Design Considerations for Information Planes

Brent Chun, Joseph M. Hellerstein, Ryan Huebsch, Petros Maniatis, and Timothy Roscoe

Intel Research at Berkeley

***Abstract*— The concept of an *information plane* has emerged recently as an important part of large, decentralized systems that aspire to be self-managing, ranging from PlanetLab to the Internet itself. In this paper we describe what an information plane is, and report our experiences in developing and deploying an information plane for the PlanetLab platform using the PIER distributed relational query processor. We recount the lessons we have learned from the experience, and the additional directions we intend to explore in the PHI project, which aims at providing an information plane that can grow to serve a significant portion of the Internet.**

1. Introduction

Recent research into widely-distributed systems has led to the emerging concept of an *Information Plane:* a service or service component that efficiently delivers timely and relevant data about the state of the system to all the dispersed components of the system. For example, an information plane for PlanetLab might be used by end-systems for resource discovery or node monitoring and load balancing. At a different scale, a hypothetical information plane for the Internet would enable end-systems and applications such as overlays to cooperate on routing decisions, worm signature detection, and fault and performance diagnosis of the network as a whole.

An information plane is broadly coextensive with the system it relays information about, and consequently is highly decentralized. In this respect it differs from traditional centralized management solutions: an information plane may have to deliver specific information to all points in the system concurrently. An information plane might still be under single administrative control, but the more interesting cases are where it is realized as a federation of cooperating but mutually suspicious entities, and the extreme case of a fully peer-to-peer model.

In this paper we discuss the specific challenges in building a robust and efficient information plane. Since an information plane necessarily consists of some kind of application-level overlay network, the familiar general issues of reliability, fault-tolerance, performance, scalability, and security exist as in any other large distributed system. Here, however, we concentrate on those challenges specific to the design space of information planes, or areas where we suspect that the information plane scenario may provide opportunities for novel solutions.

Our particular context is the PHI project. PHI aims to build an information plane for the Internet that can grow over time to link a wide variety of network data sources ranging from on-line active and passive measurement systems, routing databases, routing protocol feeds, intrusion detection logs and online feeds, and signature detection systems.

This somewhat ambitious vision comes with formidable requirements in areas such as scalability, semantics, and security. The ultimate (possibly unattainable) goal of our work is to scale to many millions of data sources, many millions of data sinks (including most end-systems), and many millions of concurrent queries.

In the rest of this paper we outline the required functionality of an information plane, give a brief survey of research projects pursuing this goal, and present lessons we have learned from our design, implementation, and deployment over the last year of an information plane on PlanetLab based on the PIER p2p query processor. These lessons include the value of having multi-resolution emulation integrated in the development cycle of an information plane, mechanisms for validation of the functionality, and the lack of a commonly accepted semantics for continuous queries within a dynamic, faulty, heterogeneous ecology of data streams. Finally, we discuss the additional research challenges in this area, with some initial thoughts as to how they might be tackled: security and fidelity, multiquery optimization, exposing failures, and the design of information plane protocols.

2. Function

Having discussed both the applications and the challenges of an information plane, what does an information plane actually do? We use the term *client* to refer to an entity at a particular location in the network which exploits the functionality of the information plane, and *sensor* to denote a specific source of data available to the information plane (such as a monitoring process executing on a particular network node). An information plane accepts requests for specified data, which we will call *queries*, from clients. From a query, the information plane sets up distributed state to create a flow of data through the information plane from appropriate sensors to the client.

There are therefore two kinds of distributed communication involved in the operation of an information plane. One is the data flow itself from sensors to clients. The second is *query signaling*: the communication involved in setting up, maintaining, and ultimately dismantling a dataflow graph within the information plane's overlay. This includes state which is both communication-related (where data should be routed through the network) and computation-related (how is should be aggregated, transformed, and otherwise processed en route).

The sheer volume of information and its distribution imply that the information plane must perform some computation on data rather than simply moving it around the network. This processing can be classified into 3 main types:

Filtering of data items includes the traditional relational algebra operations of projection and selection, but may also include more complex operations.

Aggregation operations on data can be temporal (such as a Fourier transform), spatial (such as computing an average over data from a variety of sensors), or both. They might be very simple (such as maximum or top-) or complex (e.g. data summarization algorithms). Aggregation computations generally consume more data than they generate. For this reason, performing aggregation in the network (rather than at the client) is highly desirable [10] to reduce the total bandwidth required at the client node.

