Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet-Pairs – Extended Report*

Sebastian Zander, Grenville Armitage
Centre for Advanced Internet Architectures, Technical Report 130730A
Swinburne University of Technology
Melbourne, Australia
szander@swin.edu.au, garmitage@swin.edu.au

***Abstract*—Accurate and frequent round trip time (RTT) measurements are important in both testbeds and operational networks. Active measurement techniques, such as "ping", inject probe packets that can modify the behaviour of the observed network and may produce misleading RTT estimates if the network handles probe packets differently to regular packets. Previous passive measurement techniques address these issues, but require precise time synchronisation or are limited to certain traffic types, such as TCP flows. We introduce a passive technique for RTT measurement called Synthetic Packet-Pairs (SPP). SPP provides frequently updated RTT measurements using any network traffic *already present* in the network *without* the need for time synchronisation. SPP accurately measures the RTT experienced by any application's traffic, even applications that do not exhibit symmetric client-server packet exchanges. We experimentally demonstrate the advantages of SPP in different scenarios.**

***Index Terms*—Passive measurement, Round Trip Time, RTT.**

I. Introduction

The measurement of a path's round trip time (RTT) is often crucial when evaluating and improving the performance of application protocols, transport protocols or network equipment. Also, for service providers it is becoming increasingly important to monitor and manage the RTT experienced by highly interactive applications, such as voice over IP, latency-sensitive first person shooter (FPS) multiplayer online games [1] or mission-critical business applications. Existing RTT measurement techniques differ in whether they are active (injecting additional traffic into the network) or passive (using traffic already in the network), their achievable sample rate (how many RTT measurements per time unit), the degree to which they rely on synchronised clocks at different measurement points and their ability to determine the RTT experienced by flows from different applications.

High rate sampling of a path's RTT is important for detailed observation of RTT versus time (such as watching bottleneck queues filling and draining, or the rise and fall of contention-induced transmission delays). Active measurements find this problematic, as the additional probe traffic (proportional to the desired sample rate) can interfere with the characteristics of the path being probed and skew the measured RTTs. Passive techniques can potentially sample at close to the existing traffic rate with no new load on the path.

Active measurements may also experience misleading RTTs across network paths that differently prioritise the forwarding of IP packets according to their higher-layer protocol and/or port numbers. It is hard to measure the RTT an application flow sees when the active probe packets experience different forwarding delays [2]. These problems do not affect passive measurements based on an application flow's own traffic.

Passive measurement techniques have their own challenges. One way delay (OWD) measurements require observation points with precisely synchronised clocks. Other previously proposed passive techniques require the enhancement of routers along a path or manipulation of packet contents in flight. Some RTT measurements rely on symmetric request-response patterns from the traffic under observation.

We introduce a novel passive RTT measurement technique that:

- Calculates RTT of a network path between two measurement points based on *existing* traffic.

*This report is an extended version of "Minimally-Intrusive Frequent Round Trip Time Measurements Using Synthetic Packet-Pairs" accepted as a short paper in IEEE LCN 2013, October 2013.

- Does *not* require clock synchronisation of measurement points.
- Is protocol-agnostic and works with asymmetric traffic flows (flows without request-response behaviour).
- Provides accurate RTT samples at a rate proportional to the slowest traffic rate in either direction.
- Does not require changes on network routers or modifications of packet contents.

We call our technique "synthetic packet-pairs" (SPP)[1], and have publicly released an open-source prototype implementation [4].

SPP is beneficial in scenarios where the path under test is sensitive to the additional load of active probing, frequent sampling of the path is desired (at sample rates approaching the packet rates of existing traffic), tightly synchronised measurement point clocks are unavailable (for cost or deployment reasons), or probe traffic is treated differently from the application traffic under study. SPP is also advantageous where detailed RTT estimates are desired for delay or jitter estimation, particularly for interactive applications that do not have symmetric request-response packet pairs.

We consider SPP to be non-intrusive in scenarios where RTT is measured in real-time and a (logically) separate communication path exists between the measurement points, or where RTTs can be computed offline (e.g. testbed measurements). Otherwise, it is minimally intrusive, since the packet timestamp data that needs to be transmitted uses relatively little bandwidth.

The report is organised as follows. Section II outlines the challenges and requirements for RTT measurement schemes. Section III describes our proposed SPP algorithm. Section IV describes our prototype implementation. Section V experimentally demonstrates the advantages of SPP over load-sensitive paths, where paths differentially forward different packet types, or where traffic flow is asymmetric. Section VI concludes and outlines future work.

II. Background and motivation

Here we briefly review some of the key the issues around active and passive measurement of RTT.

[1]SPP is *unrelated* to packet-pair available bandwidth probing techniques like [3]. For SPP a packet pair consists of one packet going from point A to point B followed by a 'reverse' packet going from point B to point A. For bandwidth probing techniques a packet pair is two subsequent closely-spaced packets going in the same direction.

A. Sample rate

Building up a broad sense of RTT variations over long periods of time may require sampling a path's RTT only a few times a minute. However, capturing detailed insights into rapid (yet infrequent) RTT transitions may need a sample rate approaching the packet rate of the traffic flowing across the path. Examples of this latter case include capturing the impact of one or more TCP flows congesting a bottleneck queue, observing the impact of TCP cross-traffic on time-sensitive flows through shared bottlenecks, identifying the short and long term impacts of route changes on path length, and capturing the packet-by-packet impact of contention-induced transmission delays over wireless links.

B. Active RTT measurement

Active measurement techniques inject extra packets (probes) across a network path and use their transit times to measure the path's delay at the instant each probe was sent. This approach is useful for infrequent delay measurements, or measurements across otherwise idle paths. A common example is `ping`, which transmits a probe (ICMP Echo-request) packet towards a target host and elicits an (almost) immediate reply (ICMP Echo-reply) packet. The time between query and response packets is taken to be the path's RTT.

1) Differentiated forwarding of probe packets: Unfortunately, forwarding elements (such as routers or switches) along the path may handle active probe packets differently to regular traffic, resulting in unrepresentative RTT measurements. For example, when routers handle ICMP packets in their slow path `ping` can overestimate a path's RTT [2]. Bottleneck routers may also have rules to prioritise specific application flows during times of congestion. Active probe packets that do not match the rules (such as ICMP `ping`, or UDP or TCP probes from `echoping` [5]) will generate RTTs that do not reflect the RTTs experienced by prioritised traffic.

2) Sample rate and network load: Active measurement schemes add network load proportional to the desired RTT sample rate, which can noticeably alter the network's behaviour and performance. This is particularly problematic over link technologies, such as 802.11 Wireless LANs, where modest loads in packets per second are known to cause noticeable degradation of service even when the additional active probing load is low when measured in bits/second [6].

C. Passive RTT measurement

By measuring the delays experienced by existing traffic, passive measurement techniques avoid adding load to a network path and can identify the RTTs experienced by application flows subject to different forwarding priorities.

