Assessing Internet Performance Metrics Using Large-Scale TCP-syn Based Measurements

Martin Horneffer

Abstract— **A previously introduced strategy [1] for evaluating the "overall Internet connectivity" of Internet access points comprises three main tasks: compiling a sample list of hosts representative for the entire Internet, conducting network layer measurements with these hosts and, finally, evaluating the metrics with respect to the performance of end user applications.**

This paper describes advanced techniques to extract the top TCP server hosts from monitored Internet traffic, and to perform efficient large-scale measurements of network layer metrics using TCP-syn packets. It also presents experience gained by continuously conducted measurements with these techniques and gives examples how analytical models like those described by Padhye et al. [2], [3] can be used to estimate application layer performance from results of network layer measurements.

Keywords— **Active measurements, traffic monitoring, NetFlow, performance metrics, IPPM.**

I. Introduction

The traffic analysis given in appendix A shows that several hundred to some thousand hosts attract most of the Internet traffic of a typical large user group. Thus it makes sense to use a list of roughly that size when composing a representative sample of Internet hosts.

However, efficiently measuring network layer metrics of the paths to hundreds or thousands of Internet hosts has to meet some specific requirements:

- The measurements must be low-bandwidth. Even when examining the paths to multiple hosts in parallel, the traffic generated by active measurements must be small compared to real traffic.
- No particular co-operation of the other side must be required. Installing specific software on the other side or even just asking for explicit permission is not feasible for the intended number of hosts. It would also impede an unbiased selection of hosts which relies on the fact that it is completely automated.

Thus, the measurement must be based solely on standard behavior that is exhibited by every host on the Internet and it must be able to establish network layer metrics such as connectivity, delay and loss with just a few small packets.

Simply using ping (send/receive ICMP-echo-request/reply packets) for these measurements will generally not suffice, since many host sites either block this kind of ICMP packets or rate-limit them [4].

Sending carefully designed UDP-echo-requests and analyzing answers of type UDP-echo or ICMP-port-unreachable has yielded useful results [1]. But this technique still has some drawbacks:

Martin Horneffer is with the Regional Computing Center of the University of Cologne, Germany. E-mail: Horneffer@RRZ.Uni-Koeln.DE

- Some operating systems rate-limit the overall number of ICMP replies sent, leading to false packet loss data.
- An increasing number of Internet sites block all packets that are not addressed to one particular service.
- In the future, techniques such as differentiated services might give UDP packets a different quality of service on the Internet than TCP packets.

All these problems can be overcome by using TCP instead of UDP or ICMP: a TCP-syn packet is sent to the host and a reply of type TCP-syn-ack or TCP-reset is waited for. This yields reliable measurements of connectivity, (bi-directional) packet loss and delay. No firewall can block these packets if they are addressed to the particular service offered by the host and no ICMP-rate-limiting or differentiated services can distinguish these measurement packets from other traffic. Section II describes an implementation of this approach.

An important issue is the compilation of a suitable list of Internet hosts. There are several good reasons why these hosts should be TCP servers:

- Most typical user groups are more interested in connecting to Internet servers than in offering services to other Internet users.
- The measurement technique mentioned above uses TCP ports. While it can be used on any Internet host, whether or not it runs any TCP services, and probably would work in most cases, only a host running a known TCP service gives *guaranteed* reachability.
- System uptime is not usually good for Internet hosts that are not servers; office PCs are powered down after business hours and dynamic IP addresses are unpredictably online or offline. If a hosts' uptime is not close to 100 % it is not good for measuring network connectivity.

Thus it is desirable to get the top N TCP-servers that are used by a given user group. Section III describes how this can be achieved and introduces an algorithm for automatic discovery of the server-side of Internet flows.

Experience with large-scale measurements based on the techniques mentioned above is presented in section IV. It also shows some problems that have been encountered and measures that have been taken against these problems.

Section V gives some examples of what the measured network layer metrics mean to application layer performance and, finally, section VI summarizes the paper and draws final conclusions.

II. TCP-syn based network layer measurements

A. TCP connection establishment

Establishing a TCP connection uses the so-called 3-way-handshake which typically consists of 3 packets as shown

in figure 1. Terminating a TCP connection takes another 4 packets.

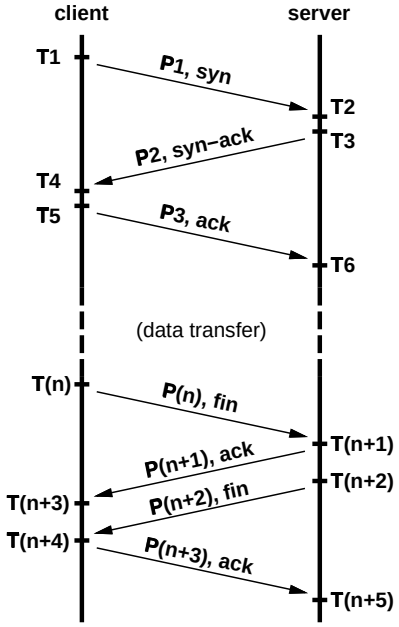

Fig. 1. Regular TCP connection establishment and termination.

TCP connections are always client/server based: first the server side does a so-called "passive open" to offer its service and waits for clients to actively connect. In more detail, connection establishment works as follows:

- At time T_1 the client initiates the connection by sending a syn-packet to the server.
- The server receives the syn-packet at T_2 and, given that there is an open service,
- responds with a syn-ack-packet at T_3.
- This packet arrives at the client at time T_4,
- and the client answers with an ack-packet at T_5.

