Measuring Network Parameters with Hardware Support

Kanchan Bhargava, Sotirios G. Ziavras and Roberto Rojas-Cessa
Department of Electrical and Computer Engineering
New Jersey Institute of Technology
Newark, NJ 07102, USA
ziavras@njit.edu

Abstract
For good quality of service (QoS) in a computer network, robust means of measuring essential network parameters are required. Much research has been done in this area and a multitude of options are available to perform this task. This report details our hardware implementation of three such schemes on FPGAs (Field programmable Gate Arrays). The first scheme calculates the packet loss and one-way network delay using RTP over UDP; the second one uses the concept of TOPPs (Train of Packet Pairs) to measure the available bandwidth, and the third scheme is a partial implementation of the TCP protocol, which can be developed further to measure various network parameters. Our analysis shows that these hardware implementations can carry out efficient parameter measurements at reasonable hardware cost. In addition, experiments are used to compare three hashing schemes for storing TCP tuple information; real benchmarks were used.

1. Introduction

The QoS of a network is associated with its end-to-end communication performance. Some of the common parameters that determine the QoS of a network and the required overall efficiency are the end-to-end delay, jitter, packet loss, bottleneck bandwidth, available bandwidth and so on. However, a problem arises in the statistical analysis of these parameters due to the bursty nature of Internet traffic. Thus, before any type of such analysis is carried out, the traffic model has to be determined. For example, variations of the Markov model, such as the Gilbert model, the extended Gilbert model, the Bernoulli Loss model, etc. have been presented in [1], [2] and [3].
There are two basic QoS architectures prevalent in current network topologies:
(a) Differentiated Services (DiffServ) - The DiffServ architecture model (RFC 2475) divides the traffic into a small number of classes and allocates resources on a per-class basis. Since DiffServ has only a few classes of traffic, a packet's "class" can be marked directly in the packet.
(b) Integrated Services (IntServ) – In the IntServ architecture model (RFC 1633), a signaling protocol is required to tell the routers which flow of packets requires special QoS treatment. This does not provide as much QoS scalability as the DiffServ model, but it allows for a tighter control of real-time traffic.

The layout of this paper is as follows. Section 2 details related work and the implementation of the packet loss and network delay measurement schemes. Section 3 is about the implementation of the available bandwidth measurement scheme. Section 4 describes the TCP implementation and presents experimental results for three hashing schemes. Future work and conclusions are discussed in Section 5.

2. Packet Loss and One-way Network Delay Measurement

2.1 Related work

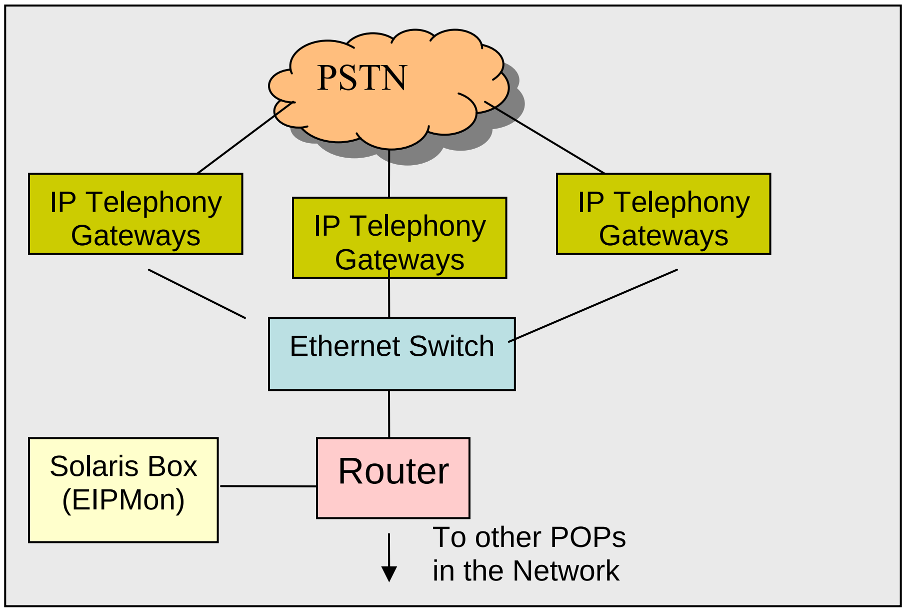
Various models are presented in the literature to depict the packet loss and network delay. [4] uses a two-state Hidden Markov model (HMM) to simulate the packet loss. A probabilistic two-state Gilbert model to simulate frame deletion based on the packet loss, burst loss and jitter that results in loss is used in [5]. In [6], an algorithm is used that works on traces captured in middle points of the network. The algorithm observes packet sequence numbers, and estimates the most likely events and TCP states that might lead to a particular sequence number pattern.

Packet loss is measured in the downlink direction, where most of the data is flowing. The basic idea in [7] is to maintain a packet loss history. Packets are distinguished based on WRED (weighted random early detection). This is an extension to the Random Early Detection (RED) algorithm [31]. The loss of probe packets in [9] is modeled like a logical multicast tree using a set of mutually independent Bernoulli processes, each operating on a different link. Thus, losses are independent for different links and different packets. The algorithm [10] used in our implementation is discussed in the next sub-section.

2.2 Measurement Algorithm for Packet Loss and Network Delay

This scheme in [10] uses the Enhanced IP Network Monitoring tool (EIPMon) that estimates one-way end-to-end network delay variation and packet loss between POPs (Points of Presence sites). EIPMon is a software tool that runs on Solaris machines. Each POP consists of a number of IP telephony gateways connected by a fast Ethernet switch to a router. For redundancy, each POP has a backup Ethernet switch and router. The router connects the POPs to the rest of the EIP network, while the IP telephony gateways connect the POPs to the Public Switched Telephone Network (PSTN). This scenario is illustrated in Fig. 1.

Fig 1. POP (Points of Presence) site



Most of the traffic between POPs is Simple Network Management Protocol (SNMP) traffic - H.323 signaling traffic being carried over TCP/IP; real-time, “audio” application (voice or fax) traffic is being carried over RTP/UDP/IP. EIPMon, which is a passive tool, listens to the existing network traffic in order to make its network QoS measurements. It does not generate its own traffic. The steps involved in the measurement are as follows:

1. Each instance of EIPMon periodically puts one of the Ethernet interfaces into the “promiscuous mode” to capture a trace of all the Ethernet packets passing through its POP.
2. The monitors are synchronized so they all capture packet traces for the same sampling periods.

3. All packet traces are live voice or data traffic. Each network packet contains an RTP (Real-time Transport Protocol) header at the beginning of its UDP payload. EIPMon uses the information in this header to estimate the actual QoS delivered by the network to the real-time application traffic.
4. *Processing steps:*
(a) The monitor first separates packets based on whether they are destined for one of the local telephony gateways in the WAN or leaving the POP where the monitor is running. All incoming packets not headed for a local gateway are rejected.
(b) Further selection is made of RTP packets. All non-RTP packets are discarded.
(c) Packets are grouped into the same RTP flow if they have the same source IP address, source UDP port number and RTP Synchronization Source (SSRC) number.
(d) Once the RTP flows have been separated, an estimation is made of the number of packets lost per flow by looking for gaps in RTP sequence numbers.
(e) The monitor then aggregates the RTP flow packet loss counts and network delay variation measurements by source POP, computes the loss and delay variation statistics and writes them to the output report for that sampling interval.
5. EIPMon derives QoS statistics from the real-time flows coming into the local POP using the information contained in the packet RTP headers. The RTP header fields of interest to EIPMon are : V, the RTP version; PT, the RTP payload type; SSRC, the synchronization source identifier; the sequence number, the RTP sequence number; and the time stamp, the sampling instant of the first octet in the RTP packets, as determined by a clock that get incremented monotonically and linearly with time. The design flow for this measurement algorithm is shown in Fig. 2.

