

Minimally Intrusive Round Trip Time Measurements Using Synthetic Packet-Pairs

Sebastian Zander*, Grenville Armitage*, Thuy Nguyen*, Lutz Mark[†], Brandon Tyo*
Centre for Advanced Internet Architectures (CAIA). Technical Report 060707A
Swinburne University of Technology
Melbourne, Australia
{szander,garmitage,tnguyen,btyo}@swin.edu.au, mark@fokus.fraunhofer.de

Abstract- In this paper we describe a passive technique for round trip time (RTT) estimation called Synthetic Packet-Pairs (SPP). Regular and frequent measurement of round trip time (RTT) between points on the Internet is becoming increasingly important for a range of highly interactive real-time applications. Active probing techniques are possible but problematic. The extra packet traffic imposed by active probes along a network path can modify the behaviour of the network under test. In addition, estimated RTT results may be misleading if the network handles active probe packets differently to regular IP packets. In contrast, SPP provides frequently updated RTT estimates using IP traffic already present in the network. SPP estimates the RTT between two measurement points without requiring precise time synchronisation between each point. SPP accurately estimates the RTT experienced by any application's traffic without needing modifications to the application itself or the routers along the path. In addition, SPP works with applications that do not exhibit symmetric client-server packet exchanges (such as many online multiplayer games) and applications generating IP multicast traffic. Given the popularity of 802.11 Wireless LANs, and their sensitivity to the load imposed by active probing schemes, we experimentally demonstrate the advantages of SPP in a small 802.11b test bed.

Keywords- Minimally Intrusive, Passive measurement, Round Trip Time, RTT, Delay.

I. INTRODUCTION

It is becoming increasingly important for service providers to monitor and manage the network-level round trip time (RTT) experienced by highly interactive applications. This is not simply voice or video over IP. The emergence of popular, multiplayer online games (for example, the latency-sensitive first person shooter (FPS) genre [1][2]) and mission-critical business applications provides ISPs with motivation to know precisely how their network paths are behaving. Measuring the time-varying RTT actually experienced by an application is a substantial challenge. It is particularly challenging if the RTT fluctuates more frequently than you can sample the path, or you do not have complete access to every hop of the path over which your application is running.

One common approach involves actively probing the path. Extra packets are injected into the network and their transit times used to estimate (sample) network delay at the instant each active probe was sent. A path's RTT is calculated by summing the actively measured delay in each direction. However, active probe traffic adds a finite, non-negligible load on the network itself (proportional to the probing or sampling rate) and certain types of active probe packets may not experience the same per-hop delays as the IP packets belonging to regular applications [3].

Another approach is passive measurement. This involves measuring the delay experienced by traffic already present in the network, and thus does not add extra load on the network. One technique measures one-way delay (OWD) by noting the time it takes an arbitrary packet to transit between two precisely synchronised measurement points [4-6]. Another technique directly estimates RTT at a single measurement point from the time it takes for an application's request packet in one direction to be answered by a matching (and expected) response packet in the return direction [7-9].

In this paper we introduce synthetic packet-pairs (SPP). SPP is a novel passive measurement technique that:

- Estimates RTT between two passive measurement points on the network, using traffic already flowing between the two measurement points.
- Does not require precise synchronisation between the clocks at each measurement point.
- Does not require symmetric or triggered request-response behaviour from the application-driven traffic being used to sample the network path.
- Effectively 'samples' the RTT as frequently as packet pairs occur between the measurement points.
- Is minimally intrusive because no changes are required to IP packet contents or application-layer payloads between the measurement points.

* Centre for Advanced Internet Architectures, Swinburne University, Melbourne Australia

[†] Autonomic Networking Technologies, Fraunhofer FOKUS, Berlin, Germany

SPP has applications wherever the network under test may be sensitive to additional load (ruling out active probing) or synchronised measurement points are unavailable (for reasons of cost or real-world deployment issues). SPP is also advantageous where ongoing RTT estimates are desired for jitter estimation, particularly for interactive applications that do not generate precisely symmetric request-response packet pairs.

Although SPP is link layer-agnostic, we conclude this paper with an illustration of SPP being used to estimate RTT across a small 802.11b Wireless LAN (WLAN) test bed. WLANs are problematic in general because they are (in various forms) being increasingly asked to carry interactive traffic and yet they are highly sensitive to the load potentially imposed by active probing schemes [10]. We first illustrate the benefits of SPP relative to active probing when the WLAN is carrying heavy TCP traffic. Then we demonstrate the use of SPP to measure RTT experienced by an application whose client-server traffic patterns are asymmetric (an online first person shooter computer game).

Our paper is organized as follows. Section II discusses related active and passive measurement work. Section III presents our SPP algorithm. Section IV discusses sources of RTT estimation error and some issues associated with realistic deployment of SPP implementations. Section V illustrates the benefits of our approach in the context of a live WLAN. Section VI concludes and outlines future work.

II. ACTIVE VS. PASSIVE MEASUREMENTS

Before introducing SPP we summarise the key attributes of existing active and passive RTT measurement schemes.

