An Architecture for Distributed Real-Time Passive Network Measurement [*]

Tilman Wolf, Ramaswamy Ramaswamy, Siddhartha Bunga, and Ning Yang
Department of Electrical and Computer Engineering
University of Massachusetts Amherst, MA 01003
{wolf,rramaswa,sbunga,nyang}@ecs.umass.edu

Abstract

We present an architecture for a Distributed Online Measurement Environment (DOME) which is a passive measurement system that correlates network information between several measurement nodes placed at different locations in the network to offer a large scale view of network operation. The system is capable of capturing packet traces and pre-processing them on the measurement node itself. Real-time queries are implemented by breaking them down into standard statistics that are updated during run-time. We present details of a prototype implementation of our architecture on an Intel IXP2400 network processor. The prototype is deployed on the main Internet access link of the University of Massachusetts and measurement results are validated against those obtained from an Endace DAG card. Performance of the prototype is compared to that of a conventional post processing system for an application to detect network anomalies.

1 Introduction

Measurements are becoming an increasingly important tool for managing and understanding computer networks. The increasing complexity of networks in terms of topology, traffic patterns, routing behavior, and throughput performance is due to a rising number of diverse and heterogeneous end-systems and network equipment. The results from measurement can give an insight into correct and faulty network behavior, provide the basis for traffic and performance models, as well as being the only means of seeing what is happening inside a network.

A number of different measurement techniques have been developed, and in this work we focus on *distributed passive real-time* measurements. *Passive* measurements observe and collect information from some or all traffic on certain links or nodes. The observed traffic is what is actually transmitted by network users and no synthetic traffic is injected into the network. The key challenge in passive measurement is the enormous amount of data that can be generated by measurement nodes collecting traces from high-speed data links. Traditionally, such traces have been stored in large databases and post-processed to extract relevant measurement metrics.

In many cases, measurements are performed only to extract a few performance metrics (e.g., average link utilization, heavy-hitter flows, or presence of TCP SYN floods) and long-term storage of measurement data is of only minor importance (or not necessary at all). *Real-time* measurement (or *online* measurement) addresses this goal by processing data traces on the measurement node itself before (or instead of) storing traces in a database. This requires that a measurement node have enough processing power to handle user queries on the node itself.

The *distributed* aspect of measurement aims at correlating packet instances, network metrics, and events across multiple measurement nodes placed at different locations in a network. Such a system expands the view of the measurement beyond a single node or link and helps understand large-scale network behavior and performance. The goal of our measurement architecture is to enable measurement capabilities and interchange of measurement results between a diverse and heterogeneous set of measurement devices by defining a common DOME ("Distributed Online Measurement Environment") architecture.

The design and implementation of such a measurement system poses a number of interesting challenges in terms of functionality, scalability, interoperability, and performance. In our work, we develop a suitable measurement system architecture and present the results of a network-processor based prototype implementation. In particular, our contributions are:

- A **node architecture** for collecting real-time passive measurements
- A **query interface** for specifying measurement tasks

[*]This research was supported in part by National Science Foundation grant CNS-0325868 and through a gift by the Intel Corporation.

- A **representation of statistics** that allow the reuse of measurement data across queries
- A **prototype implementation** based on the Intel IXP2400 network processor [12] to demonstrate the feasibility of this architecture.

Network processors have continued to steadily evolve into highly integrated components which, more recently, are being used as the basic building blocks of a large number of packet processing devices. While our prototype system is based on a network processor, it is important to note that the proposed system architecture can be implemented on any system with sufficient I/O bandwidth and processing power. Depending on hardware choices, there is a trade-off between the line rate at which packets can be measured and processing power. Network processors are ideal for such tasks since they are optimized for packet processing tasks and exploit the parallelism found in packet processing. Compared to general purpose hardware, network processors offer increased performance and better scalability as line rates increase.

One can expect that more and more devices will support some sort of measurement functionality (e.g., NICs, low-end routers, end-systems). In order to harness the power of these systems into a single, coherent network measurement infrastructure, it is important to define a common architecture, query interface, and ways by which measurement data can be reused by a single node, or exchanged between multiple nodes. This is the topic of this paper.

Section 2 briefly discusses related work. In Section 3, we introduce the system architecture of the measurement node. Section 4 discusses queries in more detail. The prototype implementation and results are presented in Section 5, and Section 6 summarizes and concludes this paper.

2 Related Work

Network measurements can be performed using active and passive measurement techniques [5]. In active measurement, synthetic traffic is injected into the network and the sender and/or receiver collect performance statistics on the traffic they generate and/or receive (e.g., NLANR's Active Measurement Project [17] and Surveyor [25]). The obtained results yield information on end-to-end performance characteristics [20, 21, 27]. In passive measurement, performance statistics are derived from locally observed traffic traces.