At time T_4 we already have all the information required to determine the network layer metrics connectivity, delay and packet loss:

- If the syn-ack-packet arrives at all, there is connectivity between the client and the server. It also indicates that no packet loss occurred (see below for a further discussion of the impact of retransmissions on packet loss measurement).
- $T_2 - T_1$ is one-way-delay from client to server and $T_4 - T_3$ is one-way-delay from server to client.

Usually the server answers immediately to the first syn-packet, so that $T_3 - T_2$ is small compared to Internet packets delay and can be neglected: $T_3 \approx T_2$. Then we get the round-trip-delay (or round-trip-time, RTT) as

$$RTT = T_4 - T_1 \tag{1}$$

This has the great advantage that all required information is available at the client side and no additional action is required at the server side.

The drawback is that *only* round-trip-delay and -loss can be obtained — there is no chance to get one-way-delay or -loss. There is also an inherent limit to the accuracy of loss measurement: since we do not control the server we cannot guarantee that the server answers all syn-packets; a packet might be lost within the server as well as in the network. However, the probability that an Internet server loses a received TCP-syn-packet usually is very small. Even if the server is overloaded, the problems start after T_5 when the operating system handles the newly opened connection to the server application. Open connections are queued at this point and only if this queue overflows the operating systems starts to refuse sending packet P_2 [5]. Accuracy of loss and delay measurements with this approach is analyzed in appendix B. This issue indeed causes some error in the result for packet loss, but since the error is small compared to packet loss actually observed in the Internet, the results are still useful.

B. TCP-syn based Measurements

The first two packets (P_1 and P_2) of a TCP connection already suffice for measurement of the basic network metrics. But there is a half open connection at that moment and the server has already created state. This half open connection should be terminated again because otherwise successive measurements could easily aggregate state on the server side, possibly constituting an unintentional denial-of-service attack. It is not required to run through the complete connection establishment and termination as in figure 1. Instead, we can simply send a packet of type TCP-rst (reset) which "ungracefully" resets the connection. This has the effect of removing all state on the server side for a particular connection. A measurement using this approach is depicted in figure 2.

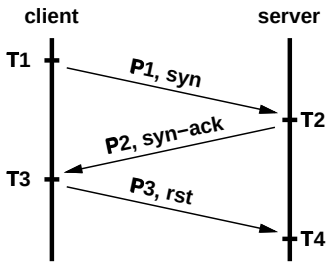

Fig. 2. Measurement by incomplete establishment of a TCP connection.

Even if the server does not offer the requested service it can be used for measurements since it has to answer with packets of type TCP-rst whenever a closed port is addressed. In this case a measurement consists of only two packets as shown in figure 3.

C. TCP retransmission

A basic feature of TCP as a reliable transport protocol is to *retransmit* a packet after a certain period (the retransmit timeout, RTO) whenever a packet has been sent to the other side and an acknowledge packet is expected but not received.

In case of measurements as shown in figure 2 a retransmission of packet P_1 can possibly be suppressed since we

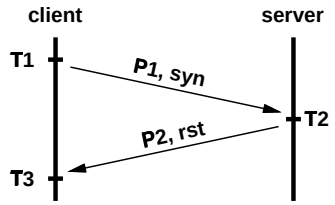

Fig. 3. Measurement by addressing a closed TCP port.

have control over the client. We just have to make sure to send this packet exactly once and not let the operating system apply the usual retransmit rules. But we cannot suppress the retransmission of packet P_2 if it is lost in the network. Figure 4 shows what happens in this case: the client only sees the retransmitted packet P_2' at the time $T_4 = T_1 + RTT + RTO$. If not accounted for, this would lead to a false value RTT' for the round-trip-time:

$$RTT' = T_4 - T_1 = RTT + RTO \tag{2}$$

Also the result for packet loss would be wrong.

This problem can be overcome if RTO is known and if it is significantly larger than usual values of RTT: whenever $RTT' = T_4 - T_1 > RTO$ then assume that the packet has been retransmitted, treat it as lost and ignore RTT'.

D. Estimating the TCP retransmission timeout

Unfortunately RTO is not a known value. The TCP RFC [6] and the host requirements RFC [7] say that RTO starts with a fixed value, $RTO_{initial}$ and then approximates the value

$$RTO = A + 4\,D \tag{3}$$

with A being the round-trip-time as measured by TCP itself and D being the mean deviation of the round-trip-time.

This is not necessarily a problem for our measurements since we only use the start of a TCP connection where typically $RTO = RTO_{initial}$ which is suggested by the RFC to be 3 seconds.

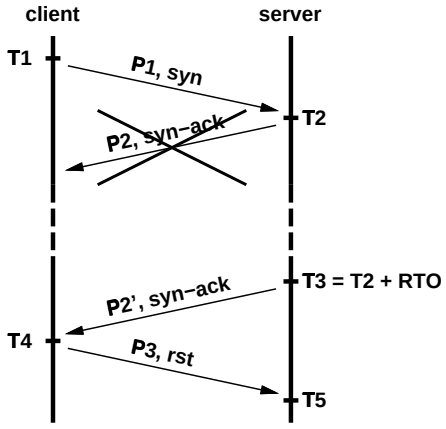

Fig. 4. Loss and retransmission of second packet.

However not all TCP implementations use the suggested value of 3 seconds for $RTO_{initial}$. Some use 1 second or any other value. Also TCP implementations are allowed to cache the A and D estimators between successive TCP connections to the same host or network. Thus in a few cases even very small values of RTO can be observed.

