The Measurement Manifesto

George Varghese, Cristi Estan

Abstract—**Useful measurement data is badly needed to help monitor and control large networks. Current approaches to solving measurement problems often assume minimal support from routers and protocols (e.g., tomography) or place the entire burden on the router to support heavyweight mechanisms (e.g., NetFlow, per-prefix counters). The thesis of this paper is that systems approaches to such problems can yield more efficient intermediate solutions by considering the ultimate use of the data, understanding implementation costs, and by distributing aspects of the solution among routers, protocols, and tools.**

We briefly show how these principles are indirectly applied in existing proposals for new measurement primitives. We then show how these principles can be used to derive new systems solutions to two separate problems: measuring route stability (by modifying route computation) and measuring traffic matrices (by implementing per-class counters that can yield traffic matrices with much smaller memory requirements than per-prefix counters). Beyond specific techniques, we hope the principles in this position paper can provide a focus for discussion among researchers, router vendors, protocol designers, and network operators (the stakeholders in the measurement enterprise) to yield effective solutions to measurement problems.

I. Introduction

Not everything that is counted counts, and not everything that counts can be counted. —- Albert Einstein.

This paper deals with the problem of producing effective measurements to help run large networks more effectively. We focus on novel approaches to *passive measurement* [8]. Passive measurement helps determine the *causes* of network performance problems as opposed to *active measurement* [14], which provides glimpses into the *effects* of network problems on users. Once causes — such as links that are unstable or have excessive traffic – are identified, network operators can take action by a variety of means. Thus measurement is crucial not just to characterize the network but to better engineer its behavior.

There are several control mechanisms that network operators currently have at their disposal. For example, operators can tweak OSPF link weights and BGP policy to spread load, can set up circuit-switched paths to avoid hot spots, and can simply buy new equipment. This paper focuses only on network changes that address the measurement problem — i.e., changes that make a network more *observable*. However, we recognize making a network more *controllable*, for instance by adding more tuning knobs, is an equally important problem we do not address.

Unlike the telephone network, where observability and controllability was built into the design, the very simplicity of the successful Internet service model has made it difficult to observe [5]. In particular, there appears to be a great semantic distance between what users (e.g., ISPs) want to know, and what the network provides. In this tussle [4] between user needs and the data generated by the network, users respond by distorting [4] existing network features to obtain desired data.

For example, Traceroute uses the TTL field in an admittedly clever but distorted way, and the Path MTU discovery mechanism is similar. Tools like Sting [14] use TCP in even more baroque fashion to yield end-to-end measures. Even tools that make more conventional use of network features to populate traffic matrices (e.g, [8], [18]) bridge the semantic gap by correlating vast amounts of spatially separated data and possibly inconsistent configuration information. All this is clever, but is it engineering?[1] Could we cut the Gordian knot of measurement complexity by providing features directly in the network?

The problem is complicated by the attitudes of the various stakeholders [4] in the measurement enterprise. *Router vendors* traditionally avoid adding measurement features on the grounds that it will impact forwarding performance. *ISPs* appear resigned to the status quo and the use of indirect means. *Protocol designers*, while bemoaning the difficulty of deploying new protocol features, would rather design new protocols. Finally, many *network researchers* privately regard passive measurement[2] as a boring exercise that could be solved by slapping down a few counters in the right places.

And yet there are signs that change is possible. Cisco Express Forwarding [3] offers per-prefix counters, a massive step up in utility (and, unfortunately, implementation complexity) from SNMP counters. ISPs are putting pressure on router vendors to add features. Juniper's DCU solution [15] uses routing protocol assistance to reduce administrative complexity in a network where routes change constantly. Finally, researchers are beginning to make imaginative proposals for new network measurement primitives [5], [6], [7].

Despite these winds of change, other than the Juniper DCU solution [15], we claim that no earlier solution represents a concerted effort to put all the options (i.e., new router implementation features, protocol changes, new tools) together to orchestrate effective systems solutions to user needs. Systems solutions are those that exploit the fact that a system consists of a number of components, all of which can be modified to help users.

Our central thesis is that *sytems solutions can be used to bridge the semantic gap inherent in network data today with reasonable implementation cost.* Further, such solutions are deployable with some achievable cooperation among the stakeholders. Failure to address the semantic gap will only cause the situation to grow steadily worse with a continued proliferation of ad hoc tools and a lack of coherent data that has hidden costs in terms of reduced productivity.

To provide an analogy of a systems measurement solution that helps users, consider the problem of a shopper wishing to categorize her monthly expenses into food, clothing, dining, etc. Assuming all her shopping is done on her credit card, she can tediously work through her monthly statement. On the other

[1] Recall the comment by hardened battle veterans on the heroic Charge of the Light Brigade: "It is beautiful, but is it war?"

[2] Active measurement is another matter; finding clever ways to manipulate an unsuspecting network to yield its secrets [12], [14] continues to fascinate researchers

hand, if credit card companies had codes for standard spending categories and merchants (or users) could enter the code when a credit card is swiped at a shop counter, then software could be modified to print a summary of spending by category in each statement.[3] This is a systems solution because it requires interaction between components such as users, merchants, and credit card companies.

The rest of this paper is organized as follows. Section II describes three principles that we propose using for guidance in the search for orchestrated solutions to measurement problems. Section III reviews existing primitives and proposals in the light of these principles. Section IV describes a preliminary skirmish, in which the principles are used to suggest a new solution to measuring route stability. Section V describes a more detailed attack on the problem of measuring traffic matrices via a solution that generalizes existing approaches ranging from tomography to per-prefix counters. Section VI summarizes the final messages of this paper.