Several passive measurement projects aim at large scale monitoring of the backbone (e.g., NLANR PMA (Passive Measurement and Analysis) [18], Sprint IPMON [9], and AT&T GigaScope [7]). These projects utilize custom hardware (ASICs, FPGAs) to obtain the performance required for monitoring high speed links. Other measurement projects use off the shelf components for monitoring edge and access links (e.g., NProbe [16]). SMARTxAC [1] is a passive measurement system with real time analysis capabilities which works on packet headers provided by a DAG 4.3E card [8]. SCAMPI [6] is a joint European effort to develop a scalable monitoring platform. None of these projects, however, leverage the use of network processors for processing measurement results on the measurement node itself. Instead, it is common to store the collected network traces in large databases. Measurement results are obtained by searching through the databases and post-processing the packet traces.

A distributed passive measurement infrastructure is presented in [3], which is generic with respect to the underlying hardware used. The capture node in this system performs very little post processing of the data and lacks a real-time query capability similar to what we propose. Instead, querying is implemented by post processing traces in real-time and streaming packet data through a measurement area network to several software "consumers." Additionally, there is no centralized trace collection database in this infrastructure.

ATMEN [14] is another distributed measurement infrastructure which uses Gigascopes[7] as its packet capture device. It also has a real-time query capability, and the ability to reuse measurement data, all of which are features of our DOME architecture. However, ATMEN is distributed in nature with its components (applications and data sources) communicating via the Internet using a communication protocol. Our DOME architecture was designed to be implemented on a single measurement system with several such systems communicating with each other. The CoMo Project [11] has developed a measurement architecture that can calculate generic metrics on the traffic stream in the form of queries.

Several software based network monitoring and analysis tools exist. These tools utilize packet capture libraries such as *libpcap* to monitor network traffic and extract various metrics of interest. Examples of such tools include SNORT [24] for intrusion detection and Netsniff [13] for traffic capture and analysis. The capabilities of these tools can be extended easily since they are software based. However, their performance is limited by the characteristics of the underlying hardware on which they are executed.

3 Capture Node Architecture

Figure 1 shows the overall system architecture. The primary components of the system are a set of DOME architecture based capture nodes (shown as "measurement enabled DOME end-system/router" in the figure). These nodes can be deployed in two ways. They can be added as a standalone passive device to those links that need to be monitored, or

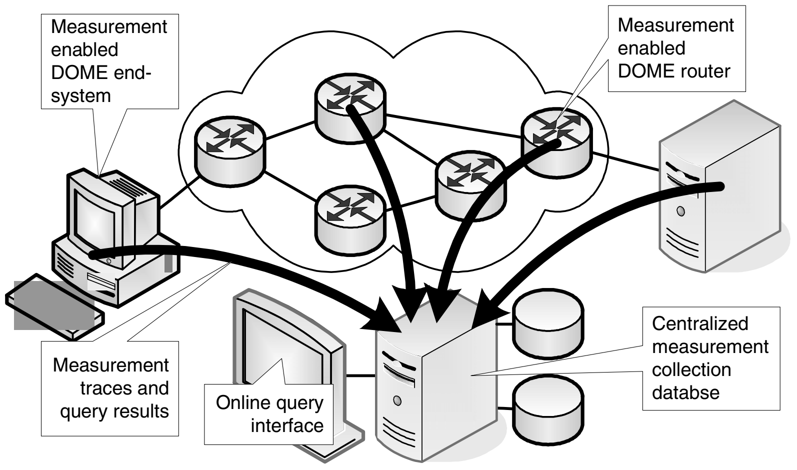

Measurement enabled DOME end-system
Measurement enabled DOME router
Centralized measurement collection databse
Measurement traces and query results
Online query interface

Figure 1. System View of Measurement Nodes and Interchange of Measurement Results.

they can be integrated into the functionality of a router. The capture nodes perform the collection, archiving, and measurement of packet traces. Each capture node is equipped with a query interface (discussed in Section 4) which is accessible via a local end-system based on the DOME architecture. All capture nodes transfer measurement traces and query results to a centralized trace collection database. A query can be issued to the entire measurement infrastructure from the centralized database. Additionally, the capture nodes can also exchange query information between themselves in order to provide better results to queries that require a distributed view of the network.

3.1 Design Challenges

The capture node needs to process incoming packets at line speed. Since the PCI bus on a commodity workstation can easily become a bottleneck at Gigabit data rates, it is not feasible to simply employ a high-performance server system for this task. Additionally, a single processor presents a bottleneck in terms of scalability as measurement tasks and online queries become more complex.

We propose to employ network processors for implementing the capture node. Network Processors (NPs) are ideal candidates for this domain as they are single-chip multi-processors, specifically designed for simple, high-bandwidth processing tasks. Further, network processors have sufficient I/O and processing capacities to scale to link speeds of 10Gbps. Commercial examples of such network processors are the Intel family of IXP systems [12] and the Hifn PowerNP [2]. Another benefit of employing network processors for network measurements is that packets can be monitored and measurement results can be pushed to the user in real-time. An important question arising from the multiprocessor nature of NPs is how to distribute various pre-processing and measurement tasks onto the different processing resources available on the NP.