To cope with this situation we use the following considerations to estimate RTO for each host. The lowest RTO value is effectively used:

- A priori the RTO is assumed to be 3 seconds.
- If detectable retransmissions are observed, the lowest RTT of these packets is considered an upper bound for RTO for this host. Detectable retransmissions happen when only packet P_3 is lost, so that the server retransmits P_2 even though the client has successfully received P_2. This is illustrated in figure 5.
- An empirical algorithm is applied to tell retransmissions from a histogram of the RTT distribution. The result is another upper bound for a host's RTO. The algorithm is outlined in appendix C.

If any of these estimates yields an RTT of less than one second, the host is omitted from the measurements due to an unsuitable TCP implementation[1].

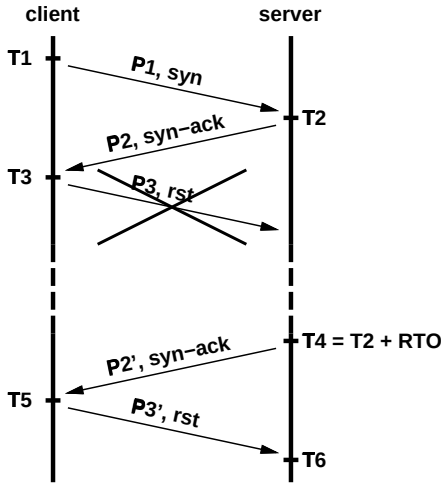

Fig. 5. Loss of third packet.

E. Implementation

A measurement tool has been implemented that uses all the methods mentioned above. It uses an architecture similar to the UDP based measurement tool presented in [1] which had proven to work well. This architecture is depicted in figure 6. The tool consists of two parts:

- The first part is called `tping` and was implemented in C. It is responsible for sending single TCP-syn packets to a given list of hosts at a controlled rate and using Poisson sampling as required by the IPPM framework RFC [8]. It uses a `raw socket` of the Unix API to send the TCP-syn

[1]Note that this means that the TCP implementation is "unsuitable" for our measurements. This does not necessarily mean that it is a bad TCP implementation in general.

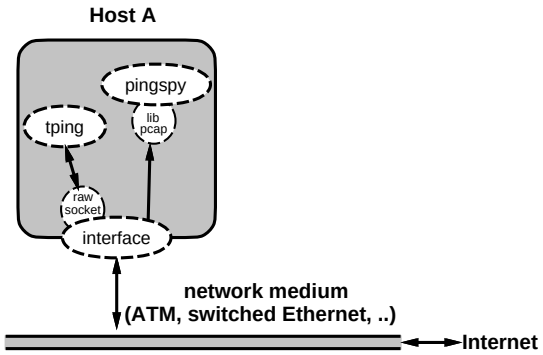

Fig. 6. Architecture of the measurement tool.

packets without involving the TCP stack of the operating system. This also solves the issue of terminating all half-opened connections: whenever a packet P_2 returns from a measurement, it is also received by the local TCP stack. Since our TCP stack does not know anything about this TCP "connection", it immediately answers with a TCP-rst packet exactly as described in section II-B.

- The second part, called `pingspy`, is responsible for observing all measurement packets that are sent or received. It saves timestamps for them, correlates all received packets with the appropriate TCP-syn packets sent by `tping` and generates the output of the measurement. `pingspy` is implemented in C++ and uses the library `libpcap` [9] for capturing relevant packets in an operating system independent way.

III. Composing host lists

As mentioned in the introduction, we want to determine the top N TCP-servers used by a given user group. Such a "user group" can be, for example, all the students or the scientists of a university, all employees of a company, the customers of an ISP, etc. We now have to observe their Internet traffic and count the amount of traffic they exchange with servers on the Internet.

A. Traffic monitoring

There are various tools for monitoring traffic. If all users share a single shared network medium like Ethernet or FDDI, tools like `tcpdump` or `snoop` could be used. If the user group is distributed over separated network segments, other tool can help such as `NeTraMet` [10] which support an architecture with distributed "meters", "meter readers" and a central "manager". A very efficient solution can be found if there is a router between the user group and their connection to the Internet and if this router supports a suitable accounting mechanism itself. An early scheme for accounting traffic in routers is the so-called "IP accounting". With IP accounting the router maintains a matrix for the traffic (counted in packets and bytes) between any observed pair of hosts or networks. This matrix can be read by an external management workstation to further process this data. IP accounting has to major disadvantages:

- It only contains information about IP addresses. It does not say anything about the protocols and service ports involved.
- Maintaining the traffic matrix in the router consumes a lot of memory, often a rare and expensive resource in routers. Mainly due to this issue many network administrators do not want their routers to do IP accounting.

Fortunately many routers nowadays support something called "NetFlow accounting" [11]. With NetFlow accounting a router counts packets and bytes for all packets with matching source and destination IP addresses, transport protocol and TCP/UDP port numbers (if applicable). Such a collection is called a "flow". The collected information is not kept in the router for a long time, but exported to a management workstation as soon as possible (or after a certain timeout).

The University of Cologne has a router at its network's edges that does use NetFlow switching and NetFlow accounting. We therefore use this method to efficiently monitor all the university's Internet traffic.

B. Determining server ports

Although NetFlow accounting provides a lot of useful information for each flow, it does not tell which side of a flow is the server and which one the client. Thus we need some extra examination in order to determine the most active server hosts from NetFlow accounting.

With both TCP and UDP, servers and clients are usually distinguished by the fact, that servers use fixed port numbers whereas clients use dynamically assigned port numbers. For example, HTTP servers use TCP port 80, telnet uses 23 and DNS servers use UDP port 53.