The EIPMon's packet capture facility is based on the Solaris' STREAMS-based kernel packet filter. The principal source of error in counting lost packets is due to packets that are lost from either the beginning or the end of the flow during the EIPMon traffic collection period. EIPMon has no way of knowing if any such packets existed, and therefore cannot count them as lost. The acceptable peak sustainable packet arrival rates are as follows: for 1500-byte packets, 100 Mbps; for 200-byte packets , 30 -40 Mbps; and for 64-byte packets, up to 3 Mbps.

2.3 Hardware Implementation
The block schematic of our hardware design for the implementation of this scheme is shown in Fig 3. As mentioned previously, the EIPMon tool captures the Ethernet frames for a pre-determined time interval. The information required for packet loss and one-way delay calculation can be obtained by examining the IP, UDP and RTP headers that are part of the Ethernet frame payload. However, in this implementation it is assumed that there is another physical interface (for e.g. an FPGA) that will perform the task of retrieving this information from the Ethernet payload. Hence, the required information, such as the port numbers and IP addresses, is available to the respective modules in the design without any direct interaction with the Ethernet frame. Let us now explain the functionality of the various modules.

- *RTP Checker Module:* This module performs two tests. Firstly, it checks whether the IP protocol field has a value of 17 (which means that the higher level protocol over IP is UDP); and secondly, it checks whether the UDP destination port number is 5004 (for RTP). The design flow proceeds further if these two conditions match.
- *Flow Generator module*: Once it is determined that the packet is for a local gateway, this module creates a flow from the incoming information. Packets with the same source IP address, source UDP port number and RTP SSRC number are sorted into a flow.
- *Packet Loss Calculator*: For an existing flow, this module keeps track of the incoming sequence numbers that are part of the RTP header. Since there is no re-transmission in the case of UDP, it marks the missing sequence numbers as belonging to lost packets and calculates the packet loss accordingly.

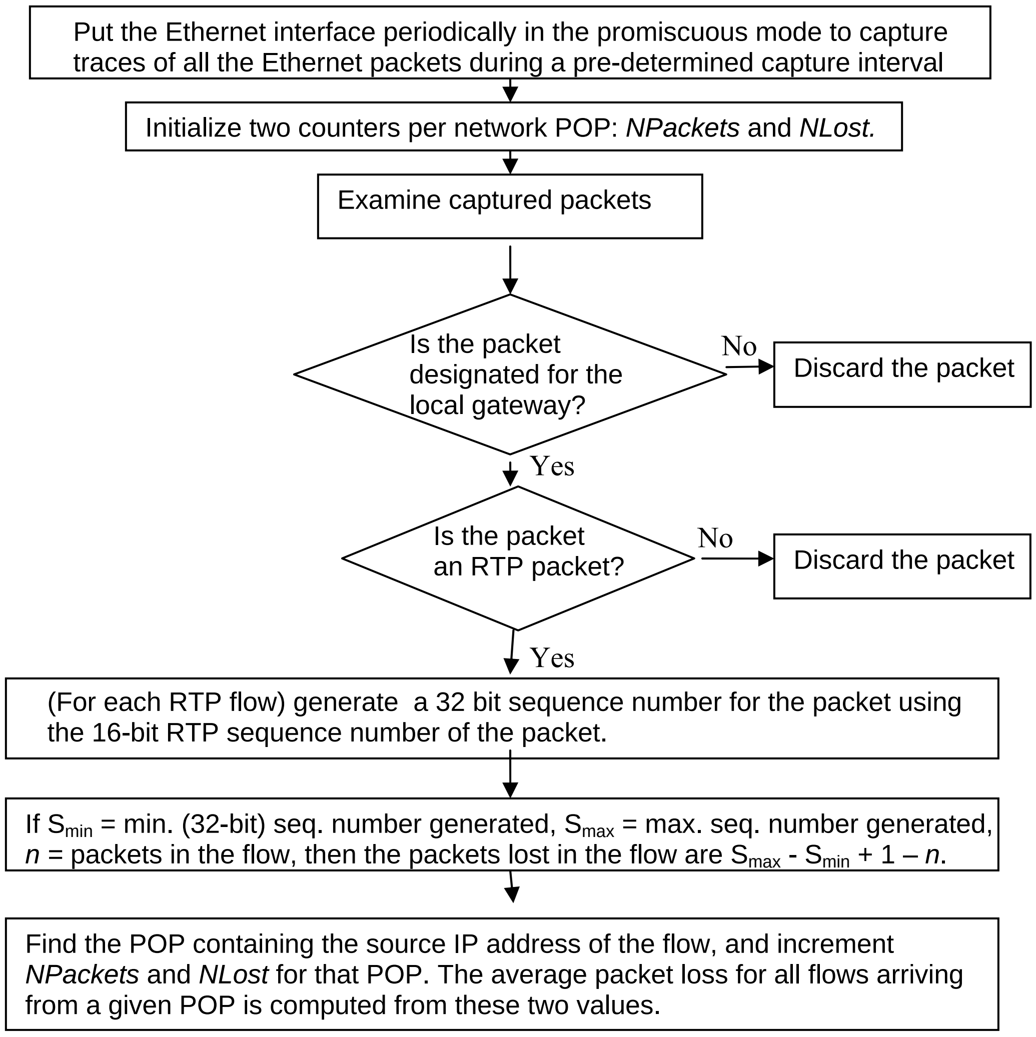

Put the Ethernet interface periodically in the promiscuous mode to capture traces of all the Ethernet packets during a pre-determined capture interval
Initialize two counters per network POP: *NPackets* and *NLost.*
Examine captured packets
Is the packet designated for the local gateway?
No
Discard the packet
Yes
Is the packet an RTP packet?
No
Discard the packet
Yes
(For each RTP flow) generate a 32 bit sequence number for the packet using the 16-bit RTP sequence number of the packet.
If S_{min} = min. (32-bit) seq. number generated, S_{max} = max. seq. number generated, n = packets in the flow, then the packets lost in the flow are $S_{max} - S_{min} + 1 - n$.
Find the POP containing the source IP address of the flow, and increment *NPackets* and *NLost* for that POP. The average packet loss for all flows arriving from a given POP is computed from these two values.

Fig 2. EIPMon measurement algorithm

- *Delay Measurement Module*: This module utilizes the timestamp field in the RTP header to calculate the one-way network delay. However, the effect of clock skew between the receiver and sender clocks is not taken into account.

Our design was written in the Verilog Hardware description Language and was synthesized using the Synplicity Pro 8.5 tool. It was mapped to the Xilinx Vertex II part XC2V6000FF1517-6 FPGA. The estimated frequency from synthesis was obtained as 158.8MHz, and that from Place and Route was 140.805MHz. The total number of clock cycles from the point the RTP checker module receives the information at the input and the packet loss is calculated is seven. Hence if a transmission rate of 10Gbps is assumed and the Ethernet frame size is 64 bytes, one packet will be transmitted in 51.2ns. If each packet is processed in seven clock cycles, it will take 7 / ($140.805 * 10^6$) seconds or 49.71ns. Thus the processing time is well within the bounds of the transmission time. The resource configuration on the Virtex II FPGA is shown in Table 1.