II. Three Systems Principles

The RISC revolution [13] in architecture can be partly attributed to the following observations. First, architects learned from circuit designers that decoding complex instructions required large clock cycles. Second, architects found that many complex instructions were not used in user benchmarks. Third, architects learned to simulate complex instructions in software, and to move some aspects (e.g., pipeline scheduling) traditionally done in hardware to the compiler. These lessons can be abstracted into three straightforward principles:

Principle P1, Understand real implementation costs: Understand costs and hence the space of feasible implementations.[4]

Principle P2, Understand real user needs: Determine those aspects of current solutions that do not match with user needs, and can thus be potentially simplified. Modify measurement primitives to reduce the semantic gap between user needs and network data.

Principle P3, Leverage other aspects of the system: Recognize that a system consists of multiple components that can cooperate to form effective solutions.

While these principles appear trite, we will attempt to show that they have some teeth in the measurement context by examining existing proposals using the principles as a yardstick in Section III, and then using the principles to suggest new solutions to two *specific* problems in Section IV and Section V. Our descent from a 30,000 feet view of the measurement world down to ground level in Section IV and Section V will be somewhat rapid, and may disconcert the reader. Unfortunately, since measurement is ultimately about measuring specific things, it is difficult to appreciate the challenges or the principles without considering specific measurement challenges. We will step back for a broader view once again in Section VI.

[3]The American Express and Discover cards do this today by having merchants enter categories (e.g., gas, grocery) but user specified codes may be more flexible.

[4]While the fact that RISC stripped away features to make hardware simpler may have been technologically right twenty years ago, it is equally a mistake to underestimate the potential of modern hardware. Thus, increasingly complex instructions have been creeping back into even classic RISC machines like the MIPS.

III. Existing and Proposed Schemes

We review standard measurement primitives in Section III-A, consider new research proposals in Section III-B, and consider an imaginative (at least to our minds) and orchestrated solution for accounting from Juniper Networks in Section III-C.

A. SNMP and Netflow

The following measurement primitives are standard. While useful, building tools based on them is akin to writing programs in assembly language: low-level, tedious, and error-prone.

SNMP Counters: There are a vast number of SNMP counters that routers implement, but for measuring the traffic mix on a link the most relevant are packet and byte counters. SNMP counters are easy to implement (**P1**), and are useful (**P2**) for managers to determine congested links and for tomography [10]. Unfortunately, in terms of **P1**, SNMP does not go far enough; today's hardware can easily support more discriminating counters. SNMP [15] also does not support the need for operators to efficiently obtain large amounts of data (**P2**) because it has high header overhead and effectively uses a window size of 1. Also, data is often lost because of the use of UDP [18].

NetFlow: NetFlow [11] allows managers to log flow records keyed on TCP/IP header fields of all packets on a link. Because of the need to write these headers to slow DRAM, earlier implementations could slow down routers considerably. NetFlow's implementation problems are partly addressed by Sampled NetFlow[1] and aggregated NetFlow. Both solutions represent an interplay between principles **P1** and **P2** by recognizing first that users can get good statistics about traffic mixes from samples, and second that users often only want the sum of traffic for a given 5-tuple. Many implementations of Sampled NetFlow are still problematic, and the vast amounts of data generated can swamp managers and tools. Despite this, network operators generally like NetFlow for its role in diagnosis. Thus while other solutions can meet user needs more directly, NetFlow still seems indispensable.

The approach of [8] correlates NetFlow data at various routers to knowledge of routes. This is not only prone to errors and incomplete data but also does not work with routers than do not support sampled NetFlow well. We can infer this by the fact that AT&T production network uses the tomogravity approach in [18] and not the correlating demand approach in [8], except as a check.

B. Research Proposals

The following research proposals in the last three years have attempted to raise the level of abstraction of network data:

Trajectory Sampling: [5] proposes using a common hash function to synchronize sampling of a given packet across all routers, and a second common hash function as a digest of the invariant packet content. The idea interplays **P1** (easy to implement at high speeds) and **P2** (meets operator need to trace packet routes to determine anomalous behavior such as packet looping).

Directly Finding Heavy-hitters: [7] attempts to finesse the need for NetFlow collection at a router by providing an algorithm to directly compute (at high speeds) the flows over a threshold. It leverages the fact (**P2**) that many existing tools that display NetFlow data (e.g., FlowScan) display only the heavy-hitters, and that knowing the heavy-hitters often suffices for traffic engineering.

Minimizing the export of NetFlow data: [6] also leverages off the fact that users only need heavy-hitters. They propose a sampling technique that can sieve the amount of NetFlow data sent to a manager while preserving any estimates of high flows, for any definition of a flow.

However, trajectory sampling still requires great complexity in a tool to gather and correlate labels from all routers. On the other hand, [7] and [6] only provide local views of heavy-hitters on one link, and do not provide the network-wide view that ISPs need.

C. Juniper's Destination Class Accounting

Juniper Network's Destination Class Accounting solution (DCU) is the only existing measurement solution we know of which effectively combines all three principles. The problem being addressed is that of an ISP wishing to collect traffic statistics on traffic sent by a customer in order to charge the customer differently depending on the type of traffic and the destination of the traffic. We will use Figure 1 — that depicts a small ISP Z — for the next three examples.