1) Clock synchronisation: Some techniques measure one-way delay (OWD) by noting the time it takes an IP packet to transit between two measurement points having precisely synchronised clocks [7]–[9] and RTT can be calculated by adding OWD in each direction.

Synchronisation using the network time protocol (NTP) is often not adequate due to varying network conditions and NTP server outages [10]. Global positioning system (GPS) timing signals can be used instead, but associated infrastructure (such as roof-mounted GPS antennas) is often complex and costly. RADclock [11] is a low-cost alternative to GPS-based synchronisation, but it is not widely deployed yet. Also, while RADclock is more robust than NTP, its accuracy can still be inadequate in adverse conditions, such as when there are time server outages.

Other OWD techniques require modifications to existing routers [12], [13] or manipulate the content of existing packets [14]. A few proposed OWD techniques do not require clock synchronisation, but their accuracy remains unclear [15]. RTT can be calculated by adding OWD in each direction.

2) Symmetric traffic flows: Some techniques directly estimate RTT at a single measurement point (usually close to the traffic's origin) from the time it takes for an application request packet in one direction to be answered by a matching response packet in the return direction [16]–[20].[2] They require symmetric client-server traffic with enough information in each packet header or payload to match request-response pairs. However, the resulting RTT can be misleadingly inflated if the 'server' does not send response packets immediately and the estimation technique does not correct for this.

Such techniques fail when traffic in each direction has no predictable timing or semantic relationship. For example, many client-server FPS online games emit packets in each direction asynchronously and at different rates [1]. Presuming that packets in each direction are request-response "pairs" would result in unpredictable intervals (of up to tens of milliseconds) added to the path's actual RTT.

[2]The TCP protocol's timestamp option also allows to measure RTT.

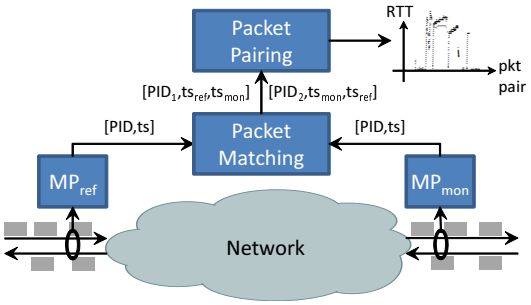


Figure 1: SPP overview: packet capturing, packet matching and packet-pair identification

3) Sample rates: Passive schemes cannot sample idle paths, as the sample rate depends on the rate at which traffic traverses the path. However, test traffic can be *generated* to achieve continuous monitoring. Furthermore, well trafficked paths can be observed in detail without adverse impact on the path's characteristics.

III. SPP Measurement Technique

SPP fills a gap in the existing set of RTT measurement techniques, and was initially documented in a technical report [21]. No new traffic is injected, no modifications are required to existing infrastructure, and traffic in each direction may be asymmetric and from unrelated application flows.

Figure 1 illustrates the three stages of SPP – capturing packets at two measurement points (MPs)[3], matching packets seen at both measurement points, and identifying packet-pairs from which to calculate RTTs.

A. Measurement points and Packet IDs

We will describe SPP in terms of two MPs, $\mathrm{MP}_{\mathrm{ref}}$ (reference) and $\mathrm{MP}_{\mathrm{mon}}$ (monitor), located so that the network traffic of interest traverses both points. $\mathrm{MP}_{\mathrm{ref}}$ passively records the passing of packets heading towards $\mathrm{MP}_{\mathrm{mon}}$ (for example, by monitoring traffic using in-line network taps or mirrored ports on a switch). $\mathrm{MP}_{\mathrm{mon}}$ performs the same action for packets heading towards $\mathrm{MP}_{\mathrm{ref}}$.

For every recorded packet both $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ log a timestamp (ts) representing when the packet was seen, and a short 'packet ID' (PID) calculated from a suitable hash function (e.g. CRC32 or other suitable hash functions in [8], [22], [23]) over key bytes within the

[3]In principle SPP can be used with multiple MPs. Consider measuring RTTs of packets traversing multiple ingress points and/or multiple egress points. Packet matching can be performed between each pair of MPs.

packet. To uniquely identify the same packet passing $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ the PID is based on portions of a packet that are invariant during transit between $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ but vary between different packets (cf. [7]–[9]). We describe which portions of packets are used in Section IV-B. Each MP accumulates a list of [PID,ts] pairs based on the captured packets. These two lists are then combined to create packet-matched lists used for packet-pair identification and RTT calculation.

B. Packet Matching

Figure 1 shows the [PID,ts] lists being combined at a third location, but they can also be combined at $\mathrm{MP}_{\mathrm{ref}}$ or $\mathrm{MP}_{\mathrm{mon}}$. If the link used to transmit [PID,ts] pairs is a physical out-of-band link or a logically isolated channel sharing the same underlying infrastructure as the monitored path, the [PID,ts] pairs may be combined in real-time without affecting the observed network. If no separate channel is available, [PID,ts] pairs may be stored at each MP for later transfer across the measured network (e.g. during non-measurement or off-peak periods). In these cases SPP is non-intrusive. If a channel along the same infrastructure being measured is used during the measurement, SPP is minimally intrusive in the sense that transmitting the [PID,ts] list uses relatively little bandwidth (see Section IV-A2).

Two input streams I_{mon} and I_{ref} represent [PID,ts] pairs from packets captured at $\mathrm{MP}_{\mathrm{mon}}$ and $\mathrm{MP}_{\mathrm{ref}}$ respectively. A queue Q_{ref} is used to buffer [PID,ts] pairs from I_{ref} not yet detected in I_{mon}. The algorithm processes packets from I_{mon} in the order of their arrival at $\mathrm{MP}_{\mathrm{mon}}$.

For each packet P_{cur} captured at $\mathrm{MP}_{\mathrm{mon}}$ we search for a packet with the same PID captured at $\mathrm{MP}_{\mathrm{ref}}$. The algorithm first checks if the packet is found in Q_{ref}. If not, new packets are read from I_{ref} into Q_{ref} until a packet matches, the maximum queue length is reached or a packet's timestamp differs by more than a pre-defined T_{delta} from the timestamp of P_{cur} (the use of T_{delta} is discussed further in Section IV-C). Before the next packet of I_{mon} is processed, all packets in Q_{ref} whose timestamps differ by more than T_{delta} from the timestamp of P_{cur} are considered lost packets and removed from Q_{ref}.[4]

The result of the packet matching algorithm is a list of $[\mathrm{PID}_1,\mathrm{ts}_{\mathrm{ref}},\mathrm{ts}_{\mathrm{mon}}]$ tuples, representing the time potential first (1) packets of pairs were seen at $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ respectively. The same algorithm is also run with the directions reversed to construct a list of $[\mathrm{PID}_2,\mathrm{ts}_{\mathrm{mon}},\mathrm{ts}_{\mathrm{ref}}]$ tuples, representing potential second (2) packets of pairs that were seen flowing from $\mathrm{MP}_{\mathrm{mon}}$ to $\mathrm{MP}_{\mathrm{ref}}$.