3.2 DOME Architecture

Our Distributed Online Measurement Environment (DOME) architecture presented in Figure 2 is one possible solution to partition measurement tasks on an NP. The software architecture on a node is divided into two domains:

1. High data rate / low processing requirements
2. Low data rate / high processing requirements

This is similar to the conventional fast-path and slow-path division in a network router. The high data rate domain shown on the left handles the processing of each individual packet which involves packet capture, filtering, metric extraction and statistics collection.

The low data rate domain deals with functionality related to the online query subsystem. These tasks are fairly processing intensive and are performed at a frequency that is far less than the rate of packet arrival. In particular, this domain deals with decomposing a given query into its constituent parts and mapping them to the filters, metrics, and statistics maintained in the high data rate domain. Additionally, this domain also interfaces with the run-time management component in order to install new queries and export query results to the end user.

3.3 Packet Capture and Anonymization

The first step in the measurement process is to capture packet headers. A fixed number of bytes of packet headers can be collected or we can parse sequences of packet headers to capture a variable number of bytes. The second

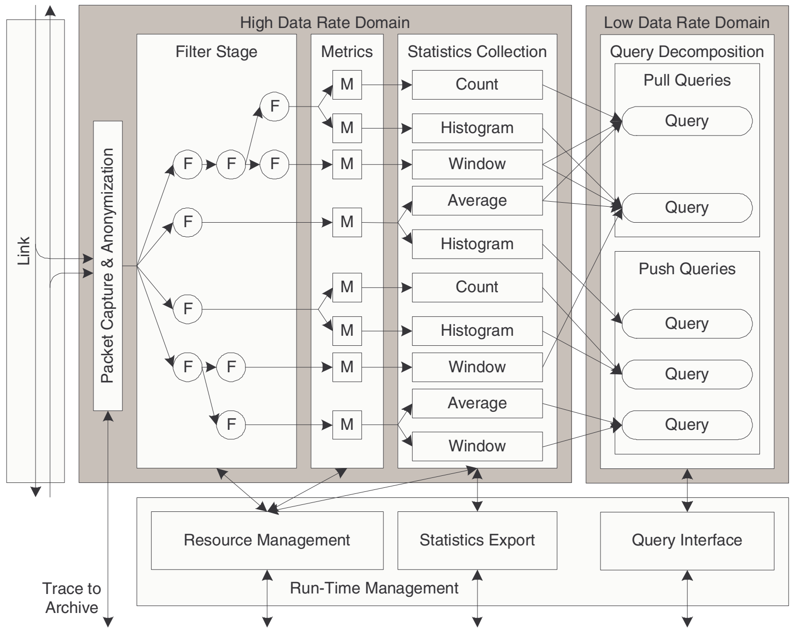


Figure 2. Distributed Online Measurement Environment (DOME) Node Architecture.

approach has the advantage that encapsulated headers and headers with options can be captured in their entirety.

Depending on the operational environment of the measurement system, it may be necessary to anonymize fields in the packet header. Several techniques for anonymizing various header fields are described in [19]. A typical example is IP address anonymization, which provides a level of privacy to network users as traces cannot be correlated back to a particular host computer. In general, it is desirable to use an anonymization mechanism that preserves as much of the relevant properties of the data field as possible. For IP address anonymization it is common to use "prefix-preserving" anonymization that maintains the subnet relationship between addresses while assigning pseudo-random network addresses [15, 26, 23]. The latter two anonymization algorithms can be initialized to generate a consistent anonymization across different nodes and thus are particularly suitable for distributed measurement. We have also shown in [23] that such anonymization can be performed at Gigabit data rates. Thus this feature can be integrated into an online measurement platform.

It is important that the anonymization step is placed early in the measurement process. This ensures that the following measurement steps do not accidentally collect data that should be protected. Once header collection and anonymization is complete, this measured data is sent out of the measurement node to the centralized trace collection database. This stage is always performed for all link data.

3.4 Online Traffic Analysis

In the **filter stage** in Figure 2, packets are classified according to filters that match the queries in the system. The filtering ensures that the statistics collection component of the measurement system only collects data from the desired subset of observed traffic (e.g., a header field value or flow identifier). The filtering stage can also be used to augment packets with meta-information (e.g., flow classification result).

The **metrics extraction** step in Figure 2 is responsible for identifying the data fields of the packet that are used to collect statistics. It is important to note that these fields can be different from the fields used for filtering. Example metrics include counts, header fields and meta information.

The **statistics collection** component in Figure 2 identifies and collects the actual data that is used for answering user queries. There are a large number of potential types of statistics (e.g., counter, sliding window, histogram). This results in a tradeoff between the amount of information that is retained in a statistic and the amount of memory and processing that is required to maintain it.

The accuracy at which these statistics are maintained depends on the amount of memory that can be provided for a particular query. Multi-resolution counters can be used to maintain statistics over different time windows (e.g., several counters that keep track of a value over the last second, the last minute, etc.).