Finally, *correlations* of one dataset against another, as in relational *joins* of data against each other. Since a database join is a selection operation on the Cartesian product of two datasets, correlations have the potential to produce more data that they consume. Nevertheless, performing correlation inside the information plane as a distributed computation is still desirable since it can reduce the total bandwidth required at the client node.

An ideal information plane architecture would provide an extensible dataflow framework into which these operations can be inserted.

To take a very simple concrete example, a client of our PlanetLab-based deployment might contact PIER and submit a query to return the 10 IP source addresses which have triggered the most Snort rules on PlanetLab recently, summed over all the nodes running Snort. PIER sets up computation state in the network to count Snort events from each source, and routing state to forward the results up a tree and thence to the client.

Finally, though we have couched this discussion in traditional database terms, the idea of a relatively static *schema* for an information plane is unworkable due to the size and dynamicity of the system. Consequently, *data integration* at query-time—i.e. when information is used, as opposed to when it is generated—is an additional challenge. This issue is not simply limited to data types; as we discuss below, differences in query semantics (confidence intervals, error distributions, retention policies, etc.) complicate the task greatly.

3. EXISTING WORK

IrisNet [6], Astrolabe [15] and Sophia [16] are early instances of information planes. IrisNet uses a hierarchical data model (XML) and a hierarchical network overlay (DNS) to route queries and data. As a result, it shares the characteristics of traditional hierarchical databases: it is best used in scenarios where the hierarchy changes infrequently, and the queries match the hierarchy. Astrolabe is a robust peer-to-peer administratively-hierarchical query processing system, though its design does not aim to scale to large numbers of concurrently posed queries or large numbers of data attributes, and it requires manual topology management of participating hosts [14]. Sophia evaluates declarative queries expressed in Prolog using distributed unification. Sophia incorporates space and time as first-order components of queries to handle very dynamic changes in the underlying monitored system without sacrificing the benefits of caching. Achieving performance and scalability in Sophia depends on multiquery optimization of Prolog expressions, currently a hard problem.

SWORD [12] is a wide-area resource discovery service that also tries to deal with complex queries over the attributes of rapidly changing data sources. SWORD retrieves subsets of a node population that satisfy constraints on individual node attributes and on attributes among the nodes in the set. A client can also control the trade-off between query processing cost and result recall using resource constraints.

Much recent work focuses on the hard problems that lie beneath an information plane: aggregation, range searches, and query planning and execution. The Scalable Distributed Information Management System (SDIMS) [19] shares many goals with an information plane, and focuses on how a flexible aggregation framework for data can be built using DHTs. SDIMS enables administratively isolated aggregation so that clients can obtain results from within the scope of their local organization or beyond, at a client-specified granularity. The work also explores how replication in time and in space can increase robustness. Along similar lines, Mercury [2] focuses on another area of functionality, multi-attribute range searches, by constructing a separate overlay that splits nodes into *hubs*, each responsible for maintaining a distributed ordered index on a set of attributes.

PIER [7] is a distributed relational query processor built as a P2P system over the Bamboo [13] distributed hash table (though PIER is agnostic with respect to DHT implementation). PIER treats a large, widely distributed set of data sources as a single, loosely-coupled relational

```
SELECT F.entryID, F.sourceIP, SUM(P.bytes)
  FROM firewallLog F, packetTrace P
 WHERE F.sourceIP = P.sourceIP
GROUP BY F.entryID, F.sourceIP
WINDOW F ['1 minute'], P ['1 minute']
```

Fig. 1. A simple continuous query (expressed in TelegraphCQ's query language [9]). For each firewall log entry that appears, it sums the bandwidth sent from the event's source to all nodes in the system over the last minute.

database, and differs from many systems in that it can accept a wide variety of complex query plans. It uses the underlying DHT for constructing trees used by hierarchical operators (such as data aggregation) and multicasting query state, and hashing tuples for indexing, parallelism (such as distributed join algorithms). In addition, the DHT can be used to implement distributed range queries.

We have operated a PIER instance on PlanetLab for the last 12 months, querying data from various PlanetLab status sensors, including the SNORT [8] intrusion detection system, and assorted slice and machine status indicators. In Section 4 we discuss what we have learned from operating a small prototype information plane using PIER.

Last but not least, the Knowledge Plane [3] lays out a broad vision for global self-managing and self-diagnosing networks. An information plane can be viewed as a somewhat more concrete and tightly scoped step in this direction.