[4]The loss calculation does not start before the first packet matches.

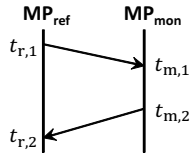
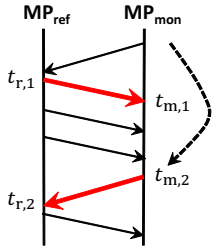
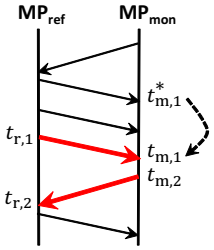


Figure 2: Packet pair and corresponding timestamps



(a) Finding a packet pair



(b) Finding the closest pair

Figure 3: SPP packet pairing approach

C. Packet Pair Identification and RTT Computation

To explain our packet-pairing algorithm we first define a short-hand notation for the timestamp $t_{i,j}$, where i indicates $\mathrm{MP}_{\mathrm{ref}}$ (r) or $\mathrm{MP}_{\mathrm{mon}}$ (m) and j indicates whether it is the first (1) or second (2) packet of a packet pair. For example, in Figure 2 $t_{\mathrm{r},1}$ is the timestamp of the first packet at $\mathrm{MP}_{\mathrm{ref}}$ and $t_{m,1}$ is the timestamp of the first packet at $\mathrm{MP}_{\mathrm{mon}}$.

1) Packet pairs: SPP makes the key assumption that each packet is used in at most one pair. Otherwise, different RTT estimates would be dependent. Furthermore, SPP ensures that the two packets of a pair are as close together as possible. Packet pairs can overlap in time, since otherwise the measurement frequency would be limited by the RTT of the path.

SPP starts with the first packet from the list of packets going from $\mathrm{MP}_{\mathrm{ref}}$ to $\mathrm{MP}_{\mathrm{mon}}$. It then searches in the second packet list (packets going from $\mathrm{MP}_{\mathrm{mon}}$ to $\mathrm{MP}_{\mathrm{ref}}$) for the first packet where the condition $t_{\mathrm{m},2} > t_{\mathrm{m},1}$ is true. A packet pair has now been identified (Figure 3a) but this pair is not necessarily the closest pair.

To find the closest pair, $t^*_{\mathrm{m},1}$ is set to $t_{\mathrm{m},1}$ and the algorithm traverses further through the first list in search for any packets where $t_{\mathrm{m},1} > t^*_{\mathrm{m},1}$ and $t_{\mathrm{m},1} < t_{\mathrm{m},2}$. As long as such packets are found the algorithm advances

in the first list. This ensures there are no other packets between the two packets of a pair (Figure 3b).

2) Computing RTT: Once a packet-pair has been identified the RTT computation is straightforward:

$$\mathrm{RTT} = (t_{\mathrm{r},2} - t_{\mathrm{r},1}) - (t_{\mathrm{m},2} - t_{\mathrm{m},1}) \ . \qquad (1)$$

The packet pairing algorithm then continues with the next packet in the first list (packets from $\mathrm{MP}_{\mathrm{ref}}$ to $\mathrm{MP}_{\mathrm{mon}}$). Note that the two directions could be reversed and the RTTs could be computed in the other direction if desired.

3) Clock synchronisation and skew: Equation 1 is based on time differences of the same clocks, so there is no need for the clocks at $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ to be synchronised over long time frames. However, error may be introduced by short-term clock *skew* during the intervals $(t_{\mathrm{r},2} - t_{\mathrm{r},1})$ at $\mathrm{MP}_{\mathrm{ref}}$ and $(t_{\mathrm{m},2} - t_{\mathrm{m},1})$ at $\mathrm{MP}_{\mathrm{mon}}$. Regular PC clocks typically skew[5] no more than a few 100 ppm [24]. Even if both skews are highly negatively correlated, we expect the error is well under 1000 ppm (an error below 0.1 ms for an RTT of 100 ms).

4) Independent pairing and achievable sample rate: A key benefit of SPP is that the packets making up each pair in Equation 1 need not be generated by the same application or hosts. That allows SPP to estimate RTT from any packet flows regardless of whether or not there is symmetric request-response behaviour. SPP can even create RTT measurements from two unrelated unidirectional flows in different directions. RTT samples are generated at the rate SPP finds packet pairs, determined by the slower of the packet rates in each direction.

IV. Practical Implementation

In this section we briefly[6] describe our open-source implementation of SPP that runs under FreeBSD[7] and Linux [4], including its two RTT calculation modes (offline and real-time), PID generation, and an optimisation of the packet matching algorithm. The acronym 'SPP' may now refer to our implementation or the algorithm depending on context.

A. Offline and real-time modes

SPP can calculate RTTs from previously captured tcpdump files (offline mode) or live capture (real-time mode).

[5]Due to ambient temperature/humidity affecting the clock crystal.

[6]Appendix A describes the overall software design in more detail.

[7]Also in the Ports collection, http://www.freshports.org/benchmarks/spp/

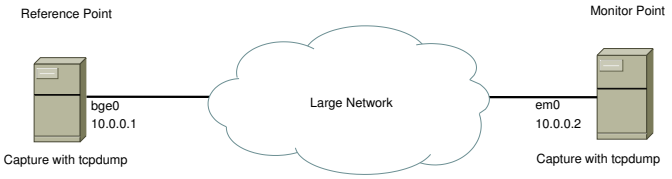


Figure 4: Using SPP with tcpdump files assuming the reference point and monitor point are located on the end hosts of a path to be measured

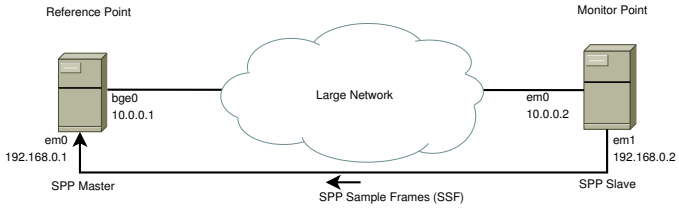


Figure 5: Using SPP with live interface monitoring assuming the reference point and monitor point are located on the end hosts of a path to be measured

1) Offline mode: In this mode we capture packets in tcpdump format at $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$, then run SPP on these two tcpdump files (potentially at an entirely separate location) to extract RTT samples. Figure 4 shows a simple example where $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ are located on the end hosts of a path to be measured and tcpdump files `ref.cap` and `mon.cap` are captured at $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ respectively.[8] Traffic of interest is presumed to run between IP addresses 10.0.0.1 and 10.0.0.2. RTTs are computed with this command:

```
spp -f ref.cap -a 10.0.0.1 -F mon.cap -A
10.0.0.2
```

Offline mode is non-intrusive in that no data needs to be transferred while the network path is being measured. Furthermore, tcpdump's powerful filter language can be used to select only certain application-specific traffic flows from which to calculate RTTs.