The Internet Assigned Numbers Authority maintains a list of port numbers that are assigned to particular services [12]. Using this as a first approach would mean to assume that all assigned port numbers are server ports and all others are client ports. However, this approach often fails in practice since there are services that are frequently offered under alternative port numbers (for instance HTTP and IRC) and there are always new services and protocols coming up and becoming popular long before they are officially documented.

As a solution, an automatic procedure has been developed that assigns a weight to each port number describing its *relative importance* as server port. This weight is calculated anew each day from observed traffic, thus taking the growing importance of new protocols and the decreasing importance of unpopular ones into account. When analyzing a flow for its server- and client-side we just compare the weight for both port numbers and consider the port with the higher weight to be the server port.

The weight of a port number is equivalent to the number of flows it attracts. A flow is defined as $(A_{src}, A_{dst}, prot, port_{src}, port_{dst}, N_{pkt}, N_{oct})$ with

A_{src} : IP address of source host,
A_{dst} : IP address of destination host,
$prot$: transport protocol (TCP, UDP, ICMP, ...),
$port_{src}$: port of transport protocol at source,
$port_{dst}$: port of transport protocol at destination,
N_{pkt} : number of packets in the flow and
N_{oct} : number of bytes (octets) in the flow.

The elements $port_{src}$ and $port_{dst}$ are only defined if the transport protocol is TCP or UDP.

The algorithms for the weights of port numbers uses symmetrical matrices B. Their elements $B_{port_{src},port_{dst}} = B_{port_{dst},port_{src}}$ contain the number of all observed flows between these two ports. A matrix B_{TCP} is used for TCP and B_{UDP} for UDP. Their elements are initially set to zero and then the algorithm shown in figure 7 is applied.

- For each flow $(A_{src}, A_{dst}, prot, port_{src}, port_{dst}, N_{pkt}, N_{oct})$:
  - If $prot$ = TCP:
    - $B_{TCP\,port_{src},port_{dst}} \leftarrow B_{TCP\,port_{src},port_{dst}} + 1$
    - $B_{TCP\,port_{dst},port_{src}} \leftarrow B_{TCP\,port_{src},port_{dst}}$
  - If $prot$ = UDP:
    - $B_{UDP\,port_{src},port_{dst}} \leftarrow B_{UDP\,port_{src},port_{dst}} + 1$
    - $B_{UDP\,port_{dst},port_{src}} \leftarrow B_{UDP\,port_{src},port_{dst}}$
- For $B = B_{TCP}, B_{UDP}$:
  - $z \leftarrow Z_{max}$
  - While B still has rows and columns and $z > 0$:
    - Set m so, that $S_m = \sum_i B_{m,i}$ becomes maximum.
    - Output of (m, S_m).
    - Remove row and column m from B.
    - $z \leftarrow z - 1$

Fig. 7. Algorithm for calculating the weight as server port.

This algorithm also uses a maximum Z_{max} for the number of server ports. This was introduced when we observed that the algorithm otherwise had a fatal dependency on user behavior: there were so called "port scans" that inevitably lead to on overload situation on the machine performing the traffic analysis. This is possible because port scans probe a huge amount of different port numbers in a short time and thereby excessively inflate matrix B. A value of 2000 for Z_{max} is reasonable, because we don't intend to do measurements to more than 2000 hosts anyways.

The algorithm has been implemented in Perl with some optimizations to control the size of B and to avoid unnecessary recalculation of the sum $S_m = \sum_i B_{m,i}$.

C. Determining server hosts

Using traffic monitoring as described in section III-A and server port detection as presented in section III-B the top N TCP-servers can be determined with the algorithm shown in figure 8 with U_K being the set of IP addresses of a given user group (here the University of Cologne) and $C_{TCP,A,port}$ and $C_{UDP,A,port}$ being counters for the amount of traffic that server A processed on its port $port$ for the user group.

This algorithm has been implemented in Perl, too.

- For each flow $(A_{src}, A_{dst}, prot, port_{src}, port_{dst}, N_{pkt}, N_{oct})$:
  - If $A_{src} \in U_K \wedge A_{dst} \notin U_K \wedge S_{port_{src}} \leq S_{port_{dst}}$:
    - $C_{prot,A_{dst},port_{dst}} \leftarrow C_{prot,A_{dst},port_{dst}} + N_{oct}$
  - If $A_{src} \notin U_K \wedge A_{dst} \in U_K \wedge S_{port_{src}} \geq S_{port_{dst}}$:
    - $C_{prot,A_{src},port_{src}} \leftarrow C_{prot,A_{src},port_{src}} + N_{oct}$
- With $prot = TCP, UDP$:
  - For all A with $\sum_{port} C_{prot,A,port} > 0$,

sorted after this sum:
  - Set m so, that $C_{A,m}$ becomes maximum.
  - Output of $\left(A, m, \sum_{port} C_{prot,A,port}\right)$.

Fig. 8. Algorithm for determining the top N server hosts.

IV. Experience

The Computing Center of the University of Cologne is continuously conducting measurements based on the UDP-echo method [1] since May 1997. TCP-syn based measurements as described in this paper have first been tested in late summer 1998 and are continuously performed since July 1999. All the measurements are done every hour to about 1000 Internet hosts. This number was temporarily increased when new algorithms were tested.

