catarina.usc.edu/pub/jamin/tcplib.

[5] Eric Helvey. Trafgen: An Efficient Approach to Statistically Accurate Artificial Network Traffic Generation. Master's thesis, Ohio University, 1998.

[6] Brian Kantor and Phil Lapsley. Network News Transfer Protocol, February 1986. RFC 977.

[7] Steve McCanne, Craig Leres, and Van Jacobson. Tcpdump. ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.

[8] Shawn Ostermann. Tcptrace. http://jarok.cs.ohiou.edu/software/tcptrace/.

[9] J. Postel and J. Reynolds. TELNET Protocol Specification, May 1983. RFC 854.

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

[11] Jonathan B. Postel. Simple Mail Transfer Protocol, August 1982. RFC 821.