2) Real-time mode: SPP can also monitor live network interfaces and calculate RTT samples 'on the fly'. Figure 5 illustrates a simple real-time mode configuration, with two instances of SPP configured as 'SPP Master' and 'SPP Slave'. Again, for simplicity we assume $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ are located on the end hosts of a path to be measured.

[8]In practice, $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ could be dedicated measurement hosts connected to selected points of a network path via in-line network taps, such as optical or electrical splitters, or mirroring ports on switches. An example of such a setup is shown in [4].

As $\mathrm{MP}_{\mathrm{mon}}$ the SPP Slave sends to the SPP Master UDP-based SPP Sample Frames (SSF) carrying [PID,ts] information about traffic it sees. The SPP Master (as $\mathrm{MP}_{\mathrm{ref}}$) also observes traffic and combines this knowledge with SSFs from the SPP Slave to calculate RTTs in (near) real-time.[9]

We use delta encoding of ts values, so each SSF/UDP packet can carry 248 [PID,ts] pairs (assuming an MTU of 1500 bytes) [4]. For example, an SPP Slave observing two hundred packets per second would send less than one SSF/UDP packet per second. SSFs over UDP provide timely delivery with minimal overhead, but packets can be lost. We plan to add sequence numbers to SSFs to detect loss and to support SCTP or TCP as alternative reliable SSF transport in the future.

Figure 5 actually shows the SPP Master and Slave having two network interfaces each. The network under observation is between the Master's bge0 interface (10.0.0.1) and Slave's em0 interface (10.0.0.2). A separate network path from 192.168.0.2 to 192.168.0.1 is available to carry the SSFs. We use the following command to configure the SPP Master to calculate RTTs using traffic seen on bge0 between 10.0.0.1 and 10.0.0.2 and SSFs received from the Slave (192.168.0.2):

```
spp -i bge0 -a 10.0.0.1 -A 10.0.0.2 -R
192.168.0.2
```

We configure the slave to observe traffic on em0 between 10.0.0.1 and 10.0.0.2 and send SSFs to the Master (192.168.0.1) with:

```
spp -I em0 -a 10.0.0.1 -A 10.0.0.2 -s
192.168.0.1
```

SPP can also be used for real-time processing on a single PC with two (or more) network interfaces, assuming two interfaces can be connected to the points of interest via electrical/optical splitters or mirroring ports on switches [4].

B. Packet ID generation

By default SPP calculates a CRC32 hash across unvarying fields in the IP packet header (source and destination address, protocol and IP identification)[10], the TCP header (sequence number, ACK number) or UDP header (length, checksum), as well as across the first 12 bytes of UDP payload [4]. We chose CRC32 because it is widely implemented, has low collision probability (less than 1^{-9} for more than 20 bytes input [22]), and is efficiently computed in hardware and software (300 ns on a 1.7 GHz Pentium 4m [22]). CRC32 is also one of the functions currently recommended by the IETF packet sampling work group for packet digests [25].

SPP allows the user to alter which IP and TCP/UDP header fields are used as hash input, and to tell SPP about any network address translation (NAT) along the path. This allows handling situations where $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ are placed at either side of a middlebox performing NAT, NAT with port translation or acting as a firewall (which may manipulate TCP header fields, such as the TCP sequence number).

C. Packet matching optimisation

Although SPP does not require time synchronisation, we use T_{delta} (see Section III-B) to drastically improve the performance of the packet-matching process. For each packet observed at $\mathrm{MP}_{\mathrm{mon}}$ only a T_{delta} time window of packets from $\mathrm{MP}_{\mathrm{ref}}$ needs to be searched.

T_{delta} is configurable on the command line, and must be larger than the expected network delay plus time synchronisation error. It defaults to 60 seconds – sufficient for situations where $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ are built on common PC hardware with clocks synchronised once a day (as this would normally keep the clocks within a few seconds of each other). Using NTP continuously is not required. SPP can also be told to apply a correction to the timestamps of $\mathrm{MP}_{\mathrm{ref}}$ with respect to $\mathrm{MP}_{\mathrm{mon}}$ if the clock offset is known. Then SPP can be used with a small T_{delta} value even if the actual clock offset was very large.

V. Experimental Evaluation

This section describes a number of scenarios where SPP proves to be valuable. We cover the use of unsynchronised clocks, measuring RTT using asymmetric or unrelated traffic in each direction, the benefits of high passive sample rates for observing rapid RTT transitions and the use of SPP in networks where traffic prioritisation or load-balancing prevents active probing from measuring application-specific RTTs.

A. No time synchronisation needed

To verify that SPP works without clock synchronisation and is unaffected by a significant clock offset, we generated UDP and TCP test traffic and captured it with

[9] SPP's real-time mode currently only filters on source and destination IP addresses; the support of tcpdump filter strings is planned.

[10] We do not use the IP length field, since it changes if IP packets are fragmented between $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$. If fragmentation occurs, SPP will always match on a packet's first fragment.

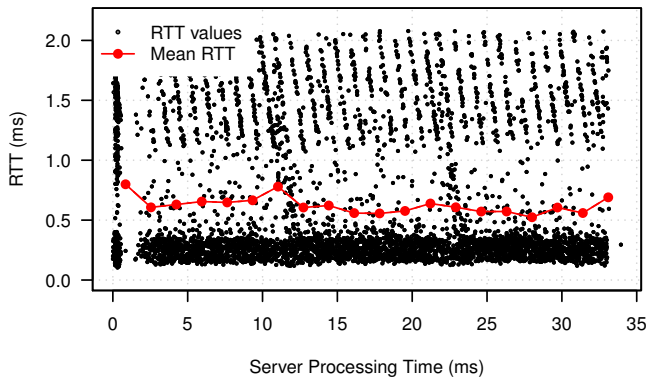


Figure 6: Path RTT versus hypothetical 'server processing time' (FPS game traffic on a LAN)

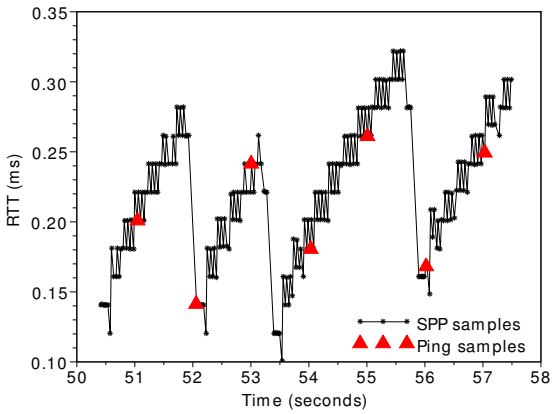


Figure 7: High sampling frequency provides detailed insights into rapid RTT variations over a consumer ADSL link

tcpdump at both MPs. We then changed the timestamps in one of the traces to simulate various clock offsets between both MPs ranging from 5 seconds to 1000 seconds. SPP computed exactly the same RTT values regardless of the actual clock offset.