A. Active Measurements

Active measurements involve the controlled injection of extra traffic into the network and monitoring of the network's subsequent behaviour. Traffic may be injected with various patterns adapted to specific measurement objectives and to emulate particular applications. For example, single packets may be injected with uniform spacing in time or short bursts of packets may be injected with varying packet sizes and inter-packet intervals within the burst. Unfortunately, active measurement has a number of disadvantages. The network may treat active probes differently (leading to unrepresentative measurements) and many tools fail to emulate realistic application traffic patterns. For example, people frequently use 'ping' as a simple active measurement tool even though routers often handle ICMP packets in their slow path (leading to overestimation of RTT) [3].

Active measurements also generate additional load on the network. This load can alter the network's overall behaviour and performance during the measurement period. Unfortunately, the more precisely you wish to track latency variations the more frequently you must inject active probe packets into the network. This induces further deviations in the characteristics of the network under test. Active probing is particularly

problematic over link technologies such as 802.11 WLANs, 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).

B. Passive Measurements

In contrast, passive measurements utilise traffic already in transit across a network. Delay is measured as experienced by the application whose packets are being monitored, and the network is not influenced by additional injected traffic. Two classes of passive delay measurements have been previously described in the literature.

Passive one-way delay (OWD) measurements (e.g. [4], [5] and [6]) require the observation of individual packets passing between two measurement points. OWD in each direction is calculated directly from the times at which a given packet passes each measurement point. A major challenge for OWD is the need for precisely synchronised time stamping clocks at each measurement point. Synchronisation is required to compensate for drifting of one measurement point's clock relative to the other. The challenge arises when each measurement point is in a separate room, city or even country. A simple approach would be to use the network time protocol (NTP). NTP is often accurate to 1ms, but can be far worse subject to network conditions and NTP server outages [11]. Precise, distributed synchronisation typically uses GPS. However, the cost of infrastructure (such as connecting all measurement points to external, roof-mounted GPS antennas) can make this a difficult choice.

Techniques for measuring OWD without accurate clock synchronisation have been proposed [12], but the approach requires a modification of routers along the path and thus cannot be easily deployed on existing operational infrastructure. A proposal to achieve high timing accuracy of PC end hosts without GPS is proposed in [13]. This technique provides highly accurate clock rate synchronisation but not absolute time synchronisation (for which the author's simply suggest NTP).

A number of techniques have also been proposed for estimating RTT at a single measurement point [7], [8] and [9]. RTT is calculated from the time between a request packet being seen heading towards a distant server, and a matching reply packet coming back from the same server. Request/response packet-pairs are matched based on well-known fields in the packet header or payload (e.g. sequence numbers in TCP or ICMP echo packets). Such techniques are limited to scenarios where a request packet in one direction 'instantly' triggers a uniquely identifiable response packet in the return direction. Error is introduced into the estimated RTT if response packets are delayed – either due to application layer processing on the server or due to packet loss and retransmission.

Single measurement point techniques are limited by the fact that two-way traffic does not always exhibit identifiable or actual request/response behaviour. An application layer protocol's packet syntax may not

include enough information to actually match request/response pairs. Alternatively, the application's client and server communication may also have no particular temporal relationship between packets flowing from client to server and vice versa. For example, common multiplayer FPS games emit packets in each direction asynchronously, at different rates and with unpredictable intervals (of up to 10s of milliseconds) between packets in each direction [14]. Single measurement point RTT estimates under such circumstances would fluctuate as the apparent server response time varied unpredictably.

III. PASSIVE RTT MEASUREMENT WITH SYNTHETIC PACKET-PAIRS

Our synthetic packet-pairs (SPP) approach strikes a middle ground. SPP estimates RTT (rather than OWD) using application-independent packet flow statistics gathered passively at two independent measurement points. SPP samples the path at a frequency proportional to the rate at which the application sends and receives packets (thus providing a detailed record of latency fluctuations over time). Unlike previous OWD schemes, SPP does not require precise synchronisation between each measurement point. Unlike previous single measurement point schemes, SPP functions quite well using two-way packet traffic generated by applications that lack symmetric request/response behaviour. SPP is minimally intrusive in that it does not modify or delay packets passing between the measurement points.

In this section we describe the matching of packets seen at both measurement points and the identification of appropriate packet-pairs for RTT estimation. Section IV discusses sources of RTT estimation error and some issues associated with realistic deployment of SPP implementations.

A. Two-point Measurements and Packet Matching

Figure 1 illustrates the basic SPP architecture (based on the OWD measurement techniques proposed in [4], [5] and [6]). SPP estimates the RTT between two measurement points (MP1 and MP2), which must be located so that the network traffic of interest traverses both measurement points. MP1 passively records the passing of packets heading towards MP2 (for example, monitoring packet traffic using mirrored ports on a switch or in-line network taps). MP2 performs the same action for packets heading towards MP1.

MP1 and MP2 independently log two things for every recorded packet: (a) the time at which the packet was seen, and (b) a short 'packet ID' calculated from a hash function (e.g. CRC32) across key bytes within the packet (rather than store copies of each entire packet). To uniquely represent a packet that has passed both MP1 and MP2 the packet ID (referred to from now on as $\langle \text{pkt_id} \rangle$) is based on portions of a packet that are invariant during transit between MP1 and MP2 but vary between different packets.