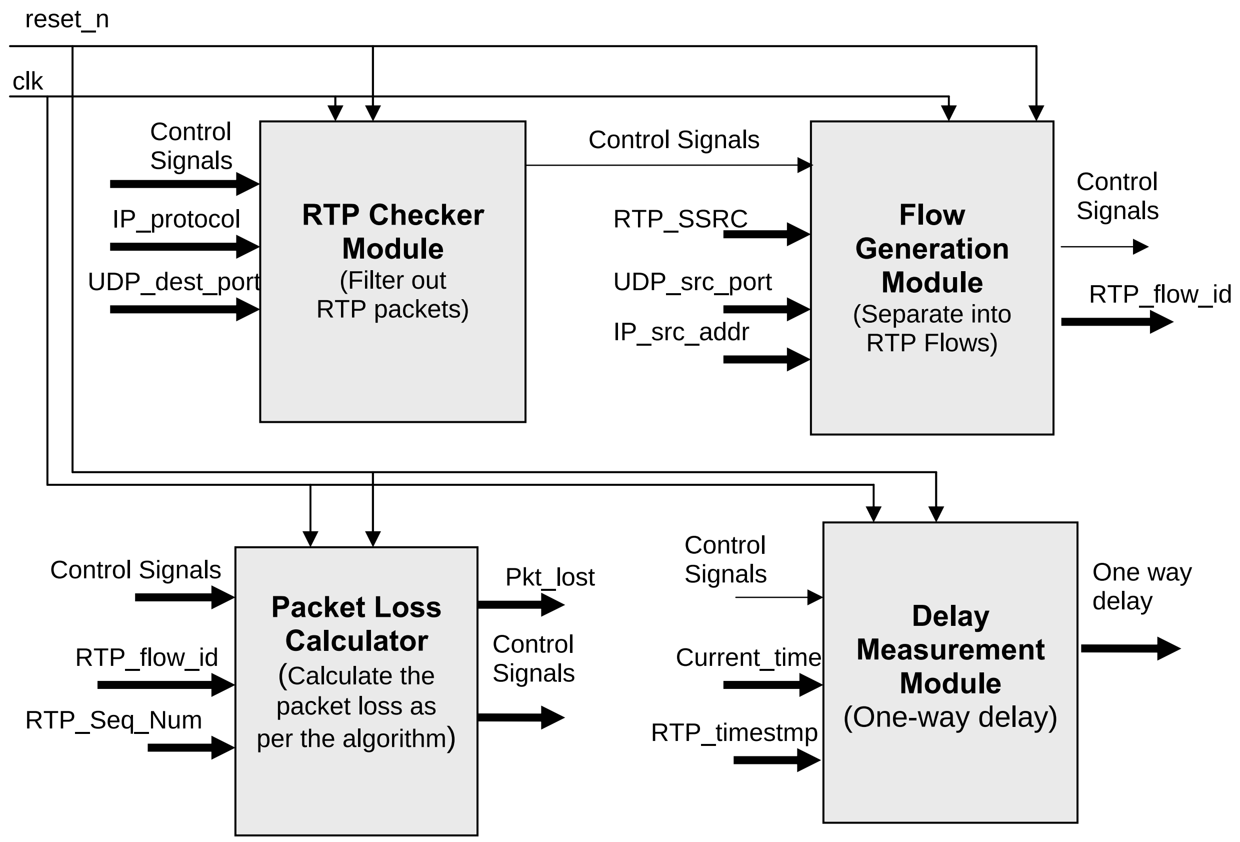


Fig. 3 Schemes to measure the packet loss and network delay

Table 1. Resource Usage Report for Packet Loss and Delay Module:

Register elements	LUTs	Multiplexers	RAM (single Port)*
676 out of 67,584 (1%)	602 out of 67,584 (0.9%)	100 out of 67,584 (0.15%)	RAM 16 X 1: 1 (out of possible 67,584) RAM 16 X 8: 6 (out of possible 8,448)

*There are 8448 CLBs (Configurable Logic Blocks) in the XC2V6000 FPGA. Each CLB has 8 LUTs (Look up Tables), and each LUT is capable of implementing a 16 X 1-bit synchronous RAM resource called a distributed SelectRAM element. Hence, within the CLB, several combinations of single and dual port RAMs can be put together. The statistics mentioned above assume a uniform RAM configuration (single or dual) for all the CLBs in the FPGA.

3. Measuring the Available Bandwidth

As described in [17], there are four major types of bandwidth estimation techniques:

- *Variable packet size (VPS)* : It estimates the capacity of individual hops. VPS measures the round-trip time (RTT) from the source to each hop in the path as a function of the probing packet size. It uses the time-to-live (TTL) field in the IP header to force probing packets to expire at a particular hop. The router at that hop discards the probing packets, returning ICMP (Internet Control Message Protocol) time-exceeded error messages back to the source. The source uses the received ICMP packets to measure the RTT to that hop.

- *Probing, packet pair/train dispersion (PPTD)*: It estimates the end-to-end capacity. The source sends multiple packet pairs to the receiver. These packets have the same size and are sent back to back. The dispersion of a packet pair at a specific link in the path helps in determining the link capacity.
- Self-*loading periodic streams (SLoPS)*: It estimates the end-to-end available bandwidth. The source sends a periodic packet stream to the receiver at a chosen rate **R**. If the stream rate **R** is greater than the path's available bandwidth **A**, the stream will cause a short-term overload in the queue of a tight link. However, if the stream rate **R** is lower than the available bandwidth **A**, the probe packets will traverse the path without causing increased backlog at any link, and therefore, their one-way delay will not increase. In SLoPS, the sender progressively attempts to bring the stream rate **R** close to the available bandwidth **A.**
- *Trains of packet pairs (TOPP):* It estimates the end-to-end available bandwidth. TOPP sends many packet pairs at gradually increasing transmission rates from the source to the receiver. Its basic idea is similar to that of SLoPS. TOPP increases the offered rate linearly, whereas SLoPS uses an iterative algorithm to adjust the transmission rate.

There are several software tools that measure bandwidth parameters based on the above techniques. Pathchar[18], Clink[19] and Pchar calculate the per-hop capacity by using VPS. Bprobe, nettimer, pathrate and sprobe measure the end-to-end capacity using packet pairs. Pathload, IGI [23] and pathChirp[22] are based on SLoPS; finally, Iperf [21], Netperf and ttcp deal with TCP connections. The scheme implemented in this paper is based on the TOPP technique which is described in detail in [20].