4. EXPERIENCE

We have operated an instance of the PIER distributed relational query processor on PlanetLab for the last 12 months. As well as having access to PlanetLab node status information, the system has also been able to query SNORT instances running on each PlanetLab node. This enables us to issue queries such as "what are the top 10 IP source addresses triggering Snort alerts across all PlanetLab nodes?", or "what other nodes within my administrative domain got alerts whose sources match those of my alerts?" PIER can also query its own internal data structures (such as Bamboo's routing table) for diagnostic purposes.

The measurement results from this exercise are beyond the scope of this paper, but here we present the insights we have obtained from the experience of building, deploying, and operating PIER in this scenario, together with their implications for the design of Internet-scale information planes.

Multiresolution emulation: Our experience with deploying PIER has been consistent with the conventional wisdom that debugging is always easier in a controlled environment. A key lesson in debugging PIER has been the benefit of a *multiresolution emulation* approach, where the complexities of a real deployment can be approximated in successively more realistic and challenging stages. As a start, PIER supports a simulation mode that allows its core query processing code — the same "production" code — to run on a message-level discrete-event simulator of the network of machines. This is invaluable for identifying early bugs in the distributed logic of query execution. For example, the simulator helped us realize that some nodes can receive result tuples before they receive the corresponding query request.

However, simulation of this level of fidelity typically catches only a certain class of errors: those triggered by the aspects of the real world that are convincingly modelled by the simulator. In deployment over a real network, early versions of PIER that worked properly in simulation produced clearly inaccurate results even for simple queries. For example, results to simple aggregation queries — e.g., counting the number of nodes in the system — fluctuated significantly from minute to minute even when we knew the population was stable. The use of Emulab [17] as a controlled emulation of the physical network was an important next degree of complexity, while preserving the ability to easily inspect the running system - notably, in discovering bugs in the way that PIER shipped tuples from node to node in a congestion-controlled manner. Control over link-level performance enables a better understanding of the dynamism in the overlay topology, and the resulting effects on query execution.

Finally, the deployment of the system on a real distributed platform like PlanetLab is important: it drives the traffic over real links and endpoints with varying and unpredictable loads, but still allows us to observe and log the execution of each element in the system, albeit at reduced levels of reproducibility.

The need for lineage and history: Distributed queries — even simple ones like that of Figure 1 — can easily translate into rather involved distributed dataflow computations. While this simple query requires a logical dataflow pipeline of just two operators — a join and a grouping+aggregation operator — these are both mapped onto a multi-hop network, with each of the operators partitioned for parallel execution on multiple network nodes. Even for simple queries like this it can be hard to ensure that results are correct, timely, and efficiently computed. When this is not the case, even with multiresolution simulation capabilities available, it is extremely difficult to identify the root cause of the problem. A faulty, unreliable or grossly inefficient information plane would be of limited use at best.

With regards to correctness, our experience with PIER to date suggests that producing a bug-free prototype information plane is non-trivial. In practice, we approach the task in a relatively ad hoc way; for example, we exe-

cute simple continuous queries and check to see if the results remain stable over time. PIER is also instrumented with copious local log output. Although these approaches are useful as debugging tools in the short term, it would be extremely helpful to formalize and enrich this process, especially in a way that can validate the correctness of arbitrary queries.

One promising avenue towards explaining the source of erroneous results is tracing the *lineage* of a result, i.e., the set of input tuples that affect it [18]. Unfortunately, tracking result lineage in centralized data warehouses is hard enough [4], [18]; it becomes significantly more complicated in the real-time distributed setting due to the dynamism of the network over which the data flow: routes may be flapping during query execution, and the set of nodes participating in the computation may be in flux. Ascribing the blame to a component or computation step in the network can be tricky, expensive, and inaccurate.

A possible implementation is to have the system produce information (either in the dataflow or in system logs) about the processing, at each step, of specific tuples through the distributed dataflow. This is analogous to radiology: by introducing a "dye" into the "bloodstream" of the system, we should be able to observe stages of the dataflow using the appropriate diagnostic tools. The "dye" in our case can be specific values in tuples that affect the outputs of operators in known ways, or a special "trace-this" flag in a tuple's header that propagates from input tuples to result tuples during processing. We have already used a similar, though rather more ad-hoc, technique in PIER, by means of tuples that record the network path as part of the answer tuple. Correctness is a critical goal for an information plane, but clearly performance is also important. Identifying the source of slowdowns in the system is somewhat different than identifying correctness bugs. Intuitively, some of the same "radiological" approach may also be applicable. For example, early on we identified in the PIER deployment a problem with head-of-line blocking in the event queue, because the non-preemptible handlers that invoke query execution logic did not yield sufficiently often. The problem was most evident in application level network code which was sensitive to even small delays. This kind of problem would be easier to detect if tuples in the system selectively carried a history of the machines, query operators, and queues they traversed, and the amount of computation, memory, and storage consumed at each component.