Each measurement point accumulates a list of $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs based on the captured packets. These two lists are then brought together to create the

packet-matched lists we ultimately use for packet-pair identification and RTT calculation. Figure 1 shows the $\langle \text{pkt_id}, \text{timestamp} \rangle$ lists being combined at a third location via an out-of-band link. If this link is a physical link or logically isolated channel sharing the same underlying infrastructure as the IP path being monitored the $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs may be brought together as they are generated, allowing near real-time estimation of RTT. If no external link is available, $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs may be stored at each measurement point for later transfer across the network being measured (e.g. during an off-peak period when the application of interest is not being used).

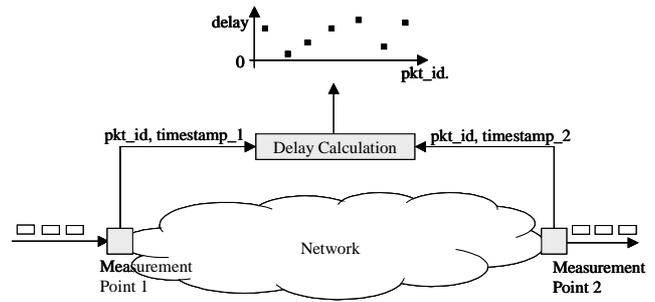


Figure 1: Packet Matching and Delay Computation

To describe the packet-matching algorithm we define MP1 as the 'reference point' and MP2 as the 'monitor point'. Two input streams I_{mon} and I_{ref} represent $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs from packets captured at the monitor and reference points respectively. A queue Q_{ref} is used to buffer $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs from packets captured at the reference point and not yet detected at the monitor point.

The algorithm processes packets from I_{mon} in the order of their arrival at the monitor point. For each packet P_{cur} captured at the monitor point we search for a packet with the same $\langle \text{pkt_id} \rangle$ captured at the reference point. The algorithm first checks if the packet is found in the packet queue Q_{ref} . If the packet is not found in the queue new packets are read from I_{ref} into the queue until a packet matches or the maximum queue length is reached or the packets timestamp differs more than T_{delta} from the timestamp of P_{cur} . (The use of T_{delta} is discussed further in section IV.B) Before the next packet of I_{mon} is processed the packets of Q_{ref} are checked against T_{delta} . All packets whose timestamp differ more than T_{delta} from the timestamp of P_{cur} are considered to be lost packets and removed from the packet queue Q_{ref} . The loss calculation does not start before the first packet matches.

The result is a list of $\langle \text{pkt_id}, \text{timestamp}_1, \text{timestamp}_2 \rangle$ tuples, representing the time a packet from MP1 to MP2 was seen at MP1 and MP2 respectively. The same packet matching algorithm is also run with the directions reversed to construct a list of $\langle \text{pkt_id}, \text{timestamp}_2, \text{timestamp}_1 \rangle$ tuples, representing the packets that were seen flowing from MP2 to MP1.

B. Packet Pair Search and RTT Computation

The primary novel contribution underlying SPP is our technique for identifying packet pairs in the absence of any explicit request/response association between the

packets in each pair. SPP then calculates the RTT in a manner that does not require synchronisation between each measurement point.

To explain our packet-pairing algorithm we first define a timestamp as $t_{\langle \text{point} \rangle \langle \text{pkt} \rangle}$ where $\langle \text{point} \rangle$ is the indices of the measurement point (1 or 2) and $\langle \text{pkt} \rangle$ refers to the packet number (whether it is the first (1) or second (2) packet of the pair). For example, t_{11} is the timestamp of the first packet at MP1 (illustrated in Figure 2) and t_{21} is the timestamp of the first packet at MP2.

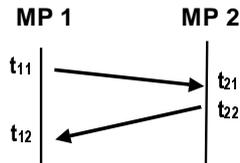


Figure 2: Packet pair and corresponding timestamps

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

$$RTT = (t_{12} - t_{11}) - (t_{22} - t_{21}). \quad (1)$$

RTT calculated in this manner is not influenced by lack of synchronisation between MP1 and MP2 because the calculation is made based on time differences of the same clocks.

Our SPP algorithm makes two key assumptions:

- There is no third packet between the two packets of a pair. A 'packet between' is defined as packet where the timestamp at MP1 is between t_{11} and t_{12} and its timestamp at MP2 is between t_{21} and t_{22} .
- Each packet is used in at most one pair. A packet may not be used in any pair if the first assumption is violated.

The aim is to avoid overlapping packet pairs and ensure the two packets of a pair are as close together as possible.

SPP starts with the first packet from the list of packets going from MP1 to MP2. (The two directions can be reversed and the RTTs can be computed in the other direction if required.) We then search in the second packet list (packets going from MP2 to MP1) for the first packet where the condition $t_{22} > t_{21}$ is true. A packet pair has now been identified (see Figure 3) but this pair is not necessarily the closest pair.

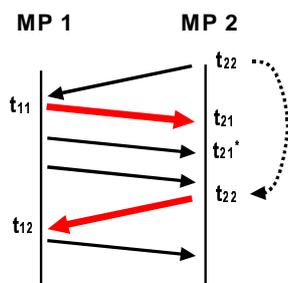


Figure 3: Finding a packet pair

To find the closest pair, the algorithm traverses again through the first list in search for any packets where ($t_{21}^* > t_{21}$ and $t_{21}^* < t_{22}$). As long as such packets are found the first packet is advanced (t_{21}^* becomes t_{21}). This ensures there are no other packets between the packet pair (see Figure 4). The RTT can be computed for this pair and the algorithm continues with the next packet in the first list (packets from MP1 to MP2).