3.1 Measurement Algorithm for the Available Bandwidth

The TOPP measurement method has two separate phases. The first one is the active probing phase where pairs of probe packets are injected into the network. The second phase is the analysis phase, where bandwidth estimates are calculated based on the reception times of the probe packets.
In the probing phase, starting at some rate $\mathbf{o}^{\mathbf{min}}$, n well-separated pairs of equally sized probe packets are sent to the destination. After these n packets have been sent, the offered rate **o** is increased by the amount **Δo** and another set of n probe packets are sent. The rate is linearly increased each time, and this goes on until the offered rate reaches some rate $\mathbf{o}^{\mathbf{max}}$, that signals the end of the probing phase, Hence, there will be

$$n_l = \left\lceil \frac{o^{\max} - o^{\min}}{\Delta o} \right\rceil \qquad (1)$$

offered rate levels. The time separation $\mathbf{\Delta T^p}$ between two consecutive probe pairs is chosen in such a way that the probability of more than two probe packets to be queued at a node is small. This is done so that the nodes along the path do not experience long bursts of probe packets. On the receiving side, the probe packets are time stamped upon reception. Once all packets have been received (or a timeout is exceeded to deal with lost packets), the time stamps are sent back to the probing host. Hence, all measurements are carried out at the source (i.e., one way approach).
The analysis phase is based on the principle of the bottleneck spacing effect. When two packets are transmitted over a link with time separation **ΔS** and service time $\mathbf{Q_b} > \mathbf{\Delta S}$, then as the packets leave the link they will actually be separated by $\mathbf{\Delta R} = \mathbf{Q_b}$. Using the size of the packets, **b**, and the time separation **ΔR** , the experienced bandwidth across that link can then be estimated as:

$$\overline{f} = \frac{b}{\Delta R} \qquad (2)$$

If $\mathbf{Q_b} \leq \mathbf{\Delta S}$ **,** then the link could service (i.e. transmit) the packets at the rate they arrive. The outcome of the probing phase is a series of time stamps for all the received probe packets. Since there are **n** time stamp pairs for each offered rate, the mean is calculated for these **n** values. Thus, there will be one $\mathbf{\Delta R_i}$

value for every offered transmission rate, $\mathbf{o_i}$. Using these $\mathbf{\Delta R_i}$ values and the size **b** of the probe packets, the bandwidth can be estimated with Eq. 2.

3.2 Hardware Implementation

The block schematic of our hardware design for this TOPP measurement scheme is shown in Fig 4. The hardware has to be preset at both the sender and receiver ends; and it is almost identical at both ends except for the fact that the receiver does not participate in the analysis phase, and hence does not need the comparator module. The hardware modules are:

- *Probe Packet Module*: This module detects the probe packets being sent at each new rate. It records the beginning and end of the packet pairs, and monitors the number of different rates at which packet pairs are sent.
- *Packet Pair Stats Module*: This module keeps track of the intra- and inter-packet arrival intervals, and calculates the $\mathbf{\Delta S}$ (or $\mathbf{\Delta R}$ at the receiver end) values for packet pairs. It also calculates the average intra-packet delay for a given transmission rate of probing packet pairs.
- *Comparator Module*: This module is used in the analysis phase of the aforementioned algorithm, and is present only at the end where the probe packet pairs are transmitted. It basically compares the incoming value of $\mathbf{\Delta R}$ with the known value of $\mathbf{\Delta S}$, and calculates the estimated bandwidth as per Eq. 2.

This design was written in the Verilog Hardware Description Language and was synthesized using the Synplicity Pro 8.5 tool. It was mapped to the Xilinx Vertex II part XC2V6000FF1517-6 FPGA. The low resource consumption of this implementation is illustrated in Table 2.

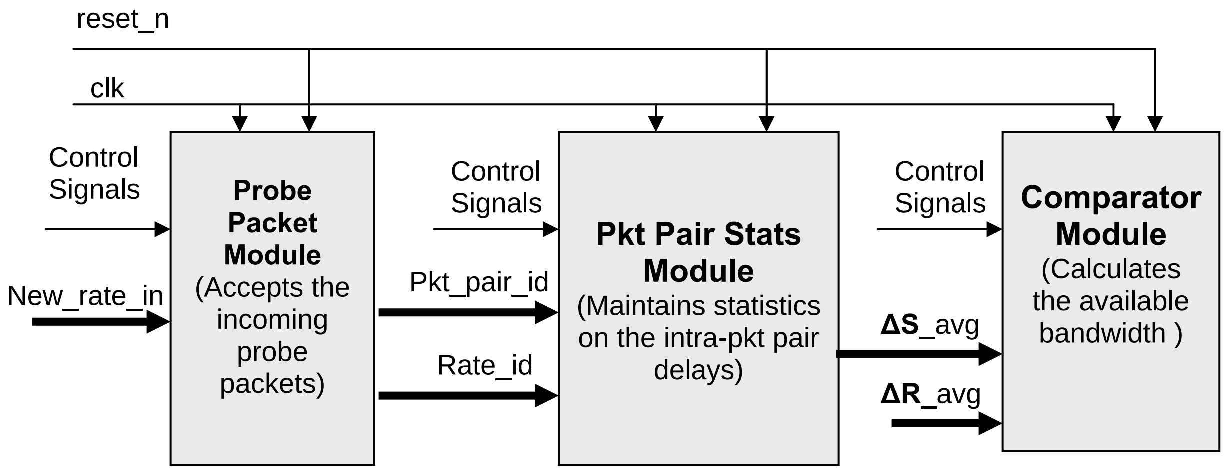


Fig. 4. Hardware implementation to measure the available bandwidth

Table 2. Resource usage report for the bandwidth measurement module:

Register elements	LUTs	Multiplexers	RAM (Dual Port)*
237 out of 67,584 (0.35%)	2896 out of 67,584 (4.29%)	2565 out of 67,584 (3.8%)	RAM 16 X 1: 32 (out of possible 33792)

*See note after Table 1 in Section 2.3 for the available FPGA resources.

4. TCP Implementation

In Section 2, a specific implementation of RTP over UDP was considered to measure the packet loss and one-way network delay. However, it is essential to take into account the other main protocol in the IP tool suite as well: TCP. The implementation discussed in this section takes into account the basic functionality of TCP, and uses the information contained in its header to maintain flow information and possibly to measure network parameters.

Significant amount of work has been done in this direction at the Applied Research Lab of Washington University in St. Louis, MO [24] [25] [26] [27]. They have designed a Field-Programmable Port Extender that controls the dynamic loading of hardware modules onto the FPGAs. Their other designs include a TCP Splitter on an FPGA that monitors all the incoming TCP packets and processes them, thereby reducing the load on the host and speeding up the TCP handling of data. Another area in which the hardware implementation of TCP has found use is that of Network Intrusion Detection Systems (NIDS). A high-performance TCP reassembly design pertaining to NIDS was proposed in [28] [29].