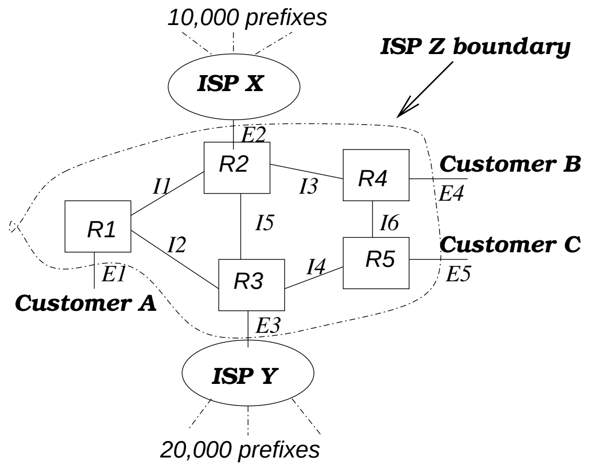

Fig. 1. Example of an ISP with customer and peer links to other ISPs X and Y

In the figure, assume that ISP Z wishes to bill Customer A at one rate for all traffic that exits via ISP X, and at a different rate for all traffic that exits via ISP Y. One way to do this would be for router $R1$ to keep a separate counter for each prefix that represents traffic sent to that prefix. In the figure, $R1$ would have to keep at least 30,000 prefix counters. Not only does this make implementation more complex (Section V-A has a detailed discussion) but it is also unaligned with the user's need which will eventually aggregate the 30,000 prefixes into 2 tariff classes. Further, if routes change rapidly the prefixes advertised by each ISP may change rapidly, requiring constant update of this mapping by the tool.

Instead, the Juniper DCU solution [15] has two components:

1, Class counters: Each forwarding table entry has a 16 bit class ID. Each bit in the class ID represents one of 16 classes. Thus if a packet matches prefix P with associated class ID C, if C has bits set in bits 3, 6, and 9, the counters corresponding to all 3 set bits are incremented. Thus there are only 16 classes supported but a single packet can cause multiple class counters to be incremented. The solution aligns with Principle **P1** because 16 counters per link is not much harder than one counter, and incrementing in parallel is easily feasible if the 16 counters are maintained on-chip in a forwarding ASIC. The solution also aligns with Principle **P2** because it cheaply supports the use of up to 16 destination-sensitive[5] counters.

2, Routing Support: To attack the problem of changing prefix routes (which would result in the tool having to constantly map each prefix into a different class), the DCU solution enlists the help of the routing protocol (**P3**). The idea is that all prefixes advertised by ISP X are given a color (which can be controlled using a simple route policy filter), and prefixes advertised by ISP Y are given a different color. Thus when a router scuh as $R1$ gets a route advertisement for prefix P with color c, it automatically assigns prefix P to class c. This small change in the routing protocol (**P2**) greatly reduces the work of the tool.

Juniper also has other schemes [15] including counters based on packet classifiers, and counters based on MPLS tunnels. These are slightly more flexible than DCU accounting because they can take into account the source address of a packet in determining its class. But these other schemes do not have the administrative scalability of DCU accounting because they lack routing support.

IV. MEASURING ROUTE STABILITY

Having left behind existing solutions, we now turn to our first new example of using the principles. We address the specific problem of measuring route stability. For example, in Figure 1 imagine that the route from Customer A to Customer B usually uses the direct path through links $I1$ and $I3$. However, if link $I3$ is flaky, assume that the link weights are such that the backup path through $I1$, $I5$, $I4$ and $I6$ (Figure 2) will be used instead. If link $I3$ keeps coming up and down, traffic to Customer B (say an important web site) may be affected because of routing instability.

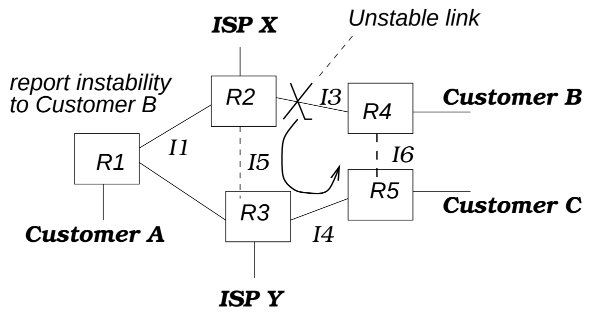

Fig. 2. Using the Dijkstra Tree to compute Route Stability Information in the ISP of Figure 1

This phenomenon could be measured indirectly by observing link failures via say SNMP counts and then correlating with the known physical topology, the link weights, and the protocol to determine that say the route to Customer B will be affected by

[5]It can also be made sensitive to the type of service by also using the DiffServ byte to determine the class.

the flaky link. However, there is a simpler and more direct way to measure route instability with help from the routing protocol or its implementation.

The idea is illustrated in the case of OSPF (by far the most common IGP used by ISPs) by the Dijkstra tree rooted at router $R1$ in Figure 2 for the same network as Figure 1. When new link state packets arrive at $R1$ from $R2$ announcing the failed link, the tree moves around to include link $I5$, $I4$, and $I6$. Normally, $R1$ is blissfully unaware of the route instability because $R1$'s own next hop for Customer B stays the same when the link bounces. However, a trivial, incrementally deployable, change to Dijsktra's algorithm can notice that a node in the tree (e.g., $R4$) has changed parents (from $R2$ to $R5$). All children of a node whose parent has changed are marked as having changed, by passing an "instability flag" down the tree.