A challenge in both these regards is to avoid the Heisenberg phenomenon of affecting the system by measuring it. Adding time-stamping calls to the code-path can affect overall system behavior; adding annotations to the in-flight tuples can change message sizes, potentially causing packet fragmentation where none was otherwise necessary.

Layer interactions: An early PIER prototype repurposes the routing information of the underlying overlay (Bamboo [13]) in order to construct its aggregation graph. Unfortunately, the aggregation machinery and the routing machinery operate under different assumptions and towards different goals. Whereas the overlay aggressively updates its routing information to ensure low-latency delivery of messages, PIER must strive to maintain an aggregation graph for the duration of a query. This incompatibility in assumptions lead our early aggregation attempts for even simple node counting queries to wild inaccuracies, for example right before and during a massive overlay routing update. The painful lesson learned is that, though convenient, mechanism reuse must be prefaced by appropriate adaptation; in this instance, even if PIER takes advantage of the DHT to obtain an initial aggregation graph with good network proximity characteristics, it must store and maintain this graph itself for as long as it requires that graph.

In a similar vein, queuing of outgoing results in an information plane node appears to be critical for performance. PIER attempts to batch tuples before transmission, using a variety of heuristics to decide when to send a message. We are convinced that a feedback-based scheme which eagerly sends tuples up to the available throughput between two nodes would provide much better performance.

Semantics: Perhaps the major lesson from our experience with PIER is the need to define (and provide) clear semantics for queries. Semantics for continuous queries is still an uncertain area in the database literature, only recently receiving attention within the context of wide-area, loosely coupled environments [1]. With a widely-distributed information plane, the problem is compounded by the need to trade-off answer quality, latency, and query cost to the system. How users express flexibly their preferences with regards to this trade-off is an open question.

5. RESEARCH DIRECTIONS

Our current work involves designing, implementing, and deploying a more complete information plane, building on our experience with PIER. We intend to incorporate the lessons we've learned from PIER in our new system; in this section, we discuss additional research directions we intend to explore.

Security and fidelity: In starting again, it is clear that the issue of security which was postponed in the design of PIER has to be fundamental in the new design. Even a single-authority information plane poses a daunting set of security-related challenges, which are multiplied in a federated or peer-to-peer case. Individual information sources may have requirements for access control and/or anonymization.

Furthermore, whether federated or under a central administrative organization, an information plane of any size has to deal with the possibility of malicious or compromised nodes. Malign entities will seek to affect the correctness and throughput of the information plane, as well as to use it as a stepping stone for attacks against other victims on the Internet. While we can take great care not to introduce vulnerabilities into the implementation, we must still assume the possibility that nodes will be compromised sooner or later. Consequently, the information plane must be able to detect misbehaving nodes so that they can be isolated or the results of their computations treated with suspicion. This is a hard problem, but one promising technique in this regard is the use of probabilistic spot-checking [5].

In addition to incorrect results ("data poisoning"), an information plane must also prevent itself being used to conduct denial of service against other network entities, for example by distributed rate limiting.

Multiquery optimization: To scale effectively in the number of simultaneous queries an information plane can handle, similar queries need to share both communication and computational resources. This is known in the database literature as multiquery optimization, and work so far has focused almost entirely on the centralized case, with some recent work on continuous queries [11]. The latter work annotates tuples with bitmaps indicating their progress through the operator graph of the query, and the current set of active queries satisfiable by this tuple.

One promising line of inquiry is to port this approach to the distributed case. This requires a compact representation of the lineage of the tuple, which can be transmitted with the tuple data between nodes in the information plane. The design of an efficient representation presents several challenges, but we note that this would also help with spot-checking for data fidelity, and validating the system.