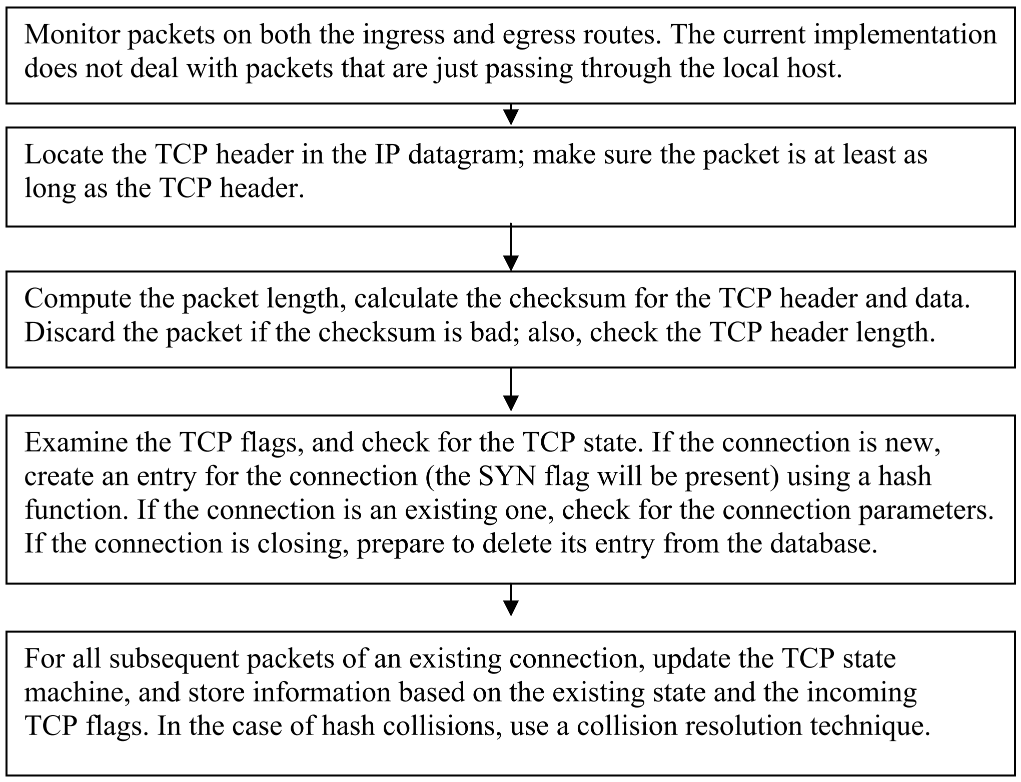
4.1 Hardware Implementation for TCP

TCP is a well-defined and documented protocol in the IP/TCP suite. Its multiple features, such as flow control, retransmission of packets, in-order delivery of packets, and so on, appeal to hardware implementation. In our paper, we have implemented the basic features of the TCP protocol, with an emphasis on hash collision management in the TCP flow memory. The idea is to have a TCP hardware implementation which can further be used for other applications related to TCP, such as the measurement of network parameters. Our design does not substitute for any of the TCP functions carried out at the host level; it only speeds up a subset of these functions.
The design flow is shown in Fig. 5. The input to the design is Ethernet frames. It will first identify whether the incoming packets belong to the TCP/IP suite, and then, depending on the octet count, the design will retrieve fields of interest such as the IP source and destination addresses, TCP port numbers, sequence and acknowledge numbers, TCP flags, the TCP checksum, the IP datagram length and the IP header length. After computing the checksum and determining the validity of the packet, the design builds up the flow information. The TCP state machine information is also updated.

There are several research papers that presented efficient hashing algorithms and hash collision resolution techniques. The design in [33] uses a set-associative hash table for fast and robust TCP session lookup. A high-speed and memory efficient TCP stream level string matching system using hash tables is proposed in [34]. An architecture that performs content scanning of TCP flows in high speed networks with the help of hash tables is developed in [35]. A collision-free hashing scheme for Longest Prefix Matching (LPM), which is a fundamental part of various networking tasks, is described in [36]. However, to the best of our knowledge, not much research has been done on comparing existing hash collision resolution techniques. The three open addressing hash collision resolution techniques implemented in this section have been well-defined and documented in various texts before[30]. Nevertheless, a comparison of the three techniques in the context of TCP traffic has not been done in either software or hardware. Our research tries to achieve this using FPGA hardware.

Our overall design is shown in Figure 6; it contains the following hardware blocks:

- *Input Mux*: This bit of hardware is not actually a part of the actual design; it illustrates the fact that the design monitors the traffic from the Ethernet layer to the application layer at the host as well as in the reverse direction.
- *Flow classifier and header module*: This module classifies the packets into different flows on the basis of their IP addresses and port numbers. It also retrieves other relevant information, such as the TCP flags, sequence number, ACK, and so on from the TCP header and forwards it to the other modules.


Monitor packets on both the ingress and egress routes. The current implementation does not deal with packets that are just passing through the local host.

Locate the TCP header in the IP datagram; make sure the packet is at least as long as the TCP header.

Compute the packet length, calculate the checksum for the TCP header and data. Discard the packet if the checksum is bad; also, check the TCP header length.

Examine the TCP flags, and check for the TCP state. If the connection is new, create an entry for the connection (the SYN flag will be present) using a hash function. If the connection is an existing one, check for the connection parameters. If the connection is closing, prepare to delete its entry from the database.

For all subsequent packets of an existing connection, update the TCP state machine, and store information based on the existing state and the incoming TCP flags. In the case of hash collisions, use a collision resolution technique.

Fig 5. Design Flow for TCP Implementation

- *Checksum calculation and verification module***:** This module verifies the header checksum and raises a flag to discard the packet in the case of any checksum error(s).
- *TCP state machine***:** This module uses the current TCP state and the incoming TCP flags to resolve the new state. It gets the current state information from the flow management module, and using the TCP flags, it calculates the new TCP state and forwards this information to the flow management module.
- *Flow management module***:** This realizes a state machine and memory for handling and storing the connection tuple, the TCP state and the valid bit information for each connection. The memory is accessed using a hash function.
- *Hash function generator*: The hash function implementation has been adapted from the one described in the Linux kernel. The pseudo code for this implementation is as follows:

```
   int h = (laddr ^ lport) ^ (faddr ^ fport);
   h ^= h >> 16;
   h ^= h >> 8;
return h & (tcp_ehash_size - 1);
```

  Here, laddr and faddr represent the local and foreign IP addresses, respectively; lport and fport represent the local and foreign TCP port numbers, respectively. The value of tcp_ehash_size would be the size of the memory. Each entry in the memory stores the TCP tuple (IP source and destination addresses, and the TCP source and destination port numbers). A valid bit is also stored along with the tuple. When a location is indexed using the calculated hash value, the valid bit is checked. If the bit is valid, then a tuple match is initiated. In the case of a successful match, it is inferred that the incoming flow is the same as the stored one, and the information is updated. However, if the valid bit is set but the tuple does not match, it means that there is a collision, and it is dealt with in the manner explained later.

- *MSL2 timer*: When a TCP connection goes into the TIME WAIT state, the MSL2 timer is activated to close the connection after a time-out. In this hardware implementation, a counter is used to keep track of the time lapsed, and, once a time-out occurs, the valid bit for that connection is invalidated, thereby making that memory available to a new connection.