These large-scale measurements have not always remained unnoticed by system administrators. Even though our methods for measurements have been carefully designed not to do any harm to affected hosts, from time to time system administrators get the impression to be "attacked" and write corresponding complaints. The reason is that they employ some kind of firewall that meticulously logs every packet that does not correspond to a "desired access". Usually they are satisfied when the nature of the measurements is explained by e-mail. In some cases they allow us to continue the measurements, in other cases they still ask us to stop the measurements just to be easy on their log files. This is done by maintaining an exception list that is used in conjunction with the host selection algorithm and contains hosts or networks that may never be used for measurements.

From the beginning of the measurements in 1998 up to January 2000, 25 complaints have been received. In 21 cases administrators complained about UDP-echo based measurements and in 4 cases they complained about TCP-syn based measurements. This is somewhat odd since the UDP-echo based measurements use a well-known protocol for exactly what it is meant for: simply echoing simple packets for diagnostic purposes. The TCP-syn based measurements actually abuse TCP's connection establishment mechanisms for something they are not meant for. And technically they are not far from what is a well-known denial-of-service attack: TCP-syn flooding[2]. Furthermore, complaints about UDP-echo based measurements continue to arrive from time to time regardless of all improvements

[2]The difference between our measurements and TCP-syn flooding attacks is twofold: 1) the rate is much lower and 2) all half-opened connections are closed as described in section II-B.

in the host selection method whereas the complaints about TCP-syn based measurements always could be traced back to one of two problem areas:

- Some FTP servers act different from normal TCP based servers[3] by even detecting single TCP-syn packets and logging them into their log files. This only occurred with an earlier version of the host selection tool that explicitly forced FTP servers to be measured on port 21 (ftp-control) instead of 20 (ftp-data). This explicit port substitution had been implemented in order to account for the fact that FTP accepts new sessions on port 21 and opens port 20 only when needed. However experience has shown that it is wiser to let the ports as they are, thus using port 20 for FTP servers.
- Sometimes the algorithm for detecting TCP servers fails and gives a host/port combination that actually does not stand for a real service. This can happen because some fraction of all TCP connections is not client/server-oriented but merely based on a peer-to-peer concept. IRC-DCC and some FTP serves are examples for peer-to-peer connections with dynamic port numbers on both sides. TCP-syn packets for a port that is not in use (but has been used for a long peer-to-peer connection some hours or days before) are sometimes detected by security software and reported to the administrator as "attack". This problem can be avoided by not using the host selection algorithm too aggressively, i.e. by not trying to detect too many TCP servers. "Only" using the top 1000 TCP servers proved to be a good and safe choice in our environment.

When system administrators complain about "suspicious activity" or "attacks" they use an amazing variety of methods to address their complaint:

- Some simply write e-mail to common mailboxes like `root`, `postmaster`, `webmaster` or `abuse` at the host that sent the packets. Sometimes they don't use the full hostname but merely the corresponding second-level domain, e.g. `root@uni-koeln.de`
- In other cases they do a lookup in the RIPE database [13] and write e-mail to all zone-, admin- and technical contact persons.
- Or they inspect the start of authority (SOA) record in the DNS and write e-mail to the corresponding `hostmaster` address.
- In one case, no network database had been used at all. Instead a complaint was sent directly to the rector of the university.

In an attempt to easy the issue of usefully addressing complaints and questions regarding the measurements, several measures have been taken:

- First a dedicated web page was created that explains the nature of the measurements and directs complaints and questions to a useful e-mail address. It also shows a summary of current results of the measurements. Of course, this does not help unless system administrators are given a hint to look at this web page.
- Then a TXT record for the measurement host was installed in the DNS giving a short explanation and a link to the web page mentioned above. This was practically to no avail since nobody appears to look for TXT records when investigating a particular host.
- Consequently a dedicated IP address was assigned for the measurements and it was given an expressive name: `measurement-for-thesis.rrz.uni-koeln.de`.
- Eventually, a virtual web server was installed at the measurement host that directly leads to the web page when addressed with the dedicated IP address.

All these measures together have the effect that most complaints are now addressed to the e-mail address mentioned on the dedicated web page. But some still use random methods.

In a few cases it was suggested to us to ask for permission prior to doing measurements. While this sounds worthwhile at first glance, it is practically infeasible. On the one hand, asking for permission contradicts the goal of unbiased selection of hosts by automated methods, and on the other hand it would lead to sending thousands of e-mails according to dynamically changing host lists. Such mass-mailings would not help either side.

V. Example results

A. Connectivity

Figure 9 shows an example plot of connectivity similar to the one on the web page. It show hourly connectivity obtained by three different measurements: TCP-syn based measurements through two different providers and UDP-echo based measurements (only provider 1). The UDP-echo based "connectivity" data is generally much lower than TCP-syn based data because this UDP-echo based measurement technique does not allow reliable detection of hosts that are unavailable for measurements. There are many hosts that simply do not answer UDP-echo requests in any way.

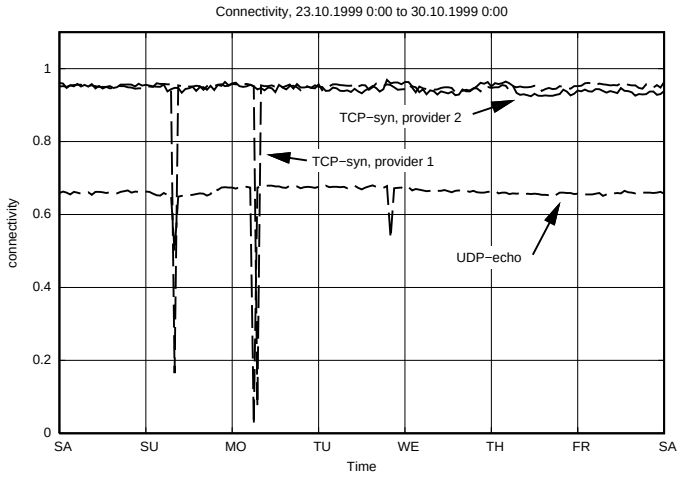

Fig. 9. Example plot for connectivity.