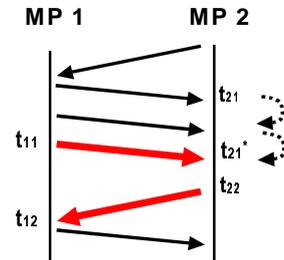


Figure 4: Finding the closest packet pair

IV. ACCURACY AND DEPLOYMENT CONSIDERATIONS

Although SPP does not require precisely synchronised clocks and is minimally intrusive, there are a number of issues to be considered.

A. Accuracy of the RTT Estimates

There are a number of considerations with respect to accuracy of RTT estimates made using SPP.

As noted earlier, equation 1 cancels out the effect of long-term clock drift between the clocks at MP1 and MP2. For this reason we observe that SPP does not require synchronised clocks. However, short-term drift of each clock impacts the consistency of RTT estimates. Specifically, drift of MP1's clock in the interval $t_{12}-t_{11}$ or MP2's clock in the interval $t_{22}-t_{21}$ may cause successive RTT estimates to differ. In practice the interval $t_{12}-t_{11}$ is unlikely to be more than one or two seconds. Modern PC clocks only drift by a few microseconds over tens of seconds [13] so we estimate this source of error to be of the order of 10 microseconds or less using common PC hardware at the measurement points.

Another potential source of inaccuracy lies with the packet recording and time stamping process itself. Most equipment will have some fixed time delay between seeing a packet and recording the packet's 'arrival time' (for example, through the router's port mirror and the measurement hardware's own packet reception process). Any random, fluctuating component to this interval will appear in equation 1 as fluctuations in the estimated RTT. (A constant delay between arrival and time stamping will have no impact. For example, if MP1 always records a packet's arrival 20 microseconds late then t_{11} and t_{12} will each be offset by 20 microseconds yet interval $t_{12}-t_{11}$ will be unaffected.) Random fluctuations are more likely when using software packet capture on non-realtime operating systems rather than when using dedicated hardware (such as Endace DAG capture cards, with timestamp accuracy in the order of hundreds of nanoseconds [18]).

With the hardware used in section V we found that FreeBSD timestamps packets with accuracy of better

than $\pm 10\mu\text{sec}$ in the kernel (measured with `tcpdump`, and assuming there is no heavy load). This is consistent with [18] reporting the error for software time stamping has a magnitude of tens of microseconds. Therefore, on a modern not heavy loaded PC the expected error is much smaller than 1ms (the granularity typically used for RTT measurements).

Another source of uncertainty is the question of when the path could be said to have a particular RTT. Ideally we would like to say something like ‘this RTT existed at time T’. Each RTT estimate begins at $T = t_{11}$, and ends at $T = t_{12}$, so it might be said that the RTT exists during an interval (t_{11}, t_{12}) . The width of this interval in time depends on two things – the actual path RTT, and the second difference term in equation 1.

Given that many application flows are client-server in nature, we arbitrarily decide that the server is near MP2 and name the second term the ‘server processing time’ (SPT), thus:

$$SPT = (t_{22} - t_{21}). \quad (2)$$

Unlike traditional single measurement point techniques, our SPP approach does not require SPT to be close to zero for useful and consistent RTT estimation. However, finite non-zero values of SPT add uncertainty to the precise point in time at which the path could be said to exhibit each RTT estimate calculated using equation 1.

As previously noted, a number of applications generate independent and asynchronous packet streams in each direction. In such cases, SPT may fluctuate widely from one packet-pair to the next, from almost zero to as large as the interval between packets emitted by the ‘server’. For example, many popular FPS games emit server to client packets at fixed rates such as 30, 50 or 60ms regardless of the client to server traffic (these examples taken from *Half Life 2*, *Quake III Arena* and *Half Life* respectively [14]). If we were passively estimating path RTT using FPS game traffic our SPT could be randomly scattered between 0 and the game server’s fixed inter-packet interval for server to client packets. (Note that under similar circumstances single point measurement techniques would simply be unable to accurately estimate RTT.)

Our current instantiation of SPP does not allow derived packet pairs to overlap in time. Consequently, SPP effectively ‘samples’ the path at most once every $t_{12}-t_{11}$ seconds. In other words, the granularity with which you can sample a path (for example, to estimate jitter) is bounded by $RTT+SPT$. A future refinement of SPP is planned that will loosen this restriction and enable RTT estimates from overlapping packet pairs.

B. Deployment Issues with SPP

A number of deployment choices influence the impact an SPP-based system would have on the network being measured.

In principle each monitoring point could create $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs for every packet passing by, leaving it up to the packet matching post-processing to weed out packets that did not pass both MP1 and MP2.

However, in practice real-time packet filtering should be applied to ensure that $\langle \text{pkt_id} \rangle$ hashes are only being calculated for a reasonable subset of packets passing between MP1 and MP2. For example, knowing one end of a particular application’s traffic flow would allow packet filtering on IP address and TCP/UDP port number before calculating hashes. Packet sampling methods can also be used to further limit the number of packets recorded per second [15].