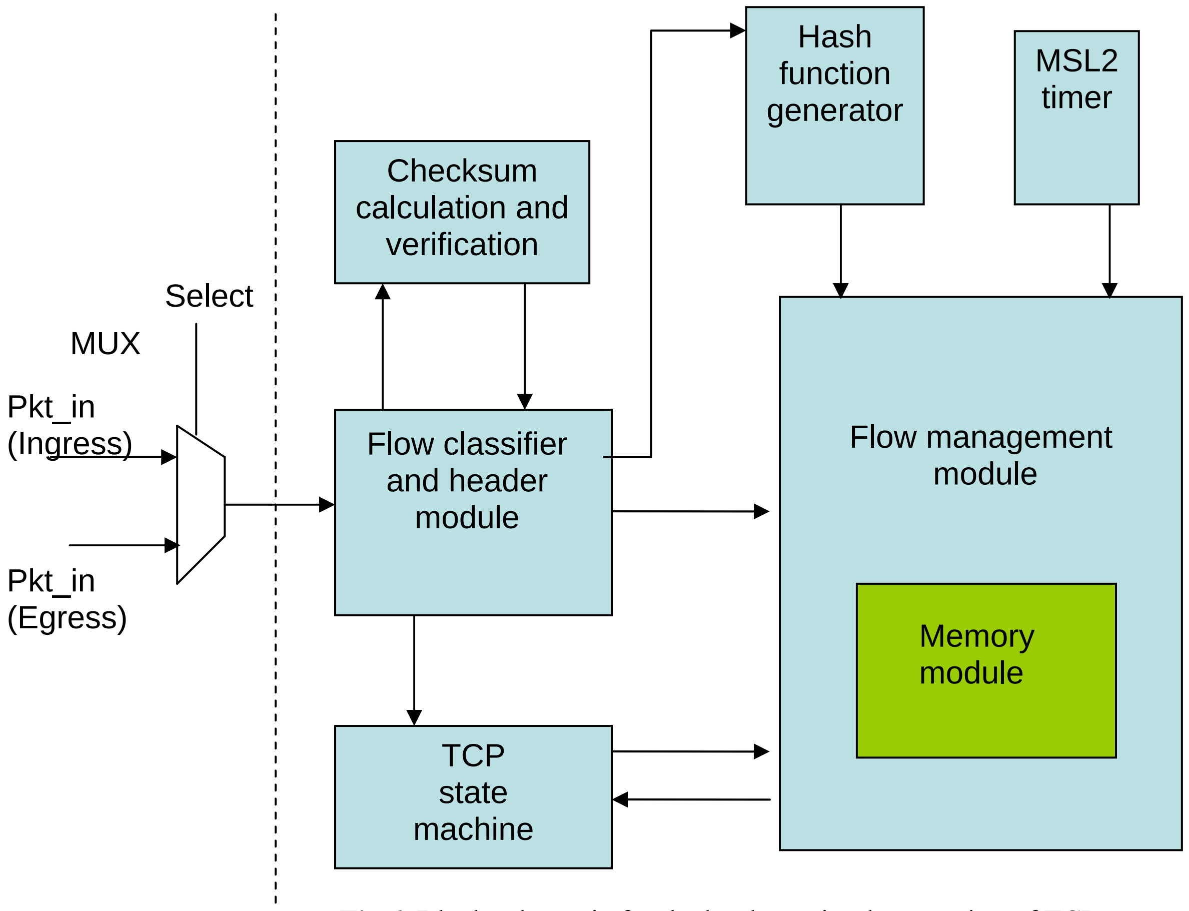


Fig 6. Block schematic for the hardware implementation of TCP

A detailed description of the design flow follows. The incoming packet can be either from the application or any higher layer from the local host, or from another host that transmitted it via the Ethernet interface. The common factor in each case is the local host IP address – appearing either as a source or a destination. Currently, there is no provision to deal with packets which are just “passing through” the network node. The way the design identifies the ingress or the egress packet is by comparing the IP addresses in the IP header with the local IP address. Once the presence of a packet is detected over the Ethernet interface, a counter is started that counts the number of bytes being received. At the proper instance, the IP addresses, TCP port numbers, TCP flags or other relevant information are recorded from the Ethernet frame. Each clock cycle records one byte of information.

The cycle count of successive operations on various blocks is given below:
1. A packet arrives and is processed by the header module. Once the IP addresses and TCP port numbers are ready, the tcp_tuple_rdy_n signal is activated; it becomes available 47 clock cycles after the beginning of packet transmission.
2. The tuple information is sent to the hash module and the hash index is calculated. This takes another six cycles.

3. Other required information is available in another seven cycles, after which the memory is accessed to see if the incoming packet belongs to an existing connection, or a new slot has to be allocated in the memory. This is accomplished in another three cycles.
4. The TCP state machine module receives information about the current state of the connection and the TCP flags; it calculates the new TCP state based on this data. This takes one clock cycle.
5. Once the memory module gets the new TCP state information, it takes two cycles to update its database.

Thus, in about 66 clock cycles the TCP information for the current connection is retrieved, processed and stored. This time span is comparable to the duration of the smallest Ethernet packet (64 bytes). Hence, it can be concluded that the hardware processing of the smallest Ethernet frame could be done transparently if the implementation was run at 1.289 GHz; this is possible with an ASIC realization of our design. For larger frame sizes, lower frequencies may suffice.

Hash Collisions:
We implemented three models to deal with hash collisions. They are all part of the open addressing scheme in which the array itself is probed or searched for the required connection (or an empty location is assigned for a new connection). Thus, these techniques do not involve pointer-based data structures. Irrespective of what technique is used, the successive hash values for probing the array are calculated using the original hash index. Thus, each hash index leads to a fixed set of hash values. These values are calculated and stored in an array while the hardware is waiting for the TCP information to come in; this saves on the clock cycles because as soon as a collision is detected, the values in the hash array can be used in a loop to find an existing connection or an empty location for a new connection. The three probing techniques that we implemented for hash collisions are as follows:

1. The first option incorporates the linear probing technique to resolve a hash collision [37]. As per this technique, a serial search is carried out through the memory to locate an existing connection, or find a vacant location to store a new connection, starting from the point where the collision occurred. Once the collision is resolved, the memory is updated using the TCP state information. Linear probing serves best the principle of locality of references but leads to primary clustering. If the initial hash function is h(k), a linear probing function would be $h(k,i) = (h(k) + i) \bmod m$ [30]. Here m is the size of the hash table, and the stepsize is i . In this design, the stepsize is one, hence the search is conducted on consecutive locations until a match is found. However, the operation of inferring, finding or deleting elements is rather complex with linear probing. Also, these operations become slower as clustering worsens.

2. The second option implements quadratic hashing. In terms of the locality of reference principle and clustering, it falls somewhere between linear hashing and double hashing, which is the third technique to be implemented. If the initial hash function is h(k) and the size of the table is m, then the i^{th} probe position for element k is given by the function:
$h(k,i) = (h(k) + c_1 i + c_2 i^2) \bmod m$, where $c_2 \neq 0$. If $c_2 = 0$, then $h(k,i)$ becomes a linear probe. For a given hash table, the values of c_1 and c_2 remain constant. In this design, both have been chosen as one. Here too, the search begins where the original hash index points in the array. This technique creates secondary clustering since the same pattern is created for each search. In fact, it may not locate an empty space if m is not a prime number; this is also true if the table is half full with m being prime.

3. The last option implements double hashing. The hash values are calculated using the original hash value h(k) and another key $h_1(k)$, such that $h(k,i) = (h(k) + i\, h_1(k)) \bmod m$. Here m and i are as defined earlier. In our design, we calculate the value of $h_1(k)$ by performing a hash operation on the IP source and destination addresses. We have chosen a specific series of operations as described in [32]. Let dhash_key

be a 64-bit value obtained by concatenating the IP source and destination addresses. The final value of dhash_key is calculated by:

```
dhash_key += ~(dhash_key << 32);
dhash_key ^= (dhash_key >> 22);
dhash_key += ~(dhash_key << 13);
dhash_key ^= (dhash_key >> 8);
dhash_key += (dhash_key << 3);
dhash_key ^= (dhash_key >> 15);
dhash_key += ~(dhash_key << 27);
dhash_key ^= (dhash_key >> 31);
```

Here, ^ represents the bitwise XOR operation, ~ represents bitwise negation, << represents left shft operation and >> denotes right shift operation.