The route processor can now maintain summary route stability statistics — such as the number of changes in the last day — on a per-prefix basis. If this is considered too expensive, a management station attached to router can compute this information if it receives all link state packets. Similar ideas can be applied to path vector protocols like BGP by passing an instability attribute with a route.

One way for a management station to receive all link state packets, is to trick an adjacent router into believing it is a router. In that case, no changes are needed to the OSPF computation at regular routers. However, the route stability measurements could be more useful if routers include summary link statistics (e.g., traffic utilization, link errors, etc.) in link state packets. Instead of using these statistics for dynamic routing, which has been considered difficult to do without endangering network stability, the management station could this information to potentially correlate routing problems to causes.

The final claim is that with some changes in the routing protocol, a simple tool can be built that can keep detailed statistics of route instabilities and even correlate them with link parameters. The solution is based on the need for a network-wide view of route changes with some explanatory power (**P2**), the fact that keeping track of route changes will not slow down route computation by much (**P1**), and the the possibility of modifying routing to add more explanatory link attributes (**P3**). This solution has advantages over a tool that polls each router because network changes can result in inconsistency when pasting together a number of unsynchronized local views. [6].

V. Measuring the Traffic Matrix

If even half the attention to 'rocket science traffic modeling' were devoted to how to estimate a reasonable ingress-egress traffic matrix, network engineers, particularly of large clouds, would find their job substantially easier

— Dennis Ferguson, Juniper Founder, 1996 ISMA

For our second problem, consider a network (e.g, Z in Figure 1) such as those used by ISPs like Sprint and AT&T. The network can be modeled as a graph with links connecting router nodes. Some of the links from a router in ISP Z go to routers belonging to other ISPs ($E2$, $E3$) or customers ($E1$, $E4$, $E5$). Let us call such links external links. Although we have lumped them together in Figure 1, external links directed towards the ISP router are called input links, external links directed away from an ISP router are called output links.

The traffic matrix of a network enumerates amount of traffic that was sent (in some arbitrary period, say a day) between *every* pair of input and output links of the network. For example, the traffic matrix could tell managers of ISP Z in Figure 1 that 60 Mbits of traffic entered during the day from Customer A of which 20 Mbits exited on the peering link $E2$ to ISP X, and 40 Mbps left on link $E5$ to Customer B.

Network operators find traffic matrices (over various time scales ranging from hours to months) indispensable. They can be used to make more optimal routing decisions (working around suboptimal routing by changing OSPF weights or setting up MPLS tunnels), for knowing when to set up circuit switched paths (avoiding hot spots), for network diagnosis (understanding causes of congestion), and for provisioning (knowing which links to upgrade on a longer time scale of months).

Unfortunately, existing legacy routers only provide a single aggregate counter (the SNMP link byte counter) of all traffic traversing a link, which aggregates traffic sent between all pairs of input and output links that traverse the link. Inferring the traffic matrix from such data is problematic because there are $O(V^2)$ possible traffic pairs in the matrix (where V is the number of external links) and many sparse networks may only have say $O(V)$ links (and hence $O(V)$ counters). Even after knowing how traffic is routed, one has $O(V)$ equations for $O(V^2)$ variables, which makes deterministic inference (of all traffic pairs) impossible. This dilemma has led to two very different solution approaches.

Approach 1, Internet Tomography: This approach (see [10], [18] for good reviews of past work) recognizes the impossibility of deterministic inference from SNMP counters cited above, and instead attempts statistical inference with some probability of error. At the heart of the inference technique is some model of the underlying traffic distribution (e.g., Gaussian, gravity model) and some statistical (e.g., maximum likelihood) or optimization technique (e.g., quadratic programming [18][7]).

Early approaches based on Gaussian distributions did very poorly [10], but a new approach based on gravity models does much better, at least on the AT&T backbone. The great advantage of tomography is that it works without retrofitting existing routers, and is also clearly cheap to implement in routers. A possible disadvantage of this method is the potential errors in the method, (off by as large as 20% in [18]), its sensitivity to routing errors (a single link failure can throw an estimate off by 50%), and its sensitivity to topology.

Approach 2, Per-prefix counters: Designers of modern routers have considered other systems solutions to the traffic matrix problem based on changes to router implementations and (sometimes) changes to routing protocols (see DCU scheme described earlier). For example, one solution that is being designed into some routers built at Cisco [3] and some startups

[6]While even LSP databases can be inconsistent across routers during periods of instability, LSPs propagate much faster than typical poll intervals for a tool

[7]Some authors limit the term tomography to the use of statistical models; thus the authors of [18] refer to their work as "tomogravity". But this is perhaps splitting hairs.

is to use per-prefix counters. Recall that prefixes are used to aggregate route entries for many millions of Internet addresses into say 100,000 to 150,000 prefixes at the present time.

A router has a forwarding engine for each input line card which contains a copy of the forwarding prefix table. Supposing each prefix P has an associated counter which is incremented (by the number of bytes) for each packet entering the line card that matches P. Then by pooling together the per-prefix counters kept at the routers corresponding to each input link, a tool can reconstruct the traffic matrix. To do so, the tool must associate prefix routes with the corresponding output links using its knowledge of routes computed by a protocol such as OSPF. In Figure 1, if $R1$ keeps per-prefix counters on traffic entering from link $E1$, it can sum the 10,000 counters corresponding to prefixes advertised by ISP X to find the traffic between Customer A and ISP X.