B. RTT measurements using asymmetric traffic

With symmetric, request-response traffic Equation 1's $(t_{\mathrm{m},2} - t_{\mathrm{m},1})$ term represents a notional 'server processing time' (SPT) – the time between a request packet eliciting a response packet from the 'server' near $\mathrm{MP}_{\mathrm{mon}}$. Asymmetric traffic provides no such relationship, and the SPT can vary widely from one sample pair to the next. Here we show that SPP provides useful RTT samples even as SPT varies widely.

We chose to use an FPS game, as they tend to use a client-server communication model but do not generate strict request-response traffic patterns [1]. Traffic was captured from a five minute game of 'Return to Castle Wolfenstein: Enemy Territory' (ET) over an Ethernet LAN; $\mathrm{MP}_{\mathrm{ref}}$ was located at one client and $\mathrm{MP}_{\mathrm{mon}}$ was located at the server.

Server-to-client game state update packets (snapshots) are sent at fixed 50 ms intervals. Client-to-server command packets are sent at uncorrelated and variable intervals from 10 to 100 ms depending on the client and the game settings. During our measurement the client-to-server packets were spaced roughly 33 ms apart. Here SPT reflects the time between the game server receiving a command packet and sending the next snapshot.

Figure 6 is a scatter-plot of RTT versus SPT for each packet-pair found by SPP. The line indicates the mean RTT (0.6–0.7 ms). The RTTs are small, and with modest jitter (consistent with traffic on a LAN). It is clear the RTT distribution is consistent across the much larger range of SPTs from 0 ms to 33 ms (SPTs were bounded by the client-to-server packet interval).

C. High sample rates (frequent measurements)

SPP's ability to provide frequent RTT measurements is especially useful if one is interested in time series measurements (such as investigating TCP congestion control behaviour). Low rate probing (such as ping) undersamples a path and misses sharp transitions in RTT (cf. Nyquist theorem).

1) Bulk TCP transfers over ADSL broadband links: Our first example is from a TCP connection taking roughly 200 seconds to push 13 MB of data out a 653 kbps home broadband Internet connection (the bottleneck in this case). SPP extracted approximately 6300 RTT samples from tcpdump files captured at both ends of the connection.

The value of SPP's high sample rate can be seen in Figure 7. A mere eight seconds of captured traffic reveals detailed RTT fluctuations driven by TCP's congestion control behaviour, which is not visible to the one-per-second ping samples.

SPP's sampling of every suitable packet pair also allows visibility into queuing behaviour along a path. Figure 8 shows a histogram of all the RTT samples taken during the transfer, revealing distinct bands roughly 20 ms apart. This reflects detailed evidence of quantisation caused by queuing and de-queuing of full-sized TCP/IP frames in the ADSL modem's upstream buffer.[11]

[11] Common ADSL-based Internet service overheads turn each 1416 byte IP frame into a 1456 byte ATM Adaptation Layer 5 (AAL5) frame. This requires 31 x 53-byte ATM cells to transmit, or very close to 20 ms at 653 kbps.

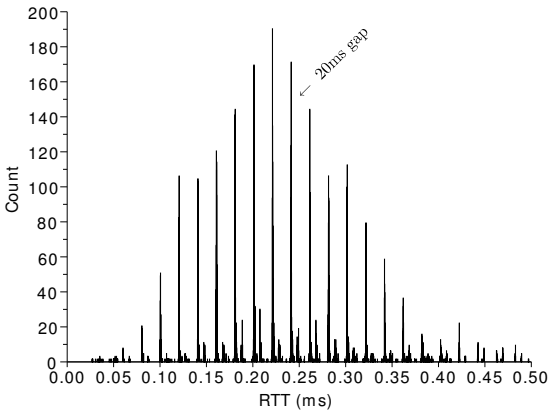


Figure 8: SPP reveals RTT bands due to upstream queuing over a consumer ADSL link

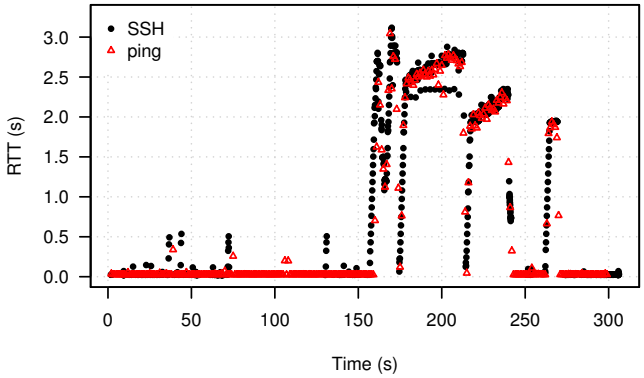


Figure 9: RTTs of interactive SSH session measured by SPP and path RTT measured by ping

2) SSH terminal sessions over ADSL broadband links: Figure 9 shows an example, where we ran an SSH interactive session and concurrent once-per-second ping over an ADSL link. We can see that SPP provides a much more detailed picture compared to ping. Also, we can see that SPP only measures the RTTs experienced by SSH and does not provide RTTs if there is no traffic (e.g. the SSH session was idle between 75 and 125 seconds) whereas ping regularly samples the path. A CDF of the SPP measurements characterises the RTTs experienced by the application whereas a CDF of the ping measurements characterises the RTTs of the path over time. For applications that do not continuously send traffic SPP provides a more accurate picture. Note that SPP can always be used for regular measurement simply by generating test traffic.

3) Challenges with active probing: Detailed insights can also be obtained using high probe rate active measurements, but to the potential detriment of the path under observation.

Consider the example in subsection V-C1. Active probing at a similar rate would require 30 pings/second. Each 64-byte ICMP Echo request becomes an 84-byte IP packet. After the usual ADSL-based Internet service overheads this becomes three 53-byte ATM cells. Sending 90 x 53-byte ATM cells per second is 38 kbps, or 6% of the 653 kbps upstream capacity.

Paths with 802.11 wireless LAN links are also challenging, as such links are sensitive to both the bit-rate and packet-rate of any active probe traffic. For example, [6] showed how TCP over 802.11b links networks could lose roughly 25% of achievable throughput (4 Mps to 3 Mps) in the presence of 100 pings/second (under 70 Kbps of regular ping traffic), and 50% of achievable throughput (down to 2 Mbps) in the presence of 333 pings/second (under 200 Kbps of regular ping traffic).