Double hashing reduces the clustering effect and results in fewer comparisons than linear hashing, thus requiring a smaller memory. Implementation, however, is more cumbersome. Assume a load factor $\alpha < 1$, which is the ratio of the stored items to the table size and also uniform hashing (that is, all probe sequences are equally likely). Then, the expected number of probes in an unsuccessful search with an open addressing hash technique is at most $1/(1-\alpha)$. For a successful search, this number becomes

$$\frac{1}{\alpha}\ln\frac{1}{1-\alpha}+\frac{1}{\alpha}$$ [38].

Simulation and Analysis:
The TCP traffic data set for analyzing the three techniques was obtained from the Information System Technology Group at the MIT Lincoln Laboratory [39]. The data was originally recorded in 1998 and is available in the pcap format. The tcpdump software was used to read this data set. Approximately, 81000 packets were scanned, and 130 SYN packets were selected from the scanned data. Each of these 130 SYN packets is either to or from a fixed IP address (which is the local host). Further, the foreign host in each packet is unique, i.e., all 130 packets have a different foreign IP address. This is similar to the scenario where 130 new connections come in (from different sources), without any of the existing connections being closed. Such a scenario puts maximum strain on the collision management technique. Fig. 7 shows the number of hash memory probing iterations as a function of the incoming new connections when attempting to find an empty location in the memory for a new connection. The inter-packet arrival time between the packets has been divided by a factor of 100000, since the simulator was unable to simulate the real-time intervals in a reasonable amount of time. The hash table can store 128 connections. It can be observed that in the case of linear probing the number of iterations starts to increase much early because of clustering. In the case of double hashing, there is less clustering since the connection flows are more evenly distributed throughout the memory, and the number of iterations increases only towards the end. For quadratic hashing, clustering shows up somewhere in between. The linear hashing technique could accommodate all the 128 connections, whereas the quadratic hashing and double hashing could accommodate 126 and 124 connections respectively.

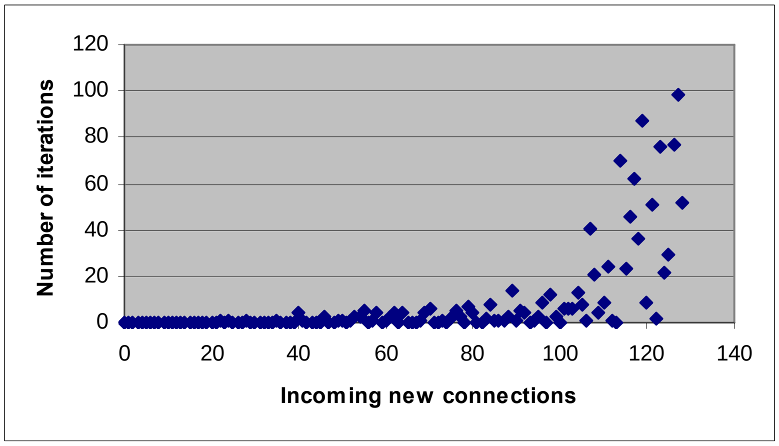


Fig 7a: Linear hashing

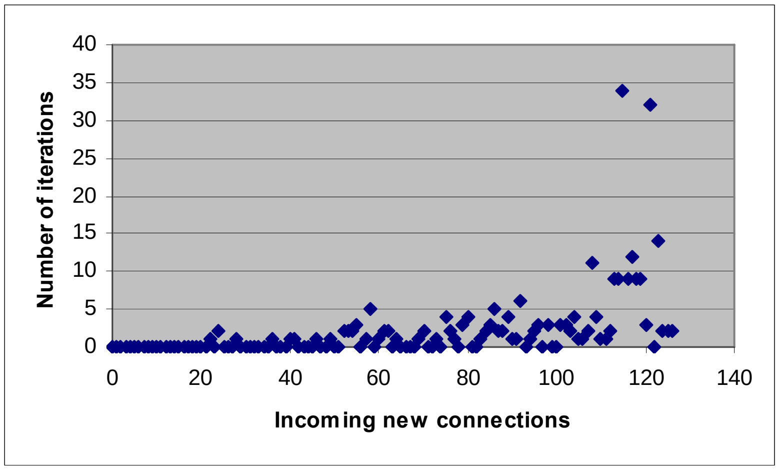


Fig 7b: Quadratic hashing

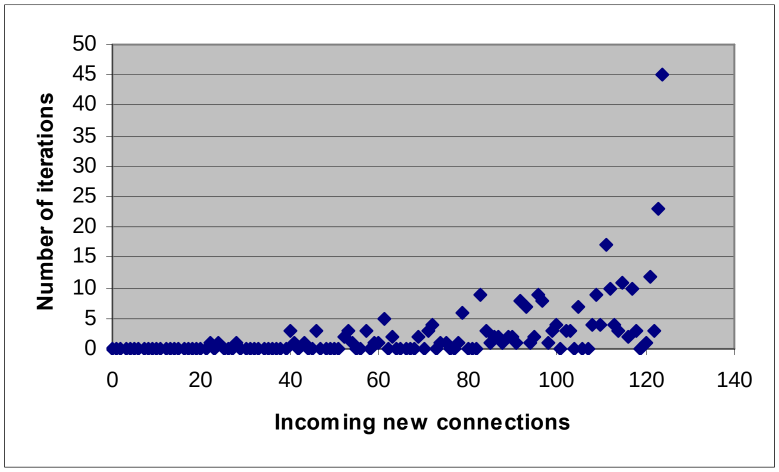


Fig 7c: Double hashing

5. Conclusion and future work

We have presented three hardware implementations of existing techniques to measure network parameters. Emphasis was made to the TCP protocol. The purpose is to speed up protocol measurements by means of hardware and bring it as close to the transmission speed as possible. FPGA-base realizations were presented. Experiments were implemented to evaluate three hashing schemes for storing TCP state tuples. The results show that clustering in the hash table is maximum in linear hashing and least in double hashing. The principle of locality of reference is maximized in linear hashing, and minimum is double hashing. Quadratic hashing lies in between the two. Therefore, the number of iterations to find an empty location increases dramatically in linear hashing as the hash table gets filled up. The iterations are most evenly distributed in case of double hashing.

Further research can be done towards using the TCP implementation for any higher layer application that uses TCP as the transport layer. Also, future work could implement pipelining to further speed up the processing of the TCP traffic. Off-chip memory would be required to support a large number of connections. A provision to monitor the traffic that is just "passing through" the host could also be incorporated in the designs.

6. References

[1] W. Jiang, H. Schulzrinne, "Modeling of Packet Loss and their effect on Real-Time Multimedia Service Quality", *NOSSDAV* 2000, Chapel Hill, NC, June 2000.

[2] M. Yajnik, S. Moon, J. Kurose and D. Towsley, "Measurement and Modeling of the Temporal Dependence in Packet Loss", *Proceedings of IEEE INFOCOM '99*, New York, vol. 1, pp345-352, March 1999.

[3] H. Sanneck, G. Carle, and R. Koodi, "A Framework model for Packet Loss Metrics Based on Loss Runlengths", *In SPIE/ACM SIGMM Multimedia Computing and Networking Conference*, January 2000.