One advantage of this scheme is that it provides perfect traffic matrices. A second advantage is that it can be used for differential traffic charging based on destination address as in the DCU proposal. The two disadvantages are the implementation complexity of maintaining per-prefix counters (and the lack thereof in legacy routers), and the large amount of data that needs to be collected and synthesized from each router to form traffic matrices.

Instead, we propose a scheme that dials between these two earlier approaches using per-class counters that aggregate multiple prefix counters into per class counters. Our scheme is related to but different from the DCU proposal [15].

To lay a foundation for our scheme, Section V-A first introduces a model for the implementation complexity (**P1**) of a large number of counters. Section V-B uses the principles to derive the general scheme, and Section V-C describes five interesting special cases, three of which yield new schemes. Section V-D proposes using a small counter space and yet eliminating tomography completely using sampling on the class counter space.

A. Implementation Complexity for Maintaining Counters

Routers being designed today have to work for 5-10 years. Factoring into account growth in the prefix table, such a router needs to support at least 500,000 and perhaps 1 million prefixes. To avoid frequent wrap around, byte counters are generally 64 bits. To keep up with wire speeds at say 10 to 40 Gbps, counters must be in fast (1- 5nsec cycle time) SRAM. 64 Mbits of fast SRAM is expensive, and has other costs in terms of board (or chip) area and power.

Besides the need for large SRAM, there is the complexity of doing a read-modify-write cycle to this memory from the forwarding chip to read, increment, and write back the counter. Shah et al. [16] show how to reduce the width of the SRAM by storing only the low order bits of each counter in SRAM but storing all 64 bits in a cheaper DRAM backing store. However, the implementation is still complex and requires sorting.

On the other hand, implementing a single counter (or a small number, say 1000, of counters) is trivial. The counters can be stored on-chip and can easily be implemented in parallel with some other parts of the forwarding function to carry almost no time penalty. Juniper's DCU hardware support is an existence proof with 16 counters.

A hidden cost when implementing a small number of counters is the cost of mapping a packet to a counter. One simple way to do this, which we advocate, is to add to each next hop table entry a *class ID*. For example with 1000 counters, a 10 bit class ID has to be added to each forwarding entry. For say a million prefixes, this is 10 Mbits, no small part of the overall SRAM used. However, many next-hop entries are large (20 bytes in some implementations to store multiple adjacencies for load balancing), and some Juniper routers already add a 16-bit class ID. Better still is to finesse the need for a class ID by mapping to a class based on information (e.g., output port, see local matrix proposal in Section V-C) already present in the forwarding entry.

B. General Solution using Class Counters

We apply the principles of Section II to the traffic matrix problem.

P1, Understand costs: Section V-A shows that 100-1000 counters are almost as simple as 1 counter except for the potential storage per prefix for class mapping, and even that can be eliminated if the mapping is a trivial function of existing next-hop information.

P2, Understand real needs: The major applications of prefix counters seem to be in accounting (Section III-C) and populating traffic matrices. In both cases, the solution seems unaligned with real needs. For example, prefix counters seem to be overkill for the traffic matrix problem because there may be millions of prefixes (120,000 today) but an ISP may have far fewer external links.

The RocketFuel [17] data seem to indicate that the large ISPs had between 6000 and 12,000 external links in 2002. Similarly, an accounting application may have only have hundreds of different providers and much fewer tariff structures (Juniper, observing that cell phones offer at most 8 tariffs, allows 16). Thus the final tool (for traffic matrix calculation or accounting) would aggregate thousands of prefixes into equivalence classes anyway. Why not have the router do this in the first place, reducing complexity for *both* the router and the management tool?

P3, Leverage other system components: Mapping from prefixes to equivalence class is not just a issue because of the forwarding implementation, but also because of the problem of refreshing these mappings when prefixes change. If a prefix advertised by ISP X begins to be advertised by ISP Y in Figure 1, who should tell each router of the new mapping? Rather than have the tool or managers do this, it is far more "administratively scalable" [15] to enlist help from the routing protocol as in the DCU proposal.

At this point, the proposed solution is obvious and has two aspects:

1. Use Class Counters: Each prefix is mapped to a small class ID of 8-14 bits (256 to 16,384 classes) using the forwarding table. When an input packet is matched to a prefix P, the forwarding entry for P maps the packet to a class counter that is incremented. For up to 10,000 counters, the class counters can easily be stored in on-chip SRAM on the forwarding ASIC, allowing the increment to internally occur in parallel with other functions.

2. Enlist Routing Support: For accounting, the DCU proposal (Section III-C) already suggests that routers use policy filters to color routes by tariff classes, and pass the colors using the routing protocol. These colors can then be used to automatically set class IDs at each router. For the traffic matrix, a similar idea can be used to colorize routes based on the matrix equivalence class (e.g., all prefixes arising from same external link or network in one class).

Some of the differences between Junipers DCU proposal [15] and ours are:

• *Scalability:* Our scheme is more scalable because N classes require at most $\log_2 N$ class ID bits per prefix, and at most one increment. Incrementing multiple counters per packet, as in the DCU proposal, requires an N bit class ID, and a more complex implementation that may not scale to large N. Thus it would be feasible for us to scale to 10,000 classes, but this appears to be harder for the DCU proposal.

• *ii) Different Application:* Given 10,000 external links, the traffic matrix application motivates the need for a larger number of classes.

