Merging Network Measurement with Data Transport (Extended Abstract)

Pavlos Papageorgiou and Michael Hicks

University of Maryland
pavlos@eng.umd.edu mwh@cs.umd.edu

Abstract. The tasks of measurement and data transport are often treated independently, but we believe there are benefits to bringing them together. This paper proposes the simple idea of a transport agent to encapsulate useful data within probe packets in place of useless padding.

1 Introduction

Overlay networks have become a popular vehicle for introducing network services. Oftentimes, to drive its services, an overlay network infers characteristics of the network via application-layer probes. For example, nodes in RON [1], Detour [2], and Pastry [3] regularly ping their neighbors to check availability and/or measure latency. MediaNet [4] uses available bandwidth [5] estimations to determine along which paths to forward media streams.

For the most part, the measurement and transport aspects of overlay networks are treated independently. The measurement service is a *black box* used by the overlay to make decisions. But the fact that measurement traffic is *in addition* to transport traffic imposes an extra burden on the network. While not a problem for a single overlay under normal conditions, as congestion and/or the number of overlay networks in use increases, measurement traffic begins to influence the total traffic.

To reduce the overhead of measurement traffic, we propose the following simple idea: *merge the task of network measurement with the task of data transport.* In many cases, measurement traffic consists largely of *null padding* just meant to consume bandwidth for timing purposes. This is the case when measuring available bandwidth, for example [5]. To avoid this wasted bandwidth, the transport layer can replace null padding with user payloads available from other streams.

While others have proposed cooperative measurement services [6–8], or observed that network characteristics can be inferred passively [9], no one has proposed merging the tasks of measurement and transport. In this extended abstract, we outline the design and preliminary implementation of a *transport agent* that provides TCP and UDP-like transport along with an enhanced API for sending measurement probes.

2 Probe-Aware Transport Agent

The two goals of our probe-aware transport agent are: (i) to minimize the bandwidth that measurement tools consume and (ii) to allow probe traffic to be responsive to congestion conditions. It is desirable that the API and the end-to-end semantics of user traffic (TCP and UDP) remain intact so that no changes are required to existing applications. Only measurement tools should be required to use the socket API extensions to set encapsulation and dispatch policy for their probes.

The challenge is how to maintain the same measurement accuracy while decreasing the probe bandwidth. The critical observation is that the *null padding*, which dominates probe packets, can be reused without sacrificing the tool's accuracy. The actual pad bytes are irrelevant to the measurement algorithm which means that probe packets can encapsulate user traffic if it is available. This approach satisfies the first goal. To address the second goal, we observe that probing schemes usually do not care about the absolute timing of probe packets but only about the relative timing between packets. Therefore, it should be possible to briefly delay certain probe packets without degrading the tool's accuracy. For example, the transport agent can delay the first packet of a packet train as long as it preserves the inter-packet timing and records the actual departure times.

There is an important trade-off between bandwidth efficiency and the timeliness of probe/user packet transmission. The bandwidth optimization achieved depends directly on how often probes encapsulate user traffic. Congestion conditions will increase this frequency, since it is more likely that user traffic will be buffered and available when probe packets are sent. Thus, probes consume less bandwidth as congestion increases. To improve the optimization under non-congested conditions, we can briefly delay certain probe packets, as directed by the measurement tool, when no user data is available. Conversely, we can delay a user packet when the application would allow it. For example, TCP already delays data to send it in larger chunks.

Socket API Extensions: Our probe-aware transport API extends the BSD-style socket API to define additional flags that affect the way packets are sent, either per-packet or per-session; these flags are presented in Table 1.

Probe packets are sent with the `PAD_PKT [x]` flag enabled. This states that the provided data should be sent with an additional x bytes of padding. Thus, if a probe tool wants to send d bytes of data (i.e., the byte content it wants delivered to the peer tool, such as control information) with x bytes of padding, it would pass only the d bytes to `send`, along with the flag `PAD_PKT [x]`. The transport agent will transport a packet of size $d + x$ bytes, and will attempt to use the x bytes portion of the packet to encapsulate user traffic. When the packet is actually sent, it is timestamped by the transport agent, and timestamped again when it is received. A separate function is used by tools to acquire the times.

The `DELAY [t]` flag can be used to delay the packet up to t ms, to increase the chances of encapsulating user data; otherwise the packet is queued for immediate departure. Note that this flag can be applied to either user or probe traffic. For

Table 1. Probe-Aware Transport API flags

`send` flags	
`PAD_PKT [s]`	Probe packet that requires `s` bytes padding.
`DELAY [t]`	Packet can be delayed up to `t` ms.
`PKT_FOLLOWS`	This packet is not the last packet of a train (others follow).
per-session flags	
`SINGLE_PKT`	Packets should not be encapsulated.
`WAIT_CONGESTION`	Packets under congestion control.

example, to send a probe that waits up to 100 ms would require flags `PAD_PKT[`x`] | DELAY [100]`.