The figure also shows that there were short breakdowns of connectivity for provider 1 on Sunday and Monday

[3]Usually the operating system waits until TCP's three-way-handshake is complete and then hands the new connection to the server application. This way the application has no chance of detecting single TCP-syn packets.

morning. This can be seen from the TCP-syn based measurements as well as from the UDP-echo based ones. And it shows that on Thursday morning something had happened that affected the connectivity between provider 2 and a small part of the Internet. This reverted to normal after a couple of days. Probably this drop in connectivity was not bad enough to instantly alarm the responsible network operators; they needed several days to detect the problem and resolve it.

B. Application layer performance

What Internet users are actually interested in is the performance of the *applications* they use. Thus it is worthwhile to investigate how network layer metrics affect the performance of the application layer. Due to the huge variety of applications being used on the Internet we cannot analyze this relationship for every single application but just for some *classes* of applications.

One example is TCP bulk transfer which is well-suited for describing transfers of large files with FTP or HTTP or sending of large messages with SMTP or NNTP. It already has been analyzed earlier [1] but with the new measurement methods and with new TCP models by Padhye et al. [2], [3] it can now be done with better accuracy and an increased range of validity.

Padhye et al. found the TCP throughput in dependency of packet loss p, round-trip-delay RTT, initial retransmit timeout T_0, maximum segment size MSS and maximum TCP window size W_{max} to be approximately

$$TP = \begin{cases} \frac{MSS\left(\frac{1-p}{p}+1/2\,w(p)+Q(p,w(p))\right)}{RTT\,(w(p)+1)+\frac{Q(p,w(p))G(p)T_0}{1-p}} & \text{if } w(p) < w_{max} \\ \frac{\frac{MSS}{RTT}\left(\frac{1-p}{p}+1/2\,w_{max}+Q(p,w_{max})\right)}{\left(\frac{w_{max}}{4}+\frac{1-p}{p\,w_{max}}+2\right)+\frac{Q(p,w_{max})G(p)T_0}{1-p}} & \text{otherwise} \end{cases} \tag{4}$$

with

$$\begin{aligned} w_{max} &= W_{max}/MSS \\ w(p) &= 2/3+2/3\sqrt{3\,\frac{1-p}{p}+1} \\ Q(p,w) &= \min\left(1, \frac{\left(1-(1-p)^3\right)\left(1+(1-p)^3\left(1-(1-p)^{w-3}\right)\right)}{1-(1-p)^w}\right) \\ G(p) &= 1+p+2\,p^2+4\,p^3+8\,p^4+16\,p^5+32\,p^6 \end{aligned} \tag{5}$$

Using this model and using the TCP-syn based measurements of the working days of week 37/1999, figure 10 shows the distribution of TCP bulk transfer rates the network would allow through two different providers at two different times of day. The other parameters of the model have been obtained on a per-host basis by monitoring traffic of the university's WWW proxy server with `tcpdump`.

In this example, provider 1 offers excellent transfer rates to a certain part of the Internet, but severely breaks down for the largest part of the Internet during working hours. Provider 2 does a better job in giving decent performance to a large part of the Internet even during peak hours.

Depending on a users needs this can be useful to tell which provider offers the better service: if the user needs particularly high data transfer rates to certain partners and if these partners happen to be well-supported by provider 1, then this provider would be the better choice. This might be the case for many scientific applications. If the user needs to achieve decent results when accessing random servers on the Internet during working hours, as it is true for web browsing and most other general applications, then provider 2 would be preferred.

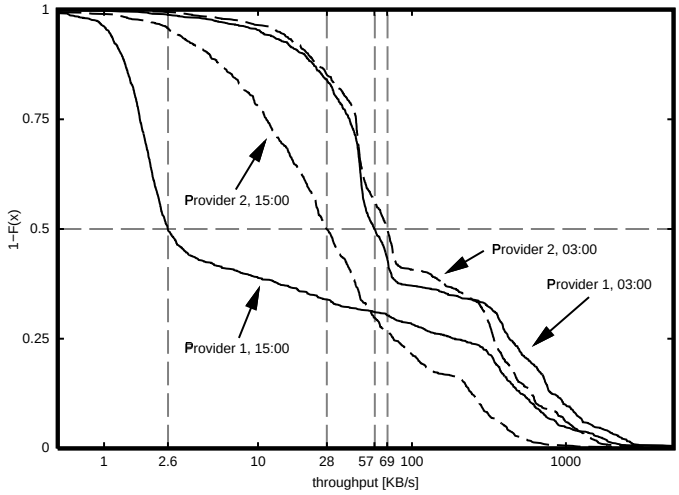

Fig. 10. Distribution of TCP bulk transfer rates through different providers at different times of day.

VI. Conclusions

Our goal was to develop methods for measuring network layer metrics on a large scale and to provide a framework that makes such large-scale measurements feasible in practice. In this paper we introduced a TCP-syn based method for performing network layer measurements and a NetFlow based system for compiling suitable host lists as an input for active measurements. We presented our implementation of these methods and the experience we obtained by actually using them for large-scale measurements over a long period of time.