3.5 Storage of Packet Headers

In addition to real-time traffic monitoring, our monitoring platform also archives packet headers for future queries. Traditional post processing of this archived data can be used to extract measurements that are not feasible to be performed via real-time querying due to resource constraints. The tradeoff here is that post processing is more time consuming than real-time monitoring. Querying of archival data is necessary for many applications such as data mining of packet headers to detect unusual trends, analysis of historical trends (e.g., growth in P2P traffic), and port-mortem analysis of certain events (e.g., traceback of security attacks).

Our current system assumes that each NP-based capture card is connected to a high performance server over a dedicated point-to-point Gigabit Ethernet link. Packet headers and other meta-data such as the GPS timestamp and other flow-level statistics are streamed to the server over this link. The server stores the incoming stream of packet headers and the associated meta-data to local high-performance storage system (such as a RAID array).

3.6 Run-Time Management

The run-time management component in Figure 2 is primarily responsible for resource management of queries. The resource management block is required if a query needs a new filter, metric, or statistic to be installed. This block ensures that sufficient computing resources are available in order to install a new query component. The statistics export block extracts the data stored in various statistics that are required to answer a query.

One main issue in designing an online measurement system is the limitation in resources that can be dedicated to queries. Not only is processing limited due to real-time constraints, but memory is also a scarce resource on embedded network systems. In order to respond to queries over different time scales, the system needs to maintain a history of all collected statistics. We have chosen to decompose queries into many smaller components (filters, statistics, etc.) so that they can be used across multiple queries. The sharing of filters and statistics between queries allows the system to support a large number of queries while limiting the number of distinct filters, metrics, and statistics. In order to reduce the number of measurements that need to be made, ATMEN [14] also attempts to reuse metrics along the time, space and application axes.

Due to resource constraints, it is infeasible to maintain statistics at full resolution for extended periods of time. Therefore, it is necessary to aggregate the statistics collected at different resolutions. We introduce a hierarchy of statistics that is maintained by our system in Figure 3.

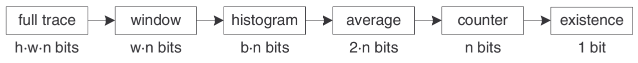


Figure 3. Hierarchy of Statistics Maintained by Query System. The required memory size of each statistics is shown for n values (header size h, window size w, histogram with b buckets).

Different levels of resolution require different amounts of memory for storage.

At the highest level the *existence* statistic provides information regarding the occurrence of a particular event or value and provides the least information. This requires the least amount of storage space (1 bit). This statistic can be extracted from the *counter* statistic which counts the number of occurrences of a certain event or value. If the counter value is non-zero, it implies that a certain event has occurred. The counter requires n bits of storage. The *counter* statistic can be derived from the *average* of a particular value. To compute the *average*, we require only two values – the sum of all samples, and the number of samples seen. Similarly, a *histogram* contains all the information required to compute an *average*. The *histogram* is assumed to have b buckets, each of size n bits. The *window* statistic stores the last w samples and contains the necessary information to construct a *histogram*. Finally, the lowest level of the statistic is the *full trace* headers which contains the most information, but also require the maximum amount of storage space. Parameter h is used to specify the size of the header.

This kind of **spatial aggregation** can be performed to adapt the statistic to the type of query that is requested. For example, if a new query that requires the average packet size is input to the system, the system needs to check if a pre-existing query maintains a statistic that is higher in the hierarchy that the *average* statistic. If this is the case, then a statistic does not need to be stored and the required information can be obtained by aggregating samples that have already been collected.

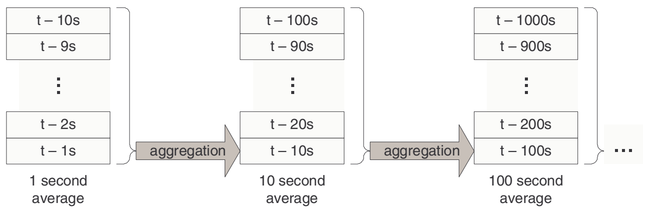


Figure 4. Aggregation of Statistic Samples over Different Time Scales by Query System.

Similar to spatial aggregation, Figure 4 shows an example of **temporal aggregation** (or "aging"). This is performed to aggregate samples along the time domain by averaging multiple samples that were collected at short time intervals into a single sample that represents a coarser time interval. The example in Figure 4 shows two levels of temporal aggregation. The first level shows how ten 1-second samples are aggregated into one 10-second sample. The second level shows how ten 10-second samples are aggregated into one 100-second sample. The term *base* refers to the number of samples that we aggregate (the *base* in this particular example is 10). In general, to limit the overall memory use of a single statistic, an upper bound on the resolution is enforced.