D. Measuring the RTT experienced by traffic of interest

It is becoming common for networks to apply different traffic prioritisation or routing (load-balancing) rules to packets belonging to different application flows. This can prevent active probing from 'seeing' application-specific RTTs. In contrast, SPP will calculate the actual RTT experienced by the traffic observed at each MP.

1) Observing the RTT of different traffic classes: Traffic prioritisation is increasingly used to minimise the latency for specific traffic through bottleneck routers. The actual classification and differentiation of flows may occur using IP addresses, TCP/UDP port numbers, direct payload inspection or the traffic's statistical properties (such as packet length distributions and inter-packet intervals [26]). Active probes will be unable to measure the RTT experienced by the prioritised traffic, since the active probe packets are unlikely to meet the classifiers rules for header fields (addresses and ports) let alone statistical properties. On the other hand, SPP can measure the prioritised traffic's RTT directly.

As an example, we classified traffic through an OpenWRT-based home router into interactive and non-interactive classes based on packet length statistics [27] and prioritised the interactive traffic on an emulated 1 Mbps ADSL upstream link (using Dummynet). We simultaneously generated FPS game traffic with ET, TCP cross-traffic with Iperf (to congest the router), and actively measured RTT with ping. Figure 10 shows a 20-second time window of game and TCP flow RTTs (measured with SPP) and ping's RTT measurements.

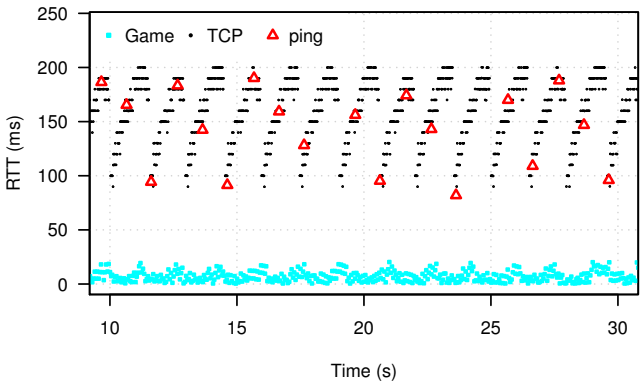


Figure 10: RTT of prioritised FPS online game traffic and TCP cross traffic measured by SPP, and RTT measured by ping (not classified as interactive traffic)

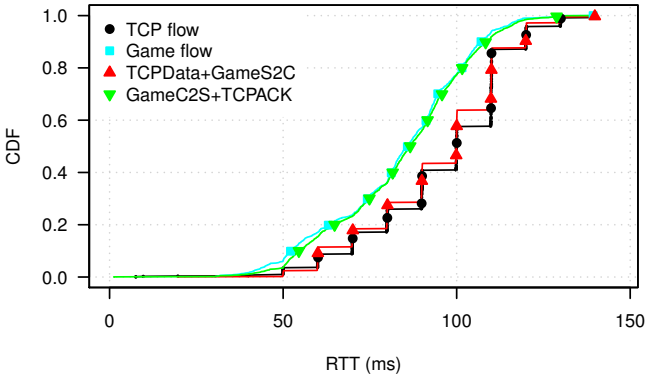


Figure 11: CDFs of RTTs measured by SPP of TCP flow, game flow, TCP data and game server-to-client traffic, TCP ACK and game client-to-server traffic

Ping sees the cyclical high RTT caused by the TCP cross-traffic, but misses both the detailed RTT fluctuations induced by TCP and the low RTT experienced by the prioritised game traffic.

2) Impact of load-balancing: Active probing can also miss the impact of router load-balancing. By way of example, we simultaneously generated a single bidirectional UDP flow (60 byte packets, 10 packets/second) and ran regular ping (one packet/second) between a host in Canberra, Australia ($\mathrm{MP}_{\mathrm{ref}}$) and a host in Berlin, Germany ($\mathrm{MP}_{\mathrm{mon}}$). SPP revealed that the UDP traffic experienced an almost constant RTT of 318 ms. In contrast, 25% of ping's RTTs were 313 ms, 50% were 318 ms, and 25% were 323 ms.

At the time traffic from Australia to Europe traversed one of two undersea cables of different lengths linking Australia to Singapore. The routers at either end used a load-balancing algorithm, which distributed ICMP ping packets randomly on both links on a per-packet basis in either direction. Ping's three different latencies reflected the three possible path combinations and their 2:1:1 relative frequencies. In contrast, the load-balancing algorithm chose one link in each direction for the duration of UDP or TCP flows, giving a more stable RTT.

E. RTT measurements using unrelated traffic

To demonstrate that SPP can measure RTT using unrelated traffic in each direction we recreated the experiment in Section V-D1 but this time *without* prioritising the game traffic. We then generated two sets of $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ traces that contain unrelated traffic: one set consisting of only the TCP data traffic and game server-to-client traffic (*TCPData+GameS2C*), and one set consisting of only the TCP ACK traffic and game client-to-server traffic (*GameC2S+TCPACK*).

Figure 11 shows CDFs of the RTTs derived by SPP from the TCP traffic alone (*TCP flow*), the bidirectional game traffic alone (*game flow*), the TCPData+GameS2C traffic, and the GameC2S+TCPACK traffic. The RTT distributions of the game flow and GameC2S+TCPACK traffic are identical. The RTT distributions of the TCP flow and TCPData+GameS2C traffic are almost identical. Both game traffic packets and TCP ACKs are much smaller than TCP data packets, thus RTTs measured using the TCP flow or TCPData+GameS2C traffic are higher on average. The RTTs with TCP data packets are quantised on 10 ms intervals, since TCP data packets were always queued at least until the next timer tick due to the 1 Mbps upstream link and the underlying OpenWRT operating system's packet queueing using 10 ms time slices.

The difference between the game flow and TCP flow RTT distributions in Figure 11 also highlights that RTTs can depend on packet size, especially on low-capacity links with high serialisation delays and/or (wireless) shared links with high contention delays. Active probes will not measure the RTT experienced by application flows, unless the probe packets are of similar sizes to the application's packets.

VI. Conclusions and Future Work

Evaluating the performance of application protocols, transport protocols or network equipment often requires accurate RTT measurements, both for researchers and service providers. Active measurement techniques are useful within limits – they add traffic in proportion to

their sample (probe) rate, and probe packets may experience different RTTs to regular traffic. Previous passive measurement techniques require precisely synchronised clocks at different measurement points or application traffic having symmetric request-response behaviour.

We proposed and demonstrated a novel passive technique called 'synthetic packet-pairs' (SPP) that measures the RTT of a network path using whatever symmetric or asymmetric two-way flow of traffic exists between two unsynchronised measurement points (MPs). We introduced an open-source implementation of SPP, which can combine traffic observations from measurement points in near real-time or offline (using traffic captured at an earlier time).