We also gave an example how results of these measurements can be used to answer application level performance questions with the most recent model for TCP bulk transfer and how this can be useful when making business decisions for deals with competing Internet service providers.

Even though the introduced methods proved to be feasible we also saw that they should not be applied uncritically and that they require certain resources (sufficiently large user base, static IP addresses, control over DNS and RIPE database objects). Thus they are not suitable to be used by the average end user. Only larger companies, universities and probably Internet service providers themselves might be able to use these methods in a sensible way. Or, in other words: "Don't try this at home!"

Appendices

I. Traffic analysis

In order to get an idea of how many hosts make a serious contribution to the Internet traffic of a typical user group we monitored the Internet traffic of the University of Cologne. The result is given in figure 11. It shows the fraction y of the university's Internet traffic which can be attributed to the x most important hosts. The first 132 hosts account for 50 % of the traffic and 7113 hosts account for 90 %. It indicates that it might be worthwhile to use some thousand hosts in order to represent the Internet and to cover the largest part of the traffic. One thousand hosts already cover more than two thirds.

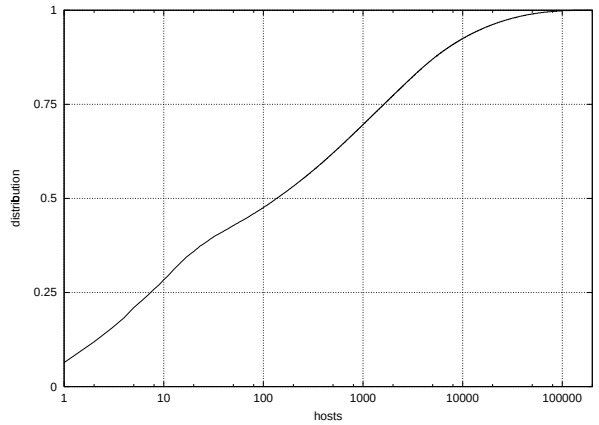

Fig. 11. Traffic distribution over hosts.

The data for figure 11 is based on the traffic of May 1997.

II. Accuracy of loss and delay measurements with the TCP-syn and the UDP-echo based approach

Since we have two completely different methods (TCP-syn and UDP-echo based) for measuring the same network metrics (round-trip-delay and loss) we can compare them in order to analyze particular strengths and weaknesses.

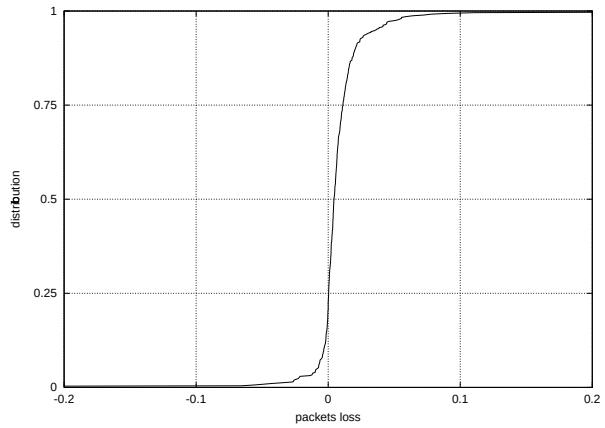

Fig. 12. Distribution of loss increase for TCP-syn measurements.

For each host that was subject to both measurement techniques we compared the results of the TCP-syn based and the UDP-echo based measurements. We calculated this difference for each hour and averaged them for each host. The data presented here is based on measurements of January 2000.

The result for packet loss is shown in figure 12: the x-axis gives the difference of TCP-syn based and UDP-echo based packet loss, positive numbers indicating that TCP-syn based measurement led to larger loss values. The y-axis gives the distribution of hosts. Ideally there would be a step function from $y = 0$ to $y = 1$ at $x = 0$. However, due to the omnipresent variation and uncertainties we can expect the function to be significantly dampened. The figure shows that some hosts yield a higher packet loss when using the TCP-syn method. This can be explained with overload conditions in server hosts: the operating system stops answering TCP-syn packets when there are too many connections that have been opened but not yet been accepted by the server application. UDP-echo packets are handled by different parts of the system and might still be answered in such a situation. It is a good design goal of all server operators to avoid such overload situations, but apparently they happen on some hosts.

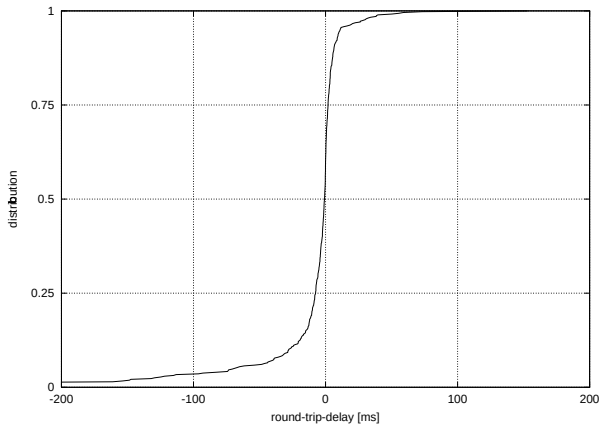

Fig. 13. Distribution of delay increase for TCP-syn measurements.

Figure 13 shows the result for round-trip-delay. The x-axis shows, in milliseconds, how much delay increases when using measurements based on TCP-syn instead of UDP-echo packets. We can see that some hosts actually yield a larger delay when measured with the UDP-echo based method. This can probably be explained by the fact that UDP-echo packets are usually answered by a user-level process called `inetd`. If a host is overloaded with respect to processing time or context switches there might a significant delay when handing a packet from the kernel to this process and handing the answer packet back. TCP-syn packets on the other hand are directly processed by the kernel without any context switch so that there are much less opportunities for additional delay.