As previously noted, Figure 1 shows delay calculations occurring at a third location connected to MP1 and MP2 via out-of-band links (or a single out-of-band link if the delay calculations are co-located with one of the measurement points). A key consideration for minimising the impact of SPP on a live network is to reduce the amount of traffic required on the out-of-band link(s). SPP is entirely non-intrusive with respect to the network path being measured if you have physical or logical out-of-band link(s) available and send $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs as each packet is recorded. If you chose to use a logical channel along the same infrastructure being measured then we believe SPP is at least minimally intrusive in the sense that the $\langle \text{pkt_id}, \text{timestamp} \rangle$ lists take up relatively little space. (Alternatively, SPP is also non-intrusive during the measurement period if you bundle up the $\langle \text{pkt_id}, \text{timestamp} \rangle$ lists and send them across the same network but outside the measurement period).

Efficient packet ID generation methods are described and compared in [5] and in [16]. In our first implementation of SPP we calculate a CRC32 hash across unvarying parts of the IP packet header (protocol, source and destination address, and total length) plus the first 20 bytes of the IP payload. We chose CRC32 (also used in [4]) because it is widely known, has low collision probability (less than $1e-9$ for more than 20 bytes input as reported in [16]), and can be very efficiently computed in hardware and software (reported in [16] to be $\sim 300\text{ns}$ on a 1.7GHz Pentium 4m). CRC32 is also one of the functions currently recommended by the IETF packet sampling work group [17] for packet digests.

A modified set of invariant packet header fields would be required if MP1 and MP2 are placed either side of a box performing network address translation (NAT) or acting as a firewall. Our initial implementation’s packet matching would fail under such circumstances – NAT changes IP addresses and UDP/TCP port numbers while packets are in transit and firewalls may manipulate TCP header fields (such as the sequence number).

Our initial implementation used 12 bytes per observed packet (4 bytes per $\langle \text{pkt_id} \rangle$ and 8 bytes to encompass a large range of absolute $\langle \text{timestamp} \rangle$ values). Roughly 121 $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs could be carried in a single TCP frame over a link with 1500 byte MTU. So, for example, monitoring an application generating two hundred packets per second in one direction would create less than 2 full-sized packets per second carrying $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs over the out-of-band link.

Loose synchronisation between the clocks at MP1 and MP2 could allow higher-order bits to be removed from every raw timestamp field, further reducing the number of $\langle \text{pkt_id}, \text{timestamp} \rangle$ bytes per observed packet. (For example, a 32-bit timestamp field would be more than adequate if MP1 and MP2 knew they were always within a few seconds of each other during the measurement period.)

SPP does not particularly care how $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs are sent over the out-of-band link. Implementations may choose TCP, UDP or some other transport protocol. UDP has the advantage of timely delivery with minimal overheads but provides no protection against loss of $\langle \text{pkt_id}, \text{timestamp} \rangle$ pairs. TCP provides reliable transport and is sensitive to congestion on the out-of-band link, but timely information delivery cannot be guaranteed.

Another implementation choice involves the use of T_{delta} in the preliminary packet-matching phase. Although not strictly required, using T_{delta} can drastically improve performance. For each packet observed at the monitor point only a T_{delta} -based time window of packets from the reference point needs to be searched. T_{delta} must be larger than the possible network delay plus any time synchronisation error. If there is no time synchronisation between the monitor points the time window is only limited implicitly via the length of queue Q_{ref} . So although our approach computes RTT without time synchronisation, loose synchronisation can significantly improve the performance of the packet-matching phase. If MP1 and MP2 were built on common PC hardware it would be sufficient to synchronise the clocks once a day (as this would normally keep the clocks within one second of each other). Using NTP is not required.

The impact on RTT and jitter estimation of the number of packets per second passing each measurement point, the configured T_{delta} , and the out-of-band link capacity is subject for further investigation. In particular, we hope to eventually characterise the impact of loose clock synchronisation on our ability to compress the $\langle \text{timestamp} \rangle$ field and optimise the packet-matching phase.

As noted earlier a separate measurement point is required at each end of the path being monitored, for example one at the client and one at the server. Our architecture also supports the simultaneous use of more than two measurement points, for example multicast traffic where packets from one ingress point traverse multiple egress points. In this case the packet matching must be performed between each pair of measurement points. If the network being monitored has multiple ingress/egress points and the goal is to monitor the delay between each pair of them we have a quadratic complexity. This impacts on the scalability of our approach for large numbers of measurement points. However, each packet matching between a pair of measurement points is independent of the packet matching of all other pairs and thus packet matching can be parallelised to increase performance.

V. ILLUSTRATING SPP IN A WLAN CONTEXT

To illustrate the potential utility of SPP we ran simple experiments over a single-hop 802.11b wireless LAN. We chose 802.11b because it is a popular wireless access technology that suffers performance degradation in the face of modest levels of small packet traffic [10]. (A comprehensive performance evaluation of other WLAN technologies is outside the scope of this paper, and not required to illustrate the use of SPP.)