The choice of a *base* has a significant impact on the amount of processing and memory required to perform temporal aggregation. If the *base* is too low, less memory is required since fewer samples need to be stored, but more processing is required since aggregation needs to be performed more frequently. As *base* increases, processing requirements decrease, but memory requirements increase since more samples need to be stored. This trend is shown in Figure 5. Ideal choices for the value of *base* depend on the processing capabilities of the run-time management system and the resolution at which query results need to be provided.

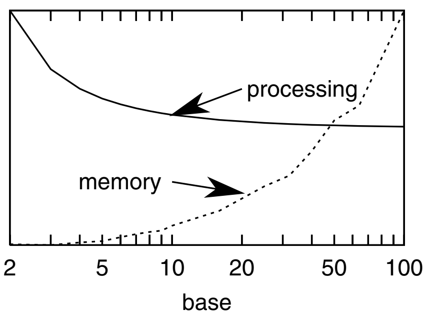


Figure 5. Cost of Aggregating Samples. The two plots show the variation in processing and memory requirements as the aggregation frequency (base) changes.

4 Queries

4.1 Design Tradeoffs

While a rule-based interface has been traditionally employed for monitoring network packet streams, it is not *a priori* evident whether such an interface is sufficiently expressive for instantiating arbitrary continuous queries. In contrast, continuous query systems [4] designed in the database community employ SQL-like query languages. The online query system described in Section 4.2 draws on many of the concepts used in this domain, but separates the components of the system so that they map onto distinct parts of the DOME architecture.

A more radical approach is to eliminate the query interface altogether and to allow high-level applications to download arbitrary code to perform queries on a capture node. Assuming that the security implications of downloadable code can be addressed using sandboxing techniques, the approach raises a different flexibility versus efficiency tradeoff. Clearly, the ability to download and execute arbitrary code is less limiting than a pre-defined query interface. However, the approach is also potentially less efficient, since it requires the application designer to hand-craft query optimizations – an effort that is likely to be less efficient than a highly optimized query engine. The issue of whether the additional flexibility warrants this loss in performance is still an open issue.

The DOME architecture supports two types of queries– queries on live data and queries on stored data. Live queries are typically continual queries which are run on the measurement node, while stored queries are typically one time queries over archived data present at the trace collection databases.

4.2 Online Queries

In many cases, queries about network traffic characteristics go beyond simple packet counts. Answers to such queries can be composed by combining the results of individual statistics that are collected on the microengines. Live queries need to be structured in such a way that they can be easily parsed and mapped to the different components of the DOME system. The basic component of a query is a *querylet* which has the form shown below:

```
<Querylet>=<Filter>* <Metric> <Statistic>
```

A querylet is a request sent to the capture node to collect data and process it. The labels correspond to the components of the high data rate domain of the DOME architecture shown in Figure 2.

The sequence of `<Filter>` specifications determine what subset of the traffic is considered for a particular query. The subset of traffic to be queried can be filtered based on the type of header (e.g. IP, TCP), and optionally, a certain bit or a range of bits of that header matching a particular value. Multiple filters can be combined together. The `<Metric>` label specifies what aspect of each packet is considered in the query. The header fields specified here can be different from the header fields specified in the filter. The `<Statistic>` label determines how the packet data (specified by the `<Metric>` label) is stored with respect to the query.

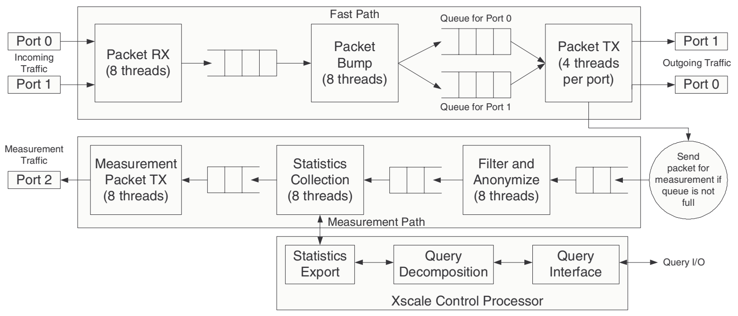


Figure 6. Prototype Measurement Node on the IXP2400 Network Processor.

We currently define six types of statistic collection data structures – a single bit flag to check for existence, a basic counter, an average, a histogram, a window, and a full trace. Some of these data structures require further parameters to be specified which determine how much memory is required (see Figure 3).

A querylet by itself contains all the information required to collect and process data for a query. However, it does not inform the capture node as to how the query data is to be exported to the end user. In order to handle this, two more labels are added to complete the query specification:

```
<Query>=<Querylet>+ <Action> <Report>
```

The `<Action>` label is used to combine the results of two or more querylets. For example, it can instruct the query system to return the ratio of two querylets which count specific types of packets. `<Report>` specifies how to report the result of the query to the user (e.g., single summary, periodic reports).

4.2.1 Query Types

There are two types of queries that can be performed on the measurement system: (1) Pull Queries and (2) Push Queries. Queries that "pull" information from the measurement node are comparable to those done on conventional packet trace collection systems. The query is sent to the system, the appropriate information is retrieved and sent in response. With the online operation of our system, another type of query is possible. "Push" queries are such that they continuously monitor the packet stream. At periodic intervals or when a particular condition is matched, a response is generated by the system.