We showed that SPP provides accurate RTT measurements at a rate proportional to the two-way traffic being observed. This enables tracking RTT fluctuations over link technologies that are sensitive to excess traffic loads, measuring latency on paths where active probe packets are otherwise treated differently to regular traffic, and seeing short-term RTT transients that are invisible to low-rate active probing.

SPP does have limitations. It may not be possible to place MPs at either end of the entire path to be measured. Also, SPP alone cannot sample idle paths. Nevertheless, it provides many benefits for measuring RTTs when two or more MPs can be placed at useful locations in the network.

In future work we plan to investigate if comparing RTT estimates from multiple packet-pairs that have an initial packet in common will allow inferring one-way delay trends.

Acknowledgements

We thank Thuy Nguyen, Lutz Mark and Brandon Tyo for contributing to the initial SPP development, Amiel Heyde for implementing the public SPP prototype and carrying out the SSH experiments, David Hayes, Atwin Calchand and Chris Holman for subsequent bug fixes to SPP, and Nigel Williams for helping with the OpenWRT traffic prioritisation experiments.

References

[1] G. Armitage, M. Claypool, P. Branch, *Networking and Online Games – Understanding and Engineering Multiplayer Internet Games*. John Wiley & Sons, April 2006.

[2] K. Auerbach, "Limitations of ICMP Echo for network measurement." InterWorking Labs, April 2004. http://iwl.com/white-papers/75-limitations-of-icmp-echo-for-network-measurement.

[3] S. Keshav, "Packet pair flow control," 1995. http://blizzard.cs.uwaterloo.ca/keshav/home/Papers/data/95/pp.pdf.

[4] A. Heyde, "SPP Implementation." http://caia.swin.edu.au/tools/spp/.

[5] S. Bortzmeyer, "echoping Home Page." http://echoping.sourceforge.net/.

[6] T. Nguyen and G. Armitage, "Quantitative Assessment of IP Service Quality in 802.11b Networks and DOCSIS networks," in *Australian Telecommunications Networks & Applications Conference (ATNAC)*, pp. 121–128, December 2004.

[7] I. D. Graham, S. F. Donnelly, S. Martin, J. Martens, J. G. Cleary, "Nonintrusive and Accurate Measurement of Unidirectional Delay and Delay Variation on the Internet," in *Internet Summit (INET)*, July 1998.

[8] T. Zseby, S. Zander, G. Carle, "Evaluation of Building Blocks for Passive One-way-delay Measurement," in *Passive and Active Measurement Workshop*, April 2001.

[9] S. Niccolini, M. Molina, F. Raspall, S. Tartarelli, "Design and Implementation of a One Way Delay Passive Measurement System," in *9th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2004.

[10] V. Paxson, "On Calibrating Measurements of Packet Transit Times," in *ACM SIGMETRICS*, pp. 11–21, June 1998.

[11] M. Davis, B. Villain, J. Ridoux, A.-C. Orgerie, D. Veitch., "An IEEE-1588 Compatible RADclock," in *International IEEE Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS)*, pp. 7–12, 2012.

[12] M. Hassan, J. Wu, "APM: Asynchronous Performance Measurement for the Internet," in *6th Asia-Pacific Conference on Communications (APCC)*, 2000.

[13] R. R. Kompella, K. Levchenko, A. C. Snoeren, G. Varghese, "Every Microsecond Counts: Tracking Fine-grain Latencies with a Lossy Difference Aggregator," in *ACM SIGCOMM Conference on Data Communication*, pp. 255–266, 2009.

[14] M. Cola, G. De Lucia, D. Mazza, M. Patrignani, M. Rimondini, "Covert Channel for One-Way Delay Measurements," in *International Conference on Computer Communications and Networks*, August 2009.

[15] L. D. Vito, S. Rapuano, L. Tomaciello, "One-Way Delay Measurement: State of the Art," *IEEE Transactions in Instrumentation and Measurement*, vol. 57, pp. 2742–2750, December 2008.

[16] J. Jiang, C. Dovrolis, "Passive Estimation of TCP Round-trip Times," *ACM Computer Communication Review (CCR)*, vol. 32, pp. 75–88, 2002.

[17] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, D. Towsley, "Inferring TCP Connection Characteristics Through Passive Measurements," in *IEEE INFOCOM*, March 2004.

[18] B. Veal, K. Li, D. Lowenthal, "New Methods for Passive Estimation of TCP Round-Trip Times," in *Passive and Active Measurement Workshop*, March/April 2005.

[19] J. But, U. Keller, D. Kennedy, G. Armitage, "Passive TCP Stream Estimation of RTT and Jitter Parameters," in *IEEE 30th Conference on Local Computer Networks (LCN)*, November 2005.

[20] D. Carra, K. Avrachenkov, S. Alouf, A. Blanc, P. Nain, G. Post, "Passive Online RTT Estimation for Flow-Aware Routers Using One-Way Traffic," in *NETWORKING 2010*, pp. 109–121, 2010.

[21] S. Zander, G. Armitage, T. Nguyen, L. Mark, B. Tyo, "Minimally Intrusive Round Trip Time Measurements Using Synthetic Packet-pairs," Tech. Rep. 060707A, Centre for Advanced Internet Architectures, Swinburne University of Technology, 2006.

[22] M. Molina, S. Niccolini and N.G. Duffield, "A Comparative Experimental Study of Hash Functions Applied to Packet Sam-

pling," in *International Teletraffic Congress (ITC-19)*, August 2005.
[23] C. Henke, C. Schmoll, T. Zseby, "Empirical Evaluation of Hash Functions for PacketID Generation in Sampled Multipoint Measurements," in *Passive and Active Measurement Conference (PAM)*, pp. 197–206, 2009.
[24] T. Kohno, A. Broido, kc claffy, "Remote Physical Device Fingerprinting," in *Proceedings of IEEE Symposium on Security and Privacy*, pp. 211–225, May 2005.
[25] T. Zseby, M. Molina, N. Duffield, S. Niccolini, F. Raspall, "Sampling and Filtering Techniques for IP Packet Selection." RFC 5475 (Proposed Standard), March 2009.
[26] T. Nguyen, G. Armitage, "A Survey of Techniques for Internet Traffic Classification using Machine Learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.
[27] S. Zander, G. Armitage, "DIstributed Firewall and Flow-shaper Using Statistical Evidence (DIFFUSE)." http://caia.swin.edu.au/urp/diffuse.

APPENDIX

In this appendix we describe in more detail the overall design of the SPP prototype and the format of the SPP Sample Frames (SSF) protocol.

