Merging Network Measurement with Data Transport

Pavlos Papageorgiou Michael Hicks
University of Maryland, College Park

Problem

Active measurement tools steal bandwidth away from user data in order to estimate end-to-end path metrics. This overhead may become prohibitive, especially in the presence of multiple active probes or during conditions of congestion.

Tool designers constantly face the tradeoff between measurement overhead and accuracy. Their inherent assumption is that this overhead is unavoidable.

Objective

Minimize the bandwidth that measurement tools consume while maintaining the same level of accuracy and timeliness.

Can we accomplish the same measurement accuracy for less?

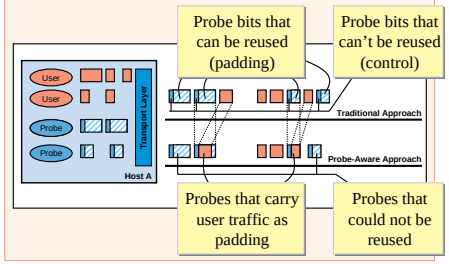
Approach

Probe packets are currently treated in the same way as user packets. But probes are different. They consist mostly of empty padding bits that are useful only *during the journey* of the probe through the network.

A transport protocol that multiplexes user and probe flows can take advantage of this distinction and get more out of every bit.

We can reuse the empty padding bits to carry useful user data bits.

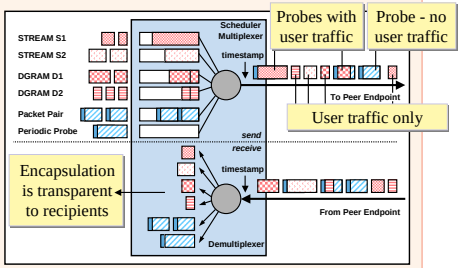
Fig 1. Example of reusing probe padding



Implementation

We have implemented a prototype transport agent which is probe-aware and can reuse the padding in probes to transport user data. It acts as a regular transport layer and exports an expanded BSD socket interface. As far as most applications are concerned, nothing has changed (TCP and UDP as normal).

Fig 2. A probe-aware transport agent



Internally the transport agent multiplexes all flows into a single stream. It applies unified congestion control across all flows that participate. It schedules packets so that there is maximal reuse of the probe padding but always within the constraints of each flow.

API

The transport agent needs to know which part of a UDP probe is padding and what are the constraints of each flow. Measurement tools do this by passing ancillary data to *sendmsg()* every time they send a probe. This is the only change required to existing code.

Table 1. Probe-aware transport API flags

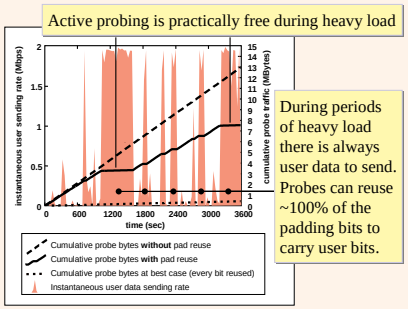
sendmsg() – per packet flags	
PAD_PKT [*s*]	Needs *s* bytes of padding
DELAY [*t*]	Can be delayed up to *t* msec
PKT_TRAIN	Part of packet train
sendmsg() – per session flags	
SINGLE_PKT	Do not encapsulate packets
CON_CONTROL	Under congestion control

Preliminary Results

We present the results of two experiments with real traffic on Emulab. In both, a client host *C* performs file downloads from a server host *S*. At the same time *S* is running four active probe schemes on the path *SC*.

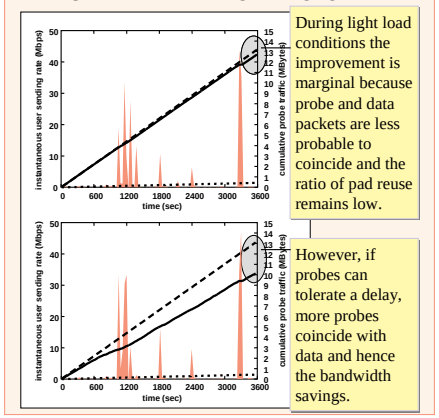
In the first experiment the path has capacity 2Mbps. Notice that the cumulative probe bandwidth levels off when the path is full.

Fig 3. Bandwidth savings during high load



In the second experiment the path capacity is 100Mbps. Notice that we get a noticeable improvement only when probes wait for data

Fig 4. Bandwidth savings during light load



Advantages

- Combines the flexibility of active probing with the benefits of passive measurement.
- Nearly eliminates measurement overhead during periods of high load.
- Minimal changes required to the existing code of measurement tools.
- No changes necessary to applications that do not send probes.
- Tools can increase accuracy by using more probes when there is user data and revert to low bandwidth when there is not.
- Expands the solution space for network measurement techniques. Tools that are bandwidth-intensive become eligible.

Disadvantages

- Works only end-to-end and cannot be used for tools that depend on ICMP.
- No bandwidth savings when no user traffic is flowing on the path.
- Increases the complexity of the transport.

Future Work

- Run existing measurement tools over our transport protocol. Evaluate for accuracy and bandwidth savings. Modify tools to allow probes to delay and be congestion friendly. Does the tool still work?
- Re-examine bandwidth-intensive techniques that may be practical under this new scheme.
- The transport protocol can become a proxy for operations that are common across all measurement tools. One striking example is the echo operation. The transport layer could send a packet, collect timestamps, count duplicates and inform the tool within a specified interval. Such a *measurement primitive* would greatly simplify tool design and enable one-way active probing.