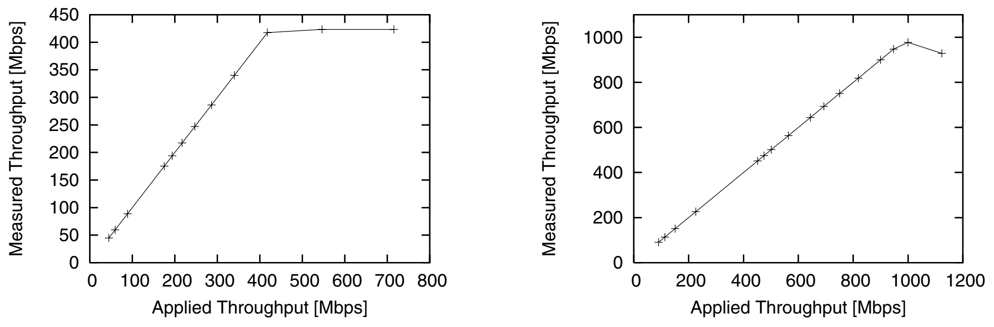
To achieve sharing between queries, the run-time management component needs to keep track of all installed filters, metrics, and statistics. When a new query is presented to the system, it is first decomposed into its components. The run-time management component checks if existing components can be reused. This can happen at several levels. For example, a query may reuse a filter that was installed as part of a previous query, but may need to install a new statistic and/or metric. As another example, the query may match existing filters and metrics and the statistic could be computed using existing statistics utilizing the hierarchy shown in Figure 3. In case a new component has to be installed, the run-time management system needs to check if enough resources are available. The memory requirements of various statistics are well defined using the parameters shown in Figure 3.

To illustrate the decomposition process, we discuss a simple example query. "What is the fraction of TCP SYN packets to other TCP packets?" This query would translate into two querylets: (1) a count of TCP SYN packets and (2) a count of all TCP packets. In order to do this, two filters are necessary: one that filters TCP packets out of the stream of all packets and a second one that checks for the SYN bit in the TCP header. This example requires only a count of packets and does not need a particular metric. The action between the querylets is a simple division of the values from both counts. The statistics component keeps counters for both querylets.

5 Prototype

We have implemented a prototype with basic measurement functions and query capability to illustrate the feasibility of the proposed architecture. The prototype is able to collect packet traces, perform anonymization, and determine several basic statistics, which can then be queried through a command line interface. This system is currently operational on the main Internet access link of the University of Massachusetts.

The prototype implementation is based on the IXP2400 network processor platform which can be found in the Radisys ENP-2611 [22] card. The IXP2400 contains eight microengines which are highly optimized for packet processing in the data plane. Each microengine contains eight threads with a zero overhead context swap. Additionally, an XScale processor is present for performing control plane related tasks. The data flow and allocation of tasks to the



(a) Small Packets

(b) Large Packets

Figure 7. Prototype Measurement Results.

underlying NP components is shown in Figure 6. The "measurement path" in Figure 6 is how we have chosen to implement the architecture shown in Figure 2. The implementation allows the measurement process to be present as a standard function in future routers through an additional "fast path" shown in Figure 6. The fast path is responsible for performing any packet processing that is done by the router (e.g. IPv4 forwarding).

The system has 3 ports – Port 0, Port 1, and Port 2. Ports 0 and 1 are used for handling normal packet traffic along the fast path. Port 2 is exclusively used for transmitting measurement traffic. In our prototype, the fast path simply bumps incoming traffic from Port 0 to Port 1 and vice versa. An entire microengine is allocated for each of the packet receive, packet bump and packet transmit (on Port 0 and Port 1) tasks. Once a packet has successfully proceeded through the fast path, it is enqueued to the measurement part of the system. If this queue is full, the packet is dropped and no measurement tasks are performed on it. Thus, the measurement path has a minimal impact on the performance of the network processor in the fast path.

The measurement path consists of three microengines which perform the tasks contained in the "high data rate domain" in Figure 2. The first microengine performs a filtering operation on the packet since only certain types of packets may need to be measured as discussed previously. Then, the packet headers are parsed and collected, and the IP addresses are anonymized. This microengine corresponds to the IP address anonymization and filtering stages shown in Figure 2. The second microengine performs any measurement related processing and collects statistics that may be required. This microengine corresponds to the metrics and statistics collection stages in Figure 2.

Currently, a number of common statistics are collected that include basic packet counts (e.g., IP, TCP, UDP, etc.) and distributions of layer 3 protocols, packet size, and TCP port numbers. For every packet that enters the measurement path, a "measurement packet" is generated which contains a trace of the packet headers and some meta data. The transmit processing for this packet is done by the third microengine in the measurement path. These packets are collected by a central trace collection database.