Finally, the `PKT_FOLLOWS` flag is used to indicate that the packet is part of a train, and should not be sent until all packets are available (i.e., a subsequent packet is submitted without this flag). Thus, the entire train may be delayed (by the first packet), but all packets in the train are sent back-to-back.

We also provide two session-level flags. If for some reason encapsulation should be avoided, users can establish sessions using the flag `SINGLE_PACKET`. Probe packets sent in such a session will not encapsulate other packets, and user packets will not be encapsulated by probes. Thus, our transport API semantics reverts to the standard semantics when this flag is set.

Additionally, we provide the flag `WAIT_CONGESTION` to subject sessions to congestion control. By default, STREAM sessions have this flag enabled (to conform to TCP semantics), but DGRAM sessions can specify it as well. This allows probe packets, which are often sent as datagrams, to be accounted for in the congestion window. However, the implementation is non-standard in that we must consider the `DELAY` and `PKT_FOLLOWS` flags when doing congestion accounting.

Figure 1 depicts our preliminary *probe-aware transport agent* which exchanges traffic between two IP endpoints. The transport agent would sit normally on top of IP; our current implementation tunnels over UDP. Internally, the transport agent multiplexes user and probe streams into one packet stream with uniform congestion control, as in the Congestion Manager [10], and encapsulates user traffic in probe packets, unless explicitly disallowed by `SINGLE_PACKET` flags.

For example, consider a measurement tool (session P2 in Figure 1) that periodically sends a probe packet consisting of 10 bytes of control data and 990 bytes of padding. Before the transport agent sends the packet out, it attempts to find user data to fill the 990 available bytes from candidate TCP sessions $B1$, $B2$ and UDP packet streams $D1$, $D2$. If any of them have bytes waiting in their buffers, then up to 990 bytes of user traffic will be encapsulated in the probe.

We have run preliminary experiments with real traffic on Emulab[1] that demonstrated bandwidth savings up to 95% during congestion conditions, i.e., most of the probe traffic piggy-backed on top of user traffic during that period. We are in the process of completing a fully functional implementation and mod-

[1] http://www.emulab.net

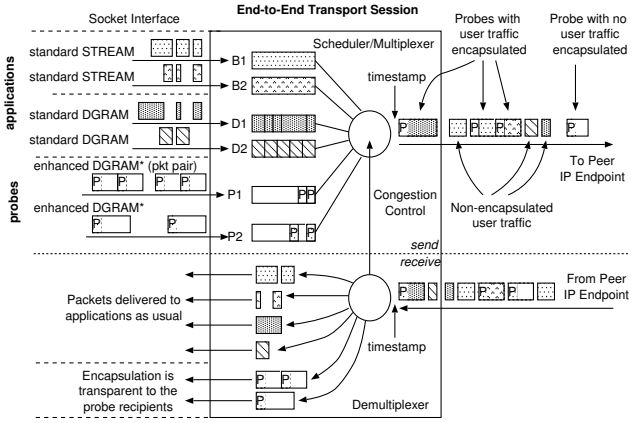

Fig. 1. Overview of a probe-aware *transport agent*

ifying a number of measurement tools to run on top of our transport agent. We intend to run wide-area experiments with real traffic on PlanetLab[2] and continue our performance measurements in the controlled setting of Emulab.

References

1. Andersen, D.G., Balakrishnan, H., Kaashoek, K., Morris, R.: Resilient overlay networks. In: SOSP. (2001)
2. Savage, S., Collins, A., Hoffman, E., Snell, J., Anderson, T.: The end-to-end effects of Internet path selection. In: SIGCOMM. (1999)
3. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001) 329–350
4. Hicks, M., Nagarajan, A., van Renesse, R.: User-specified adaptive scheduling in a streaming media network. In: IEEE Conference on Open Architectures and Network Programming. (2003)
5. Prasad, R., Murray, M., Dovrolis, C., Claffy, K.: Bandwidth estimation: metrics, measurement, techniques, and tools. IEEE Network **17** (2003)
6. Nakao, A., Peterson, L., Bavier, A.: A Routing Underlay for Overlay Networks. In: SIGCOMM. (2003)
7. Srinivasan, S., Zegura, E.: Network Measurement as a Cooperative Enterprise. In: International Workshop on Peer-to-Peer Systems. (2002)
8. Danalis, A., Dovrolis, C.: ANEMOS: An Automomous NEtwork MOnitoring System. In: Passive and Active Measurement Workshop. (2003)
9. Paxson, V.: Automated Packet Trace Analysis of TCP Implementations. In: ACM SIGCOMM. (1997)
10. Balakrishnan, H., Rahul, H.S., Seshan, S.: An Integrated Congestion Management Architecture for Internet Hosts. In: ACM SIGCOMM. (1999)

[2] http://www.planet-lab.org