The results can by summarized as follows:

- If a host is suitable for measurements with the UDP-echo approach at all, this approach yields the best results

for packet loss. In some cases the TCP-syn based approach yields numbers that are somewhat too high.

- Delay measurements are more accurate when done with the TCP-syn based approach. The UDP-echo based approach yields too high delay values in some cases.

III. Empirical algorithm for estimating the retransmission timeout

The empirical algorithm separates retransmissions from normal measurements by comparing the distribution of RTT values with a simple model for a "typical" distribution. We use a negative exponential distribution[4]:

$$f(x) = \begin{cases} 0 & \text{if} \quad x < c \\ a\, e^{\frac{c-x}{b}} & \text{if} \quad x \geq c \end{cases} \tag{6}$$

In this case x is the round-trip-delay, $f(x)$ is the probability density in dependency of $RTT = x$ and a, b and c are parameters of the model. The parameters can be explained as follows:

a is the maximum probability density.

b influences the area beneath the function:

$$H = \int_{x=0}^{\infty} f(x)\, dx = \int_{x=0}^{\infty} a\, e^{\frac{-x}{b}}\, dx = a\, b \tag{7}$$

In the discrete case, H is the number of measurements.

c is the minimum packet delay.

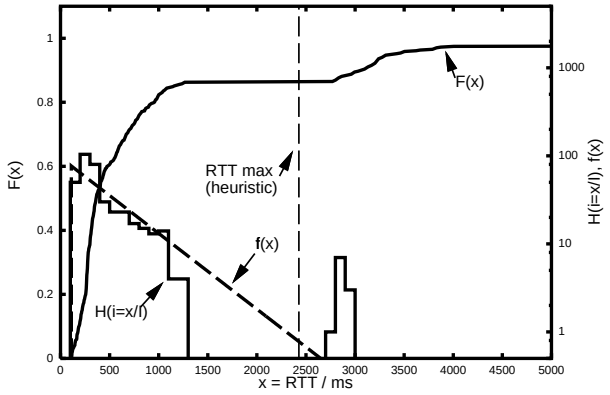

Fig. 14. Algorithm for eliminating retransmissions.

All measured RTT values are put into a histogram $H_0, H_1, \ldots H_{n-1}$ with resolution $I = 100\,\text{msec}$, the parameters of equation 6 are adapted according to the measured values and then histogram and model are compared as shown in figure 14. The cluster of normal measurements is considered to end where $H(i = x/I)$ drops significantly below $f(x)$ and the cluster of retransmissions is considered to start where $H(i = x/I)$ again raises beyond $f(x)$ (if there are retransmissions at all). Figure 14 illustrates this algorithm. It also shows the original distribution function $F(x)$. The data for this example is based on $24 \cdot 20 = 480$ measurements to one host that have been performed on one day.

[4]It is known that most Internet metrics cannot be well-modeled using a simple function like this. However, we do not need a "good" model here. We just need a rough approximation for a simple algorithm to separate two clusters of data from each other.

Acknowledgments

The research presented in this paper is part of a PhD thesis at the University of Cologne. The author would like to thank Prof. Dr. Rainer Schrader and Dr. Wolfgang Trier for their support and many fruitful suggestions, all colleagues at our computing center for their assistance, and Roland Franzen for his help with my English. Special thanks to Michael Rüsel from NetCologne GmbH (the major local Internet provider in Cologne) for letting me perform measurements through an additional Internet access point, making the results much more interesting.

References

[1] Martin Horneffer, "Methods for performance-analysis of internet access points," in *Proceedings of the TERENA Networking Conference '98*, vol. 30 of *Computer Networks and ISDN Systems*, pp. 1607–1615. Elsevier Science, September 1998.

[2] Jitendra Padhye, Victor Firoiu, and Don Towsley, "A stochastic model of tcp reno congestion avoidance and control," Tech. Rep., Dept. of Computer Science, University of Massachusetts, 1999.

[3] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. SIGCOMM*. ACM, September 1998.

[4] Havard Eidnes, "Subject: Re: Icmp and future testing," Note sent to IPPM mailinglist, Fri, 04 Dec 1998 19:22:05 +0100, Archived at `http://www.advanced.org/IPPM/archive/0612.html`.

[5] Richard W. Stevens, *TCP/IP Illustrated*, vol. 1: the protocols, Addison-Wesley, 1994.

[6] Jon Postel, "RFC 793: Transmission Control Protocol," 1981.

[7] R. Braden, "RFC 1122: Requirements for internet hosts – communication layers," 1989.

[8] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "RFC 2330: Framework for IP performance metrics," February 1998.

[9] Steve McCanne, Craig Leres, and V. Jacobson, "`libpcap` — packet capture library," Documentation and source available via ftp from `ftp.ee.lbl.gov`.

[10] Nevil Brownlee, "NeTraMet web page," `http://www.auckland.ac.nz/net/NeTraMet/`.

[11] Cisco Systems, "NetFlow Switching Overview," `http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/switch_c/xcprt3/xcovntfl.htm`.

[12] Internet Assigned Numbers Authority, "IANA Protocol/Number Assignments Directory," `http://www.iana.org/numbers.html`.

[13] "Réseaux IP Européens (RIPE)," `http://www.ripe.net/info/ripe/ripe.html`.