All query related processing is performed on the XScale control processor. The query system consists of three components shown in Figure 6. The Query Interface is the frontend for the whole system and is used to input queries into the system and obtain the results for queries. The Query Decomposition block parses the query and splits it into its constituent filters, metrics and statistics. Our prototype is installed with a set of predefined filters and statistics which the query system is aware of. Additionally, the temporal aggregation technique described in Section 3.6 is performed on samples. Since the network processor is software programmable, we have the capability to dynamically install query components when needed. This is currently work in progress. The microengines maintain the statistics collection data structures in high speed SRAM memory. The Statistics Export component in Figure 6 is responsible for extracting the required values from this memory and reporting them to the Query Interface.

5.1 Evaluation

5.1.1 Simulation Performance

The measurement system was simulated on the simulator for the IXP2400 network processor. The prototype parsed packet headers, anonymized IP addresses, encapsulated the trace into an UDP/IP packet for transmission and performed basic statistics collection. Simulation traffic consisted of unidirectional 60 byte TCP/IP packets over Ethernet. We were able to sustain a transmit rate of up to 1120Mbps (approximately 900,000 packets per second) on the measurement port (Port 2). In practice, this rate is not feasible due to the bandwidth limitations on the interface (1Gbps), but it

Table 1. Comparison of measurement reports of IXP2400-based DOME node and DAG card for a 24 hour period.

Measurement		DAG 4.3GE	IXP-based DOME	Difference
Packet count	up-link	3,558,914,492	3,558,914,492	0
	down-link	3,677,879,928	3,677,879,928	0
Byte count	up-link	1,576,830,624,304	1,576,830,624,304	0
	down-link	1,430,657,109,468	1,430,657,109,468	0
IP statistics	IP packets	7,236,735,411	7,236,735,411	0
	IP options	11	11	0
	non-IP packets	59,009	59,009	0
TCP statistics	TCP packets	6,576,779,992	6,576,779,992	0
	TCP options	835,325,139	835,325,139	0
	TCP SYN	178,529,684	178,529,684	0
	TCP FIN	93,456,130	93,456,130	0
	TCP RST	38,633,106	38,633,106	0
UDP stats	UDP packets	629,678,683	629,678,683	0

gives a rough estimate of the processing limitations on the system.

5.1.2 Operational Results

The measurement hardware was also tested on the Internet access link of the University of Massachusetts. The node was observed to be functional at data rates of up to 140,000 packets per second. No higher data rates were observed during the test.

In a lab experiment, synthetic traffic of up to 1Gbps was generated with various packet sizes. The measurement throughput results are shown in Figure 7. With large packets (1510 bytes), the full 1Gbps of observed traffic can be processed and "measured." Sustained trains of very small packets (60 bytes) cause congestion on the measurement output port because measurement packets contain additional information (meta data) that needs to be sent to the database. We are currently looking at changing the node-database interface to transmit packet headers in aggregated fashion to solve this problem.

5.2 Verification

In order to verify the proper operation of our prototype, we ran our IXP based capture node in parallel with an Endace GIGEMON system which contains a DAG 4.3GE [8] card. Both measurement systems were used to monitor the same link by means of an optical splitter. In one set of experiments, both monitoring systems were setup to count the same statistics on the packet stream over a 24 hour period. The results are shown in Table 1 and show no differences between the numbers reported by both systems. In another set of experiments, the actual packet headers captured by both measurement systems were compared byte-by-byte to ensure that our measurement node performs packet capture without any errors.

Table 2. Comparison of Post-Processing Measurement to Online Measurement for 1-Hour Anomaly Detection Experiment.

	Post-Processing	Online Measurement
Trace storage requirement	26 GB	N/A
Average data rate	58 Mbps	64 kbps
Anonymization	42 min	online
Anomaly detection	7 min	online
Minimum time to detection	49 min	real-time

5.3 Comparison to Post-Processing Measurement Systems

The main benefit of our proposed architecture is highlighted when comparing the performance of online measurement to that of measurement by post-processing. We have implemented a recently published anomaly detection algorithm by Gu et al. [10]. The processing consists of classifying packets by their port numbers and traffic type (UDP, TCP data, TCP SYN, etc.) and counting their occurrence in a 1-second time interval. The results are compared to a baseline distribution by computing the relative entropy for each packet count. If the entropy exceeds a given threshold 30 or more times in a 1-minute window, traffic is considered anomalous for this particular category of traffic.

In the post-processing scenario, a packet trace is collected, then anonymized, and then processed to extract packet counts. In the online scenario, packet counts and entropy values are computed on the online measurement node. Table 2 shows a comparison of both approaches for a 1-hour experiment. The amount of data that needs to be transferred is three orders of magnitude less in the online system and anomalies can be detected without delay (or at user specified intervals). These results clearly show that online measurement is preferable in cases where results can be computed by the measurement node.