• *iii) Different routing support:* We suggest coloring routes based on the type of traffic matrix (see Section V-C), not the tariff class.

Subtleties: This is not a complete proposal; there are tricky issues (e.g., OSPF load balancing) that need to be addressed, and can be. Two issues worth noting are as follows. First, some applications may wish to select traffic (or accounting) classes based on QoS. This can be easily done using the DiffServ bits (in addition to the class ID) to select a class. Second, many papers [8] point out that the real traffic matrix is a point-to-multipoint demand because traffic from say Customer 1 may have multiple egress points to the same Provider. This is easily handled by aggregating all these egress points into the same class.

Notice that our proposal scales better than the DCU proposal by allowing only one counter to be incremented per packet. But the accounting department of an ISP may want accounting data, and the operations department may want traffic matrix data. A small generalization would allow a prefix to be mapped to a superclass ID which can then be parsed to yield a small number of class IDs. For example, a 16 bit superclass ID could be partitioned into an 8 bit class ID for accounting, and an 8 bit class ID for traffic matrix calculation. A received packet increments both. Note that we have separated the degree of parallelism from the number of counters; these two are confounded in the DCU proposal.

Finally, class counters can be specified abstractly *without reference* to traffic matrices and accounting, by only requiring that edge routers have the ability to map prefixes to superclasses, and that routers increment a counter, per class, for every class in a superclass. Generalizing this way can be considered "modularizing along a tussle boundary" [4] as in the DiffServ model.

C. Five Special Cases of Class Matrices

The mechanism above provides the traffic counts from each input link to each destination class. By aggregating input links (at either the router or the tool) also into classes, this method yields the class to class traffic matrix. Let N be the number of classes. We now consider five special cases (Figure 3 shows three cases in a different order from the text below) of class matrices. The first two are well known.

1, One class: If $N = 1$, we have a single SNMP counter, which requires tomography to find the traffic matrix.

2, Prefixes: If N is the number of prefixes we get per-prefix counters — see bottom of Figure 3. Note that the class mechanism is a strict generalization of the two earlier solutions (SNMP and per-prefix counters) which represent two extremes. The next three are more interesting.

3, External Links: We have already referred to this idea, shown in the middle of Figure 3. This provides the traffic to and from external links. Links could be aggregated by provider to handle point-to-multipoint data. RocketFuel data [17] indicates that an $N = 12,000$ sufficed in 2002 for the largest ISP, but this number is probably growing, especially with ISPs consolidation and mergers.

4, PoPs: A PoP is a physical location where an ISP houses a collection of routers. Many ISPs find the PoP-to-PoP traffic matrix to be very valuable [2] and aggregate the router-to-router matrix to find this. This can be done directly by classes by setting each PoP into a separate class. For example, in Figure 1, $R4$ and $R5$ may be part of the same PoP, and thus $E4$ and $E5$ would be mapped to the same class. RocketFuel data [17] indicates a great reduction in the number of PoPs, with $N = 150$ sufficing for the largest ISP.

5, Router Ports: To dial down the number of classes still further, consider making every output port in a router a class (top of Figure 1). This provides the *local traffic matrix* at a router — i.e., the traffic between every input port and output port on each router. For example, in Figure 1, there would be a separate counter at $R1$ for the traffic from links $E1$ to $I1$, and another for traffic from $E1$ to $I2$. Two implementation advantages that accrue are: first, from packets to classes is now trivial and there is no need for a class ID in each forwarding entry (output port is effectively the class ID); second, many routers have a small number of ports from 16 to at most 256 (Juniper T-series) and the reduced class count implies a smaller number of counters.

The conjunction of local matrices at all routers *does not* provide the global traffic matrix. For example, in Figure 1, suppose the local matrix at router $R1$ says that 20% of output traffic on link $E1$ comes Customer A, and that 80% come from another input port (say Customer D, not shown). Suppose the local traffic matrix at $R2$ says says that 20% of the output traffic on Port 22 on link $I3$ comes from input link $I1$. There is still no way to tell whether the 20% of the traffic on $I3$ comes entirely from Customer A or none of it comes from Customer A (or anything in between).

However, local matrices provide more equations that help constrain tomography schemes even more, and thus seems likely to produce more accurate solutions. Although one may expect a factor of P^2 gain in the number of equations where P is the number of ports (e.g., this is a factor of 256 for a 16 port router), the equations are not independent. In practice, some early experiments [9] show a factor of 2 increase in the number of equations.

Despite only a two-fold gain in the number of *equations*, for inference on the AT&T network, the accuracy improvement of