[4] D. Falavigna, M. Matassoni, S. Turchetti; "Analysis of different acoustic front-ends for automatic Voice over IP Recognition", Automatic Speech Recognition and Understanding, 2003. ASRU '03. 2003 IEEE Workshop on 30 Nov.-3 Dec. 2003 pp:363 - 368

[5] E.J. Daniel, K.A. Teague; "Sensitivity of MIL-STD-3005 MELP to packet loss on IP networks", Circuits and Systems, 2002. MWSCAS-2002. The 2002 45th Midwest Symposium, Volume 3, 4-7 Aug. 2002 pp:III-77 - III-80 vol.3

[6] P. Benko, G. Malicsko, A. Veres; "A large-scale, passive analysis of end-to-end TCP performance over GPRS", INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies; Volume 3, 2004 pp:1882 - 1892 vol.3

[7] T. Soetens, C.S. De, O. Elloumi; "A relative bandwidth differentiated service for TCP micro-flows", Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on 15-18 May 2001 pp:602 – 609

[8] N.G. Duffield, Lo Presti, V. Paxson, D. Towsley; "Inferring link loss using striped unicast probes", INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE Volume 2, 22-26 April 2001 pp:915 - 923 vol.2

[9] R. Cascares, N.G. Duffield, J. Horowitz, D. Towsley; "Multicast-based inference of network-internal loss characteristics", Information Theory, IEEE Transactions on Volume 45, Issue 7, Nov. 1999 pp:2462 - 2480

[10] P. Skelly, M. Li; "EIPMon: an enhanced IP network monitoring tool", Internet Applications, 1999. IEEE Workshop on 26-27 July 1999 pp:20 - 27

[11] J. Feigin, K. Pahlavan; "Measurement of characteristics of voice over IP in a wireless LAN environment", Mobile Multimedia Communications, 1999. (MoMuC '99) 1999 IEEE International Workshop on 15-17 Nov. 1999 pp:236 - 240

[12] S.B. Moon, P. Skelly; D. Towsley; "Estimation and removal of clock skew from network delay measurements", INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE Volume 1, 21-25 March 1999 pp:227 - 234 vol.1

[13] L. Zheng; L. Zhang; D. Xu; "Characteristics of network delay and delay jitter and its effect on voice over IP (VoIP)", Communications, 2001. ICC 2001. IEEE International Conference on Volume 1, 11-14 June 2001 pp:122 - 126 vol.1

[14] B.K. Choi, S. Moon, Z. Zhang; K. Papagiannaki, C. Diot; "Analysis of point-to-point packet delay in an operational network", INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies Volume 3, 2004 pp:1797 - 1807 vol.3

[15] P. DeLeon, C.J. Sreenan; "An adaptive predictor for media playout buffering", Acoustics, Speech, and Signal Processing, 1999. ICASSP '99. Proceedings., 1999 IEEE International Conference on Volume 6, 15-19 March 1999 pp:3097 - 3100 vol.6

[16] Y. Shu; F. Xue; Z. Jin; O. Yan; "The impact of self-similar traffic on network delay", Electrical and Computer Engineering, 1998. IEEE Canadian Conference on Volume 1, 24-28 May 1998 pp:349 - 352 vol.1
[17] R. Prasad, C. Dovrolis, M. Murray, K. Claffy; "Bandwidth estimation: metrics, measurement techniques, and tools", Network, IEEE, Volume 17, Issue 6, Nov.-Dec. 2003 pp:27 - 35
[18] V. Jacobson, "Pathchar: A Tool to Infer Characteristics of Internet Paths," ftp://ftp.ee.lbl.gov/pathchar/, Apr. 1997.
[19] A.B. Downey, "Using Pathchar to Estimate Internet Link Characteristics," *Proc. ACM SIGCOMM*, Sept. 1999, pp. 222–23.
[20] B. Melander, M. Bjorkman, P. Gunningberg; "A new end-to-end probing and analysis method for estimating bandwidth bottlenecks", Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE Volume 1, 27 Nov.-1 Dec. 2000 pp:415 - 420 vol.1
[21] "Iperf: Testing the limits of your network", http://dast.nlanr.net/Projects/Iperf/
[22] V. Ribeiro *et al.*, "pathChirp: Efficient Available Bandwidth Estimation for Network Paths," *Proc. Passive and Active Measurements Wksp.*, Apr. 2003.
[23] N. Hu and P. Steenkiste, "Evaluation and Characterization of Available Bandwidth Probing Techniques," *IEEE JSAC*, 2003.
[24] J.W. Lockwood; "Evolvable Internet hardware platforms", Evolvable Hardware, 2001. Proceedings. The Third NASA/DoD Workshop on 12-14 July 2001 pp:271 - 279
[25] D.V. Schuehler, J. Lockwood; "TCP-Splitter: A TCP/IP flow monitor in reconfigurable hardware", High Performance Interconnects, 2002. Proceedings. 10th Symposium on 21-23 Aug. 2002 pp:127 - 131
[26] D.V. Schuehler and J.W. Lockwood ,"A Modular System for FPGA-Based TCP Flow Processing in High-Speed Networks", http://www.arl.wustl.edu/~lockwood/publications/TCP_Processor_04.pdf
[27] T.S. Sproull, J.W. Lockwood, D.E. Taylor; "Control and configuration software for a reconfigurable networking hardware platform", Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on 22-24 April 2002 pp:45 – 54
[28] M. Necker, D. Contis, D. Schimmel; "TCP-Stream reassembly and state tracking in hardware", Field-Programmable Custom Computing Machines, 2002. Proceedings. 10th Annual IEEE Symposium on 22-24 April 2002 pp:286 - 287
[29] S. Li, J. Torresen, O. Sorasen; "Improving a network security system by recongurable hardware", Norchip Conference, 2004. Proceedings, 8-9 Nov. 2004 pp:135 – 138
[30] http://en.wikipedia.org/wiki/Hash_table
[31] S. Floyd, V. Jacobson; Random Early Detection gateways for Congestion Avoidance V.1 N.4, August 1993
[32] http://www.concentric.net/~Ttwang/tech/inthash.htm
[33] F. Pong ; "Fast and Robust TCP Session Lookup by Digest Hash", Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on; 12-15 July 2006
[34] Y. Sugawara, M. Inaba, K. Hiraki; "High-speed and memory efficient TCP stream scanning using FPGA", Field Programmable Logic and Applications, 2005. International Conference on 24-26 Aug. 2005
[35] D.V. Schuehler, J.W. Lockwood, J. Moscola; "Architecture for a hardware-based, TCP/IP content-processing system", Micro, IEEE Volume 24, Issue 1, Jan.-Feb. 2004 pp:62 – 69
[36] J. Hasan, S. Cadambi, V. Jakkula, S. Chakradhar; "Chisel: A Storage-efficient, Collision-free Hash-based Network Processing Architecture", Computer Architecture, 2006. 33rd International Symposium on 17-21 June 2006 pp.203 – 215
[37] W. Litwin; "Linear Hashing: A New Tool for File and Table Addressing", 6th Conference on Very Large Databases, 1980, pp.212-223.
[38] Introduction to Algorithms, by T.H. Cormen, C.E. Leiserson and R.L. Rivest, MIT Press, 1990
[39]http://www.ll.mit.edu/IST/ideval/