6 Summary

In summary, we have presented an architecture for a passive distributed measurement system. The capability to correlate measurement information from different measurement nodes in the network provides a deeper insight into large-scale network behavior. The ability to process traces on the measurement node itself is the key enabler to answer queries in real-time which in turn reduces load on the trace collection end-system. The decomposition of queries into simple measurement statistics and filters provides an extensible way to answer complex questions regarding network operation. Our prototype system shows the feasibility of this design and the significant performance benefits that can be achieved over post-processing network measurement systems.

References

[1] Advanced Broadband Communications Center. *SMARTxAC-Passive Measurement and Real Time Analysis*, 2005. http://www.ccaba.upc.es/contingut.php?dir=Projects/SMARTxAC.

[2] J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhikhi, M. Siegel, and M. Waldvogel. IBM PowerNP network processor: Hardware, software, and applications. *IBM Journal of Research and Development*, 47(2/3):177–194, 2003.

[3] P. Arlos, M. Fiedler, and A. Nilsson. A distributed passive measurement infrastructure. In C. Dovrolis, editor, *Passive and Active Network Measurement*, number 3431 in LNCS, pages 215–227, Boston, MA, Mar. 2005. Springer.

[4] S. Babu and J. Widom. Continuous queries over data streams. *ACM SIGMOD Record*, 30(3):109–120, September 2001.

[5] S. Bhattacharyya and S. Moon. Network monitoring and measurements: Techniques and experience. In *Tutorial at ACM Sigmetrics 2002*, Marina Del Rey, CA, June 2002.

[6] J. Coppens, E. P. Markatos, J. Novotny, M. Polychronakis, S. V., and S. Ubik. SCAMPI-a scaleable monitoring platform for the internet. In *International Workshop on Inter-Domain Performance and Simulation*, Budapest, Hungary, Mar. 2004.

[7] C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, and O. Spatscheck. Gigascope: High performance network monitoring with an SQL interface. In *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, page 623, Madison, WI, June 2002.

[8] Endace Measurement Systems. *DAG4.3GE Network Monitoring Interface Card*, 2005. http://www.endace.com/dag4.3GE.htm.

[9] C. Fraleigh, S. B. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot. Packet-level traffic measurements from the Sprint IP backbone. *IEEE Network*, 17(6):6–16, Nov. 2003.

[10] Y. Gu, A. McCallum, and D. Towsley. Detecting anomalies in network traffic using maximum entropy estimation. In *Internet Measurement Conference*, Oct. 2005.

[11] G. Iannaccone, C. Diot, D. McAuley, A. Moore, I. Pratt, and L. Rizzo. The como white paper. Technical Report IRC-TR-04-017, Intel Research Cambridge, Sept. 2004.

[12] Intel Corp. *Intel Second Generation Network Processor*, 2005. http://www.intel.com/design/network/products/npfamily/.

[13] U. Keller and J. But. Netsniff - design and implementation concepts. Technical Report CAIA-TR-050204A, Swinburne University of Technology, 2005.

[14] B. Krishnamurthy, H. V. Madhyastha, and O. Spatscheck. ATMEN: A triggered network measurement infrastructure. In *Proceedings of the 14th International World Wide Web Conference (WWW)*, Chiba, Japan, May 2005.

[15] G. Minshall. *TCPDPRIV*. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html, 1.1.10 edition, Aug. 1997.

[16] A. Moore, J. Hall, E. Harris, C. Kreibech, and I. Pratt. Architecture of a network monitor. In *Passive and Active Measurement Workshop*, La Jolla, CA, Apr. 2003.

[17] National Laboratory for Applied Network Research. *Active Measurement Project*, 2005. http://watt.nlanr.net/.

[18] National Laboratory for Applied Network Research - Passive Measurement and Analysis. *Passive Measurement and Analysis*, 2003. http://pma.nlanr.net/PMA/.

[19] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *ACM SIGCOMM Computer Communication Review*, 36(1):29–38, Jan. 2006.

[20] V. Paxson. End-to-end routing behavior in the internet. *IEEE/ACM Transactions on Networking*, 5(5):601–615, Oct. 1997.

[21] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, June 1999.

[22] Radisys Corporation. *ENP-2611 Product Data Sheet*, 2004. http://www.radisys.com.

[23] R. Ramaswamy and T. Wolf. High-speed prefix-preserving IP address anonymization for passive measurement systems. *IEEE/ACM Transactions on Networking*, to appear. available as Technical Report TR-06-CSE-01, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst.

[24] Snort. *The Open Source Network Intrusion Detection System*, 2004. http://www.snort.org.

[25] Surveyor Home Page. http://www.advanced.org/surveyor/.

[26] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon. Prefix-preserving IP address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *Proc. of 10th IEEE International Conference on Network Protocols (ICNP'02)*, pages 280–289, Paris, France, Nov. 2002.

[27] M. Yajnik, S. B. Moon, J. F. Kurose, and D. Towsely. Measurement and modeling of the temporal dependence in packet loss. In *Proc. of IEEE INFOCOM'99*, pages 345–352, New York, NY, Mar. 1999.