We show two things. First, SPP can be used to track latency fluctuations more precisely, and with less collateral damage, than active probes during bulk data transfers. Second, SPP measures RTT accurately when applied to asymmetric traffic patterns generated by protocols that are not inherently request/response based. We show that SPP creates useful RTT estimates even when the effective SPT is a multiple of the actual RTT.

A. A One-hop 802.11b Wireless Testbed

Figure 5 shows our testbed with one wireless client and one server communicating in infrastructure mode over a Cisco Aironet 1200 access point (AP). Both server and client were 2.4GHz Celeron PCs running FreeBSD 4.9 with an out-of-band Ethernet connection for sharing SPP $\langle \text{pkt_id}, \text{timestamp} \rangle$ lists. The client had a Netgear 802.11b interface located within a few feet of the AP.

Bulk TCP data transfer over the wireless link was achieved with `nttcp` [19]. Asymmetric traffic was generated using the FPS game *Wolfenstein Enemy Territory* (ET) [20]. The lack of routers in our simple topology meant that ‘ping’ was suitable for active probing. All traffic at server and client was captured using `tcpdump` (utilising the packet timestamps generated by the FreeBSD kernel upon packet arrival). RTTs for `nttcp` and ping traffic were computed after each experiment using `OpenIMP` [21], and an implementation of SPP running on the server.

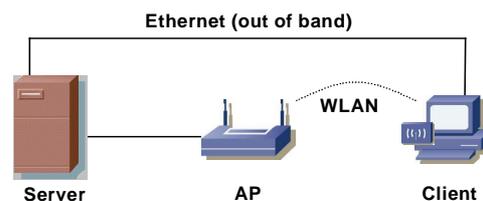


Figure 5: 802.11b wireless LAN testbed

(Strictly speaking the out-of-band Ethernet link was not required. We could have transferred $\langle \text{pkt_id}, \text{timestamp} \rangle$ lists over the wireless link itself after each trial was run. However, an out-of-band link is not unrealistic. In practice an operator might well provision such a link in order to monitor the performance of a fixed, point to wireless link in near-real time or test a new wireless link technologies prior to deployment in the field.)

B. The Negative Consequences of Active Probing

To illustrate the consequences of active probing we ran five 5-minute tests where the link was actively probed during bulk data transfer. Each test involving repeated transfers of 8Mbyte of data from the server to the client. Active probing was achieved by continuously pinging the server from the same client during each bulk data transfer. Different trials used ping intervals of one, 0.05 and 0.01 second respectively. (The 0.01 second interval providing a rough equivalent of SPP's effective sampling rate, since the TCP traffic provided median and peak SPP sample rates of ~160/sec and ~225/sec respectively.)

The cumulative distribution function (CDF) in Figure 6 (one throughput value per 8Mbyte nttcp run) shows the significant degradation of achievable TCP throughput caused by concurrent active probing in an 802.11b context.

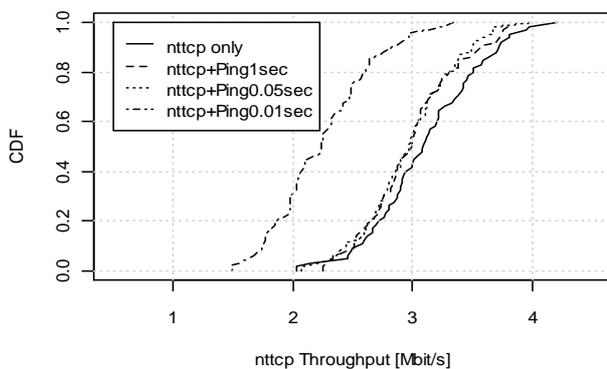


Figure 6: TCP throughput with different ping intervals

With no active probing nttcp achieved a median throughput of just over 3Mbit/sec. Probing at one and 0.05 second intervals caused slight degradation, whilst probing at 0.01 second intervals caused a substantial drop of 1Mbit/sec in the median throughput. (This is noteworthy in light of the fact that pinging every 0.01 seconds represents little more than 51Kbit/sec of active probe traffic, assuming 64 byte ICMP packets.)

During these trials we also observed that the RTT reported by ping increased as the active probe rate increased (median and maximum went from 12ms to 16ms and 26ms to 40ms respectively). Clearly active probing can be a highly disruptive method of obtaining finely grained insight into a link's dynamic latency characteristics.

C. The Relative Accuracy of Active Probing versus SPP

Using tcpdump to capture the traffic during each bulk data transfer allowed us compare the relative accuracy of SPP-derived RTT measurement results with results from active measurement using ping. RTT estimates were calculated using SPP applied separately to ping's ICMP echo request and reply packet traffic, and the TCP Data and ACK packet traffic generated by nttcp.

The CDF in Figure 7 reflects the range of RTTs measured during the trials in Figure 6 with a ping interval of one second. RTT values reported by ping itself are very close to the RTTs estimated by SPP

through passive monitoring of ping's own ICMP packet pairs. This suggests SPP is doing a good job of estimating the RTT experienced by ping packets. However, they both differ from the range of RTTs measured by SPP through passive monitoring of the TCP traffic.