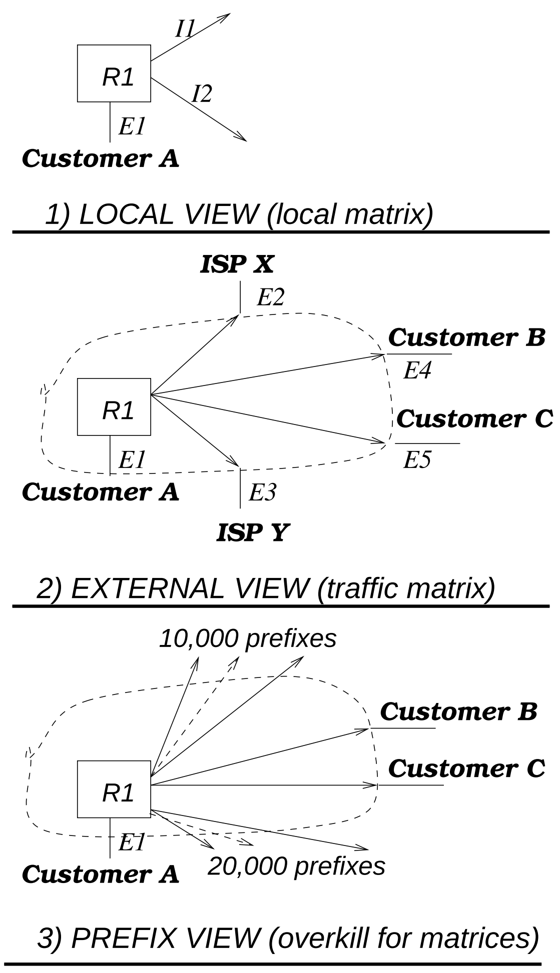
I1
I2
R1
E1
Customer A
1) LOCAL VIEW (local matrix)
ISP X
E2
Customer B
E4
Customer C
E5
E3
ISP Y
2) EXTERNAL VIEW (traffic matrix)
10,000 prefixes
20,000 prefixes
3) PREFIX VIEW (overkill for matrices)

Fig. 3. Using the class counter to produce various views that range from the standard per-prefix view of traffic (bottom) to a local traffic matrix (top).

knowing the local matrix everywhere is roughly the same as knowing the top 15 complete rows (e.g. by turning on NetFlow at 15 busiest backbone routers). With no noise, simulations show that the best-known tomographic inference scheme [18] has an error of 11%; the error reduces to 3% using the extra information in local matrices. With a noise level of 5% in the SNMP link data, simulations show a relative error of 14% without local matrices, and only 5% with local matrices.

For a noise level of 10%, the relative error is 18% without local matrices and 7% with local matrices. This is perhaps surprising because the final error is less than the amount of input noise assumed! While this was on the AT&T network, local matrices should help tomography even more on a sparse network; if the topology is a star, note that the local matrix gives the complete traffic matrix! Overall, local matrices increase the accuracy of the best known tomography schemes ([10], [18]) by a factor of 2.5 to 4.

It is worth asking whether local matrices (which can be considered to be the one-hop local view) can be pushed further to instead compute and use the two-hop (or k-hop) local matrices. It appears at present that the marginal gain in terms of independent equations generated by $k > 1$ hop matrices is small, and thus the knee of the tradeoff curve seems to be at $k = 1$.

D. Sampling Classes as an Alternative to Tomography

We have seen that current data [17] indicates that an implementation must use around 12,000 classes for the external link matrix, 150 for the POP matrix, and even as small as 32 for the local matrix. In Section V-A, we claimed that 100-1000 counters was easy to implement. Assuming that on-chip SRAM sizes can scale faster [13] than the growth rate of external links, we believe that even handling the external link matrix is feasible using our proposal, now and in the future.

However, some implementations may wish to have a smaller number of classes (say 32) in order to use on-chip SRAM for other purposes. An interesting question worth answering is whether one can infer larger matrices (e.g., external link matrix) using a number of classes that is strictly smaller than the number of rows of the desired matrix.

One approach, as in the local matrices idea above, is to compute some exact numbers and use them to constrain statistical inference in tomography. But a completely different approach is to realize that the final output of tomography is a set of *statistical* traffic pair estimates based on *deterministic* inputs. Why not consider turning this proposition on its head and calculate *deterministic* traffic-pair estimates based on *statistical inputs*?

In other words, even if we have only 32 class counters and we wish to watch say 12,000 external links in the AT&T network, why not sample (in time) the links being watched? Thus in any subinterval, 32 randomly selected output links are watched faithfully by the class counter. At the next subinterval, another 32 randomly selected output links are watched.

Assume a traffic estimate is required for K subintervals (e.g., for estimating the hourly traffic matrix, one may wish to use 3600 one second subintervals). Suppose traffic from Customer A to ISP Y was watched for k out of the K possible subintervals. Then a simple estimate for the traffic of Customer A to ISP Y can be found by scaling the sum of the traffic over the watched intervals by the factor K/k.

The random sampling can be implemented by the route processor by picking a class counter to replace, reading the value, and updating the affected prefixes. This is fairly time consuming and thus implementation constraints will require subintervals of at least seconds to avoid burdening route processors. While there is the usual tradeoff between the sampling rate and accuracy, the analysis for such class sampling seems different from that of standard sampling (e.g., [5]). This is because the samples are batched with a random interval between batches and not between individual samples.

Such batch sampling could cause problems with bursty distributions. For example, suppose a flow class sends traffic during the busiest hour and nothing else for the other hours. If the subintervals are in units of hours, then there is a strong chance that the sampling will miss this flow's traffic. However, this is not true if the subintervals are in units of seconds. Thus as with tomography there is some dependence on the input traffic model. However, the dependence is quite different and can (hopefully) be abstracted in terms of some simple summary property of the distribution such as the maximum (with high probability) burst length. One conjecture is that class sampling should do well if the sampling subinterval is much smaller than the maximum burst length, but this needs to be studied.

Further, the accuracy of class sampling can be improved using the sample-and-hold technique of [7]. This comes at the cost of only finding the large elements in the traffic matrix, but this appears to suffice [8] for the traffic matrix and accounting applications. The results in [7] indicate that the use of sample-and-hold can make N counters have the accuracy of using roughly N^2 counters and ordinary sampling. This should provide good accuracy for even small values of N. In general, comparing the accuracy and robustness (especially to assumptions about traffic distributions) of class sampling versus tomography seems an interesting open problem.