Exposing failures: In a large P2P system like an information plane, failures can be expected to be common. Indeed, it is reasonable to assume that at least one component of the system will be in a state of failure all the time. While attempting to work around failures and still deliver useful results is a goal, the "right thing to do" in the face of failures is also partly application-specific - for example, applications might wish to know precisely what failed, while others might wish to know consequent error bounds on the results. We feel that an information plane should expose such failures to clients as part of the results; trying to mask failures in any way amounts to imposing undue policy restrictions on clients. We will look at both how failure semantics can be expressed in queries and how failure-related information can be combined with query results.

Information planes as protocols: Finally, and perhaps more importantly, it is clear that an information plane is more than an implementation. As much as anything, it is also a *set of protocols* used by nodes to exchange data and control information. This in turn raises the question of interoperability, and the need to defend against badly behaved, buggy, non-compliant and/or malicious hosts at the protocol level, a question that existing monitoring systems (our own included) have hitherto avoided. An aim of PHI is to define and implement such a protocol suite.

Any information plane protocol suite is like to have at least two parts. The first is a *dataflow signaling protocol* which is used by users and participating nodes to set up and tear down query state in the system. This amounts to a signaling protocol for setting up dataflow within the information plane. Key issues in the design of this protocol include an external representation of partial query plans.

The second is a *tuple transfer protocol* which is used for exchange of data between nodes as part of a dataflow. A major part of this protocol is an external representation for tuples, which must not only include the data in the tuple, but enough tuple lineage information to be useful to multiquery optimization mechanisms and auditing functions.

6. CONCLUSION

Information Planes for large-scale distributed systems, from PlanetLab to the Internet itself, represent an important set of challenges for research, both as driving applications for distributed systems work and as interesting network services in their own right.

In this paper we have attempted to define the function of an information plane, and laid out some of the main design challenges. Our experience operating PIER on PlanetLab for the last year has been invaluable in deriving design principles, but we feel the area contains exciting challenges for future work, some of which we have attempted to outline.

REFERENCES

[1] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The Price of Validity in Dynamic Networks. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Paris, France, June 2004.

[2] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proc. ACM SIGCOMM*, Portland, OR, USA, Sept. 2004.

[3] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski. A Knowledge Plane for the Internet. In *Proc. ACM SIGCOMM*, pages 3–10, Karlsruhe, Germany, 2003.

[4] Y. Cui and J. Widom. Lineage Tracing for General Data Warehouse Transformations. In *Proceedings of 27th International Conference on Very Large Data Bases*, Rome, Italy, Sept. 2001.

[5] F. Ergün, S. Kannan, S. R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. In *Proc. of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 259–268, Dallas, TX, USA, 1998.

[6] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan. IrisNet: An Architecture for a World-Wide Sensor Web. *IEEE Pervasive Computing*, 2(4), October-December 2003.

[7] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Querying the internet with pier. In *Proc. of the 29th International Conference on Very Large Data Bases*, September 2003.

[8] J. Koziol. *Intrusion Detection with Snort*. SAMS, May 2003.

[9] S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Madden, F. Reiss, and M. A. Shah. TelegraphCQ: An Architectural Status Report. *IEEE Data Engineering Bulletin*, 26(1):11–18, 2003.

[10] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *Fifth Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, Dec. 2002.

[11] S. R. Madden, M. A. Shah, and J. M. Hellerstein. Continuously adaptive continuous queries over streams. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2002.

[12] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable Wide-Area Resource Discovery. Technical Report CSD-04-1334, University of California Berkeley, Berkeley, CA, USA, 2004.

[13] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz. Handling Churn in a DHT. In *Proc. of the 2004 USENIX Technical Conference, Boston, MA, USA*, June 2004.

[14] R. van Renesse. The Importance of Aggregation. In A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, *Proc. of Future Directions in Distributed Computing*, volume 2584 of *LNCS*, Heidelberg, Germany, Apr. 2003. Springer.

[15] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[16] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *Proc. of the 2nd Workshop on Hot Topics in Networks*, Cambridge, MA, USA, Nov. 2003.

[17] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proc. USENIX OSDI*, pages 255–270, Boston, MA, USA, Dec. 2002.

[18] A. Woodruff and M. Stonebraker. Supporting Fine-Grained Data Lineage in a Database Visualization Environment. In *Proceedings of the 13th International Conference on Data Engineering*, pages 91–102, Birmingham, UK, Apr. 1997.

[19] P. Yalagandula and M. Dahlin. A Scalable Distributed Information Management System. In *Proc. ACM SIGCOMM*, Portland, OR, USA, Sept. 2004.