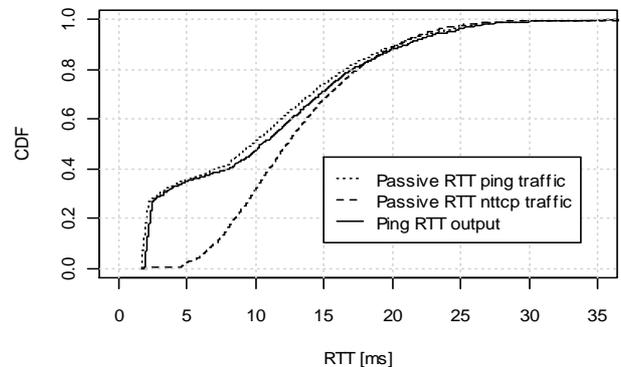


Figure 7: RTTs measured by ping, passive ping monitoring and passive nttcp monitoring (ping 1/second)

This difference reflects the fact that SPP measures the RTT actually experienced by the application (in this case nttcp) whose packets are being passively monitored. As noted earlier, SPP samples the path at a rate proportional to the rate at which the application causes packet pairs to be generated. This ensures that RTT samples are generated across many of the operating conditions nttcp experienced during each 8Mbyte data transfer, and they directly relate to the RTT experienced by nttcp. TCP's congestion window fluctuates throughout each trial, varying the traffic load on the WLAN. During relatively unloaded intervals both TCP and ping experienced low RTTs. As serialisation delay dominates the unloaded link RTT, small ping packets experience lower RTT (1.5-2ms) (and experience it more frequently) than MTU-limited TCP packets (5ms) during such periods. Conversely, when the WLAN is heavily loaded the higher RTTs are dominated by media-access delays and thus both TCP and ping experience very similar network behaviour (in this case when the measured RTT is over 15ms).

Increasing the active probe rate to 0.05s and 0.01s intervals failed to improve ping's ability to 'see' the RTT being experienced by TCP. The WLAN link's performance degraded (as shown in Figure 6) whilst the relationship between actively-probed and SPP-derived RTT estimates looked similar to that shown in Figure 7. (Increasing the active probe packet size to emulate MTU-limited TCP packets would also be self-defeating, simply creating further degradation of WLAN link capacity.)

D. The Utility of SPP When Coupled With Asymmetric Traffic

We also demonstrated SPP using traffic that does not exhibit explicit request/response behaviour. In particular we saw that variations in SPT (from equation 2) have minimal impact on RTT estimates. Our specific example was Wolfenstein Enemy Territory (ET) – an online FPS game where server-to-client packets are sent at fixed 50ms intervals by default and client-to-server packets

are sent at unpredictable intervals between 10ms and 100ms [14]. (There are thousands of online games active across the Internet at any given time, so they are potentially a good source of traffic from which to passively estimate RTT.)

We ran five 5-minute ET game sessions over the 802.11b link and simultaneously actively probed the path with a ping interval of one second. Using SPP we measured RTT and SPT from the game traffic, effectively sampling the link roughly 9 times per second. Figure 8 shows the distribution of RTTs estimated by SPP based on both game and ping traffic, along with the RTTs reported every second by ping itself. As we expected the curves are basically identical (except that ping itself reports slightly higher values due to its time stamping packets in user space rather than kernel space).

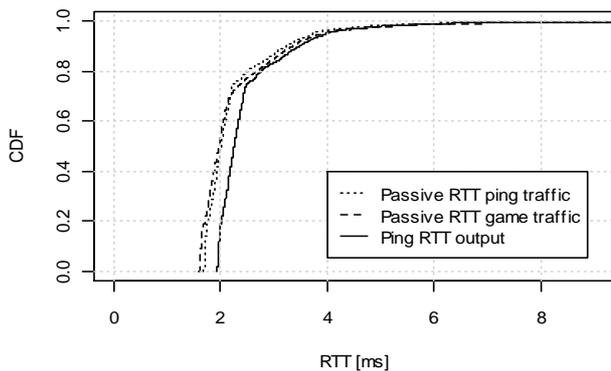


Figure 8: RTTs measured by ping, passive ping monitoring and passive game monitoring (ping 1/second)

The main outcome here is to see how SPP's RTT estimates are unaffected by SPT. Recall that for every packet pair SPP can estimate RTT from equation 1 and SPT from equation 2. Figure 9 is a scatter-plot version of Figure 8, with each estimated RTT plotted against SPT (with 5% and 95% of RTT estimates falling between the dark lines). As the ET client and server are unsynchronised we observed that SPT values range between 0.5ms and 50ms (bounded by the ET server's message interval). The upper end is almost 25 times the median RTT. Nevertheless, across this range of SPT the distribution of passive RTT estimates remains consistent with ping's active probing in Figure 8.

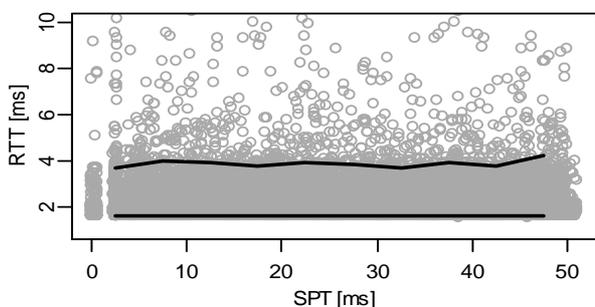


Figure 9: Passive RTTs of game traffic versus SPT

VI. CONCLUSIONS AND FURTHER RESEARCH

Measurement of network path delays is increasingly of interest to network service and application providers. Active measurement techniques are of limited utility, particularly when we wish to closely track latency fluctuations over link technologies that are sensitive to excess traffic loads (such as WLAN environments). Previous passive measurement techniques have required either precisely synchronised clocks at diverse measurement points, or single measurement point tracking limited to applications having specific packet-pair semantics and symmetric request/response behaviour.