A. Overall Design

Figure 12 shows the block diagram of the SPP prototype. The overall SPP Tool block is responsible for the command line argument processing, the creation of threads and the main loop. The Packet Processing block reads packets from a network interface and generates the [PID,ts] pairs. The Packet Matching block matches the packet lists from $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$. The Packet Pairing block identifies the packet pairs based on the $[\mathrm{PID}_1,\mathrm{ts}_{\mathrm{ref}},\mathrm{ts}_{\mathrm{mon}}]$ and $[\mathrm{PID}_2,\mathrm{ts}_{\mathrm{mon}},\mathrm{ts}_{\mathrm{ref}}]$ lists. The Slave block exports the generated [PID,ts] pairs to the SPP Master. The Master block receives the [PID,ts] pairs send by the slave and passes them to the Packet Matching block.

The CRC32, Timeval and SSF Protocol blocks contain support functionality used by the other building blocks. CRC32 is used by the Packet Processing to compute the packet IDs, Timeval is mainly used by Packet Pairing to compute the RTTs, and SSF Protocol implements the SSF protocol (described in Appendix B) used by the Slave and Master blocks.

The SPP implementation makes use of pthreads (POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995)) for concurrent processing, which is especially useful if SPP runs on multi-core CPUs. The thread configuration depends on the mode SPP is running in.

An SPP Slave uses two threads. The first thread runs the Packet Processing, and the second thread sends the SSF protocol messages to the SPP Master (Slave block). An SPP Master uses three threads. The first thread runs the Packet Processing, the second thread receives and processes the SSF protocol messages from the SPP Slave (Master block), and the third thread runs the Packet Matching and Packet Pairing. If SPP is used in offline mode it uses three threads. The first two threads run the Packet Processing for $\mathrm{MP}_{\mathrm{ref}}$ and $\mathrm{MP}_{\mathrm{mon}}$ respectively, while the third thread runs the Packet Matching and Packet Pairing.

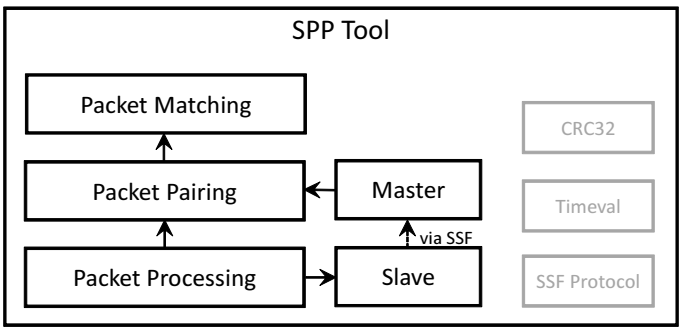


Figure 12: Block diagram of SPP prototype

B. SSF Protocol

The SSF protocol is a simple protocol to transport the [PID,ts] pairs from the SPP Slave to the SPP Master. SSF uses UDP as underlying transport protocol. Currently, SSF does not provide a reliable data transport and does not perform congestion control. We plan to support SCTP or TCP as alternative for reliable congestion-controlled SSF transport in the future.

The design of the SSF protocol is based on the Real-Time Transport Protocol (RTP) specified in IETF RFC 3550. Using the RTP protocol as basis provides some advantages. For example, the SSF traffic could be prioritised in the network similar to traffic prioritisation for audio/video RTP flows and for debugging we can utilise existing RTP decoders, such as the one implemented by Wireshark (http://www.wireshark.org).

Each SSF/UDP packet has a 12-byte fixed header shown in Figure 13. The first two bits of the header are the version number (V). The current SSF version is 3. The following 7 bits are reserved for future use. The next 7 bit are the Format Code, which specifies the length of the delta timestamps in bytes including the D bit (see below). The default size of the delta timestamps including the D bit is 16 bits, so the default value of the Format Code is 2. The next 16 bit are the Sequence Number. The SPP Slave increments the sequence number for each SSF packet sent. The sequence number allows the SPP Master to detect lost [PID,ts] pairs. Note that although the protocol has a sequence number field, the

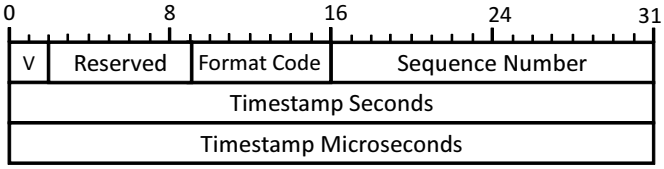


Figure 13: SSF protocol header

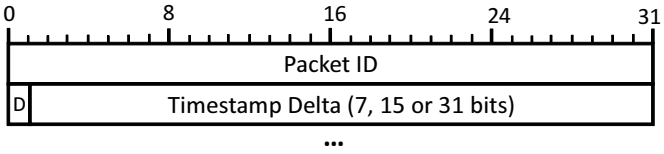


Figure 14: Packet ID, timestamp pair entries

current prototype version 0.3.4 does not use it. Finally, the header contains two 32-bit fields that represent the base time for all [PID,ts] pairs in the SSF packet. The first 32 bit are the number of seconds since epoch, and the second 32 bit contain the number of microseconds.

The rest of a SSF/UDP packet is used by the [PID,ts] pairs (Figure 14). The 32-bit packet ID field acts as "unique" identifier for packets. The direction bit (D) specifies whether a packet went from $\mathrm{MP}_{\mathrm{ref}}$ to $\mathrm{MP}_{\mathrm{mon}}$ (D set to 0) or from $\mathrm{MP}_{\mathrm{mon}}$ to $\mathrm{MP}_{\mathrm{ref}}$ (D set to 1). The Timestamp Delta specifies the time offset relative to the base timestamp defined in the header of when a packet was observed. Adding the offset to the base time gives the time the packet was observed. By default the Timestamp Delta granularity is 100 microseconds, but this can be configured. The size of the Timestamp Delta is also configurable and can be set to 7, 15 or 31 bits (default is 15 bits). This means depending on the size of the Timestamp Delta and the selected granularity one SSF packet can cover different time periods. For example, with the default 100 microsecond granularity, one SSF packet covers periods of ~13 ms, ~3.2 seconds or ~59.6 hours for Timestamp Delta's of 7, 15 and 31 bits respectively.

The variable Timestamp Delta allows to tune the trade-off between the number of packets sent (the overhead of the SSF/UDP/IP headers) and the size of [PID,ts] pair entries. For example, if the rate of observed packets is high, at a given granularity we can choose a smaller Timestamp Delta and SSF packets still contain the maximum number of [PID,ts] pairs (given the maximum packet size), and thus we can minimise the header overhead. However, if the rate of observed packets is low, a larger Timestamp Delta would help to avoid sending half-empty SSF packets. The granularity can be used to tune the tradeoff between the bandwidth required by SSF and the required timestamp/RTT precision. For a given Timestamp Delta size, we can choose the lowest acceptable timestamp precision, which increases the time coverage of SSF packets and ensures SSF packets contain as many [PID,ts] pairs as possible (given the maximum packet length).

Note that independent of the Timestamp Delta settings an SPP Slave will send an SSF packet at least every t seconds, where t is configurable (default is 10 seconds).