VI. Conclusions

We first state our *specific* conclusions based on the specific new techniques we examined in this paper.

First, route stability can be measured precisely and even possibly explained by watching routing traffic, changing route computation slightly, and passing attributes that may explain link behavior. While dynamic link attributes such as congestion level have been eschewed by routing architects because their use in route calculation could lead to instability, such attributes may help tools correlate link status and the performance of routes.

Second, traffic matrices can be calculated more directly using class counters and routing support. Even using an easily feasible number of class counters, local traffic matrices can be used to improve the accuracy of tomogravity schemes [18], PoP matrices can possibly be directly calculated, and link to link matrices can be estimated by class sampling techniques. Per-prefix counters [3] are overkill at least for present applications, and will be hard to scale to million prefix tables. It is possible to abstract the class counter mechanism into superclasses that can be partitioned into classes, without specifying in the advance all the applications for which they can be used. Such abstractions preserve IP's hourglass model, as opposed to burdening IP with application semantics.

But the main message of this paper is broader than the two specific examples we used to illustrate the ideas. The *general* conclusions are as follows. First, we believe that there is a large semantic gap between data provided easily by the network and the needs of users like ISPs. This leads to either the need to trick the network into providing data (e.g., [14]), or to collect, correlate, and synthesize vast amounts of spatially separated data (e.g., [8]). Second, we believe this semantic gap can be bridged at reasonable implementation cost by a systems approach.

The systems approach is codified in this paper using three principles. Any such list is necessarily incomplete, but they can perhaps serve as a first cut. The principles essentially seek to close the loop between user needs and implementation constraints, while surveying the range of possible changes that can be done to intervening components such as routing protocols.

Fourth, we believe that besides the two examples we used in this paper (route stability and traffic matrix calculation), there are many other network-wide metrics whose calculation can be reconsidered using these principles. Other examples include link bandwidth, end-to-end error rates, and routing update delays. These three examples were taken from papers on these topics (all of which assumed the network to be a black box) in the 2001 Internet Measurement conference. A more careful survey could probably find a larger number of such problems.

Fifth, we believe that it is possible for the stakeholders to work together. We believe network researchers can be convinced that the field of measurement is not merely a boring study of counters, but can furnish rich and beautiful problems. We believe that ISPs are already discovering a business case for better data collection that can lead to running their networks more profitably. Finally, we believe that router vendors can make changes when pressed.

Amplifying the last point, it may be argued that protocol changes, as advocated by Principle **P3**, are hard to accomplish. However, this flies against the number of protocol changes made in the last few years (for MPLS traffic engineering, supporting VPNs, etc.). In general, it appears that as long as there is a strong business case for the two dominant router companies, protocol and implementation changes can and do happen.

To establish such a business case it would clearly help for ISPs to form a user consortium to specify measurement requirements. After all, router vendors already take the NEBS standards for reliability in telecommunications equipment very seriously. Thus we end, somewhat tongue-in-cheek and with apologies to Karl Marx, with a final slogan:

ISPs of the world unite — you have nothing to lose but the chains that imprison the data you need!

References

[1] Sampled netflow. URL = http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/120newft/ 120limit/120s/120s11/12s_sanf.htm.
[2] S. Bhattacharya et al. Pop-level and Access-link traffic dynamics in a Tier-1 pop. In *SIGCOMM Internet Measurement Workshop*, November 2001.
[3] Cisco express forwarding commands. http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12supdoc/12cmdsum/12csswit/cscef.htm.
[4] D. Clark et al. Tussle in cyberspace: Defining tomorrow's internet. In *Proceedings SIGCOMM 2002*, September 2002.
[5] N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. In *Proceedings ACM SIGCOMM*, August 2000.
[6] N. Duffield, C. Lund, and M. Thorup. Charging from sampled network usage. In *SIGCOMM Internet Measurement Workshop*, November 2001.
[7] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings SIGCOMM 2002*, September 2002.
[8] A. Feldmann et al. Deriving traffic demands for Operational IP networks: Methodology and experience. In *SIGCOMM 2000*.
[9] left blank for anonymity Local matrix results. January 2003.
[10] A. Medina et al. Traffic matrix estimation: Existing techniques and new directions. In *Proceedings SIGCOMM 2002*, September 2002.
[11] Cisco netflow. http://www.cisco.com /warp /public /732 /Tech /netflow.
[12] J. Padhye and S. Floyd. On inferring TCP behavior. In *Proceedings ACM SIGCOMM*, August 2001.
[13] D. Patterson and J. Hennessy. *Computer Organization and Design*, page 619. Morgan Kaufmann, second edition, 1998.
[14] S. Savage. Sting: A TCP-based network measurment tool. In *USENIX Symposium on Intenet Technologies and Systems*, 1999.
[15] C. Semeria and J. Gredler. Juniper networks solutions for network accounting. In *Juniper White Paper, 200010-001*, 2001.
[16] D. Shah et al. Maintaining statistics counters in router line cards. In *IEEE Micro*, Jan 2002.
[17] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies using RocketFuel. In *Proceedings SIGCOMM 2002*, September 2002.
[18] Y. Zhang et al. Fast accurate computation of large-scale IP matrices from link loads. In *ACM SIGMETRICS 2003, to appear*, May 2003.