We have described a novel approach called synthetic packet-pairs (SPP) that measures RTT of all types of two-way traffic with minimal impact on the network under observation. SPP observes packets at two measurement points whose clocks need not be synchronised together, and samples the path as frequently as the application flow generates unique, non-overlapping packet-pairs. Data from each measurement point may be combined in near real-time using an out-of-band link (for non-intrusive measurement), or combined during off-peak periods using the monitored network itself (for minimally intrusive measurement). SPP enables accurate RTT estimation even when the application traffic under observation exhibits unpredictable delays between packets being sent in each direction (such as multiplayer FPS games).

Further research is possible on a number of fronts. We plan an extension of SPP to utilise overlapping packet pairs, removing the impact of SPT on the effective RTT sample rate. Comparing RTT estimates from multiple packet-pairs that have an initial packet in common will also allow inferring of one-way delay trends. Finally, we are working on a performance evaluation of our approach.

REFERENCES

- [1] S. Zander, G. Armitage. Empirically Measuring the QoS Sensitivity of Interactive Online Game Players. Australian Telecommunications Networks & Applications Conference 2004 (ATNAC2004), Sydney, Australia December 2004.
- [2] G. Armitage. An Experimental Estimation of Latency Sensitivity in Multiplayer Quake 3. 11th IEEE International Conference on Networks (ICON), Sydney, Australia, September 2003.
- [3] K. Auerbach. Why ICMP Echo (Ping) Is Not Good For Network Measurements. InterWorking Labs, http://www.iwl.com/Resources/Papers/icmp-echo_print.html, April 2004.
- [4] 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. Internet Summit (INET), Geneva, Switzerland, July 1998.
- [5] T. Zseby, S. Zander, G. Carle. Evaluation of Building Blocks for Passive One-way-delay Measurement. Passive and Active Measurement Workshop, Amsterdam, The Netherlands, April 2001.
- [6] S. Nicolini, M. Molina, F. Raspall, S. Tartarelli, Design and implementation of a One Way Delay passive measurement system, 9th IEEE/IFIP Network Operations and Management Symposium (NOMS), Seoul, Korea, April 2004.
- [7] N. Brownlee. Packet Matching for NeTraMet Distributions. RTFM Get-Together, IETF Adelaide, <http://www.auckland.ac.nz/net/Internet/rtfm/meetings/>, March 2000.
- [8] J. Jiang, C. Dovrolis. Passive estimation of TCP round-trip times. ACM Computer Communication Review 32, 2002.

- [9] B. Veal, K. Li, D. Lowenthal. New Methods for Passive Estimation of TCP Round-Trip Times. Passive and Active Measurement Workshop, Boston, USA, March/April 2005.
- [10] T.T.T. Nguyen, G. J. Armitage. Quantitative Assessment of IP Service Quality in 802.11b and DOCSIS networks. The Australian Telecommunication Networks and Applications Conference (ATNAC), Sydney, Australia, December 2004.
- [11] V. Paxson. On calibrating measurements of packet transit times. ACM SIGMETRICS, June 1998.
- [12] M. Hassan, J. Wu. APM: Asynchronous Performance Measurement for the Internet. 6th Asia-Pacific Conference on Communications (APCC), 2000.
- [13] A. Pásztor, D. Veitch. PC Based Precision Timing Without GPS. ACM SIGMETRICS, Los Angeles, USA, June 2002.
- [14] G. Armitage, M. Claypool, P. Branch. Networking and Online Games - Understanding and Engineering Multiplayer Internet Games. John Wiley & Sons, UK, (ISBN: 0470018577) April 2006
- [15] K. C. Claffy, G. C. Polyzos, H.-W. Braun. Application of Sampling Methodologies to Network Traffic Characterization. ACM SIGCOMM, San Francisco, CA, USA, September 13-17, 1993.
- [16] M. Molina, S. Niccolini and N.G. Duffield. A Comparative Experimental Study of Hash Functions Applied to Packet Sampling. International Teletraffic Congress (ITC) 19, Beijing, 2005.
- [17] T. Zseby, et al. Sampling and Filtering Techniques for IP Packet Selection. <http://www.ietf.org/internet-drafts/draft-ietf-psamp-sample-tech-07.txt>, work in progress, July 2005.
- [18] S. Donnelly. Endace DAG Time-Stamping Whitepaper. http://www.endace.com/Library/timestamping_whitepaper.pdf, 2006.
- [19] NTTCP, <http://www.freebsd.org/cgi/url.cgi?ports/benchmarks/nttcp/pkg-descr/> (July 2006)
- [20] Enemy Territory, <http://games.activision.com/games/wolfenstein> (July 2006)
- [21] OpenIMP Internet Measurement Project, <http://www.ip-measurement.org/openimp/> (July 2006)