

# Streams, Flows and Torrents

Nevil Brownlee\* and Margaret Murray†

*Abstract*— RTFM (RFCs 2720-2724) considers network traffic as being made up of bidirectional flows, which are arbitrary groupings of packets defined only by attributes of their end-points. This paper extends RTFM’s view of traffic by adding two further concepts, *streams* and *torrents*. Streams are individual IP sessions (e.g. TCP or UDP) between ports on pairs of hosts, while a torrent refers to all traffic on a link.

We present stream measurement work using a meter located at UCSD (University of California, San Diego) to measure response times for DNS requests to the global root and gTLD nameservers. This example shows how to configure NeTraMet to collect flow data for stream-based flow metrics, and demonstrates the usefulness of global DNS response plots for network operations.

*Keywords*— RTFM, NeTraMet, Flow Measurement

## I. INTRODUCTION: TRAFFIC, FLOWS, RTFM AND NETRAMET

Traffic measurement supports computer network management in a variety of ways. Network operators routinely analyze traffic data as part of routine traffic engineering, security incident detection and response, and billing activities. One or more of several methodologies offer strategies for tracking traffic measurements of interest. Selection among strategies often presents difficult trade-offs.

Active measurement methodologies introduce probe traffic on a link for the purpose of assessing link (or, at a higher level, end-to-end) round-trip-time (RTT), packet loss and available bandwidth. Active measurements are particularly useful for addressing questions about network activity in real time. However, probes introduce network overhead, which can be problematic on congested links. Additionally, active probes may themselves influence resulting measurements, so probing software must consider differences between TCP or UDP, how to optimize probe parameters, and how frequently to launch probes and collect measurements. For example:

- One might measure available bandwidth by flooding a link with test UDP packets; this would severely affect the link’s other users.
- One could test a web server by downloading a (non-cached) test web page at frequent intervals; this uses TCP streams and would have minimal effect on other users.

Alternately, non-intrusive passive measurement methodologies use monitors to measure both performance and workload characteristics of traffic on a particular link. While offering the capability to discern traffic trends over

\* The University of Auckland, New Zealand and CAIDA, University of California, San Diego. E-mail: [nevil@caida.org](mailto:nevil@caida.org)

† CAIDA, University of California, San Diego. E-mail: [marg@caida.org](mailto:marg@caida.org)

Support for this work is provided by DARPA NGI Contract N66001-98-2-8922, NSF Award NCR-9711092 “CAIDA: Cooperative Association for Internet Data Analysis” and The University of Auckland.

time, the sheer volume, storage, and analysis requirements when processing passively monitored traces immediately becomes an issue. Even if trace files collect only packet headers for later analysis, files can quickly become very large, requiring careful design of archiving and analysis logistics. Trace files are particularly useful for studying historical trends in many ways (e.g. the growth in usage of a new network protocol). Unfortunately, data reduction is difficult, and analyzing large trace files may be time-consuming and resource-intensive.

In 1995 the IETF’s Realtime Traffic Flow Measurement (RTFM) working group began to develop a system to enable real-time traffic data reduction and to minimize the size of captured measurements. The RTFM traffic measurement system, documented in [7] and [10], consists of a general model of traffic flows, together with a distributed, asynchronous system for measuring those flows.

RTFM flows are arbitrary groupings of packets defined only by the attributes of their endpoints. Endpoints may be either 1) a complete 5-tuple (Protocol, Source IP address, Destination IP address, Source port number, Destination port number); 2) a pair of netblocks (e.g., 192.168.1/24 and 192.168.2/24); or 3) two lists of netblocks.

The collection of all flows making up the total traffic on a network link is defined here as a *torrent*. While RTFM enables measurement of a link torrent, this measurement is of questionable use for managing the link.

The power of the RTFM approach lies in its asynchronous architecture, which simplifies operation and provides redundancy, and its ability to identify and express characteristics concerning the traffic flows comprising the torrent. Three network entities comprise an RTFM system [3]:

**Meters** gather data from packets so as to produce flow data.

**Meter Readers** collect flow data from meters.

**Managers** specify real-time data reduction by downloading configuration data (called ‘rulesets’) to meters, while specifying the frequency interval at which meter readers read flow data.

Managers use rulesets to extract attributes (measurements) of interest from RTFM flows. There are three general types of attributes:

1. Address attributes: source and destination attributes, e.g., TransAddress (IP port number), PeerAddress (IP address) and AdjacentAddress (layer below peer, e.g. MAC address). These attributes describe a flow’s address at the transport, network and adjacent layers; by identifying a flow’s endpoints they specify which packets within the torrent are considered part of the flow.

2. Summary attributes: information about flows, e.g., time of first and last packet in flow, total byte count, total packet

count. Note that since RTFM flows are bi-directional, ‘to’ counters concern packets from source to destination, and ‘from’ counters concern packets from destination to source. Summary attributes [7] have scalar values. Packet and byte counts are never reset, so calculation of the difference between two successive meter readings yields a packet or byte count for the interval.

3. Computed attributes: Rulesets provide a mechanism for detecting and grouping higher-level derived measurements. RTFM Managers load rulesets, written using the Simple Ruleset Language (SRL) [8], onto RTFM meters. Meters then execute rulesets upon reading each packet header.

The RTFM working group includes distribution-valued attributes in its extended architecture [9]. This class of attributes enables implementation of flow metrics that are not expressible as scalars (e.g., packet sizes and intra-flow interarrival times). Use of distribution-valued attributes enhances the statistical value of summary attributes. Rulesets specify parameters of RTFM distributions, (i.e. limits (lower and upper), and number of buckets) to ensure that the counts are well-distributed over the buckets. Furthermore, since each bucket in a distribution is a counter, one can compute differences in distributions across successive meter readings.

NeTraMet [4] open source software is the first implementation of an RTFM system. For a step-by-step introduction to NeTraMet, see [3]. NeTraMet includes RTFM meters which run on Unix systems, an SRL compiler, and combined Manager/Readers. NeTraMet comprises a toolkit suitable for constructing production flow measurement systems. Additionally, a variant version of NeTraMet exists that uses CoralReef [2] to access packet headers. The CoralReef NeTraMet has two significant advantages; it supports access to high-speed interfaces such as the Dag cards developed by the WAND group [11], and it allows one to read packet headers from trace files. Access to archived trace files allows one to run many different rulesets (i.e. to perform different analyses) on the same data. <sup>1</sup>

## II. FLOWS AND STREAMS

Within a torrent there are usually a few flows carrying large amounts of traffic, and many others carrying only small amounts. These two kinds of flows are often described as *elephants* and *mice*. Upon examination of flows, one may discern *streams*, comprising bi-directional microflows. Streams are individual IP sessions (e.g. TCP or UDP) between ports on pairs of hosts. An extension of the RTFM meter maintains queues of streams belonging to currently active flows. While summarisation of flow activity can be useful, analysis of the streams contained within flows offers useful insights.

Streams introduce a new dimension into RTFM, making it possible to observe distributions of stream properties such as their sizes (in bytes and packets) and lifetimes. Each stream can also maintain a queue of information about recent packets, making it possible to match pairs of

<sup>1</sup>Analysis of trace files also serves to test and validate different versions of NeTraMet software.

packets (e.g., requests and responses within a DNS stream) so as to implement the ‘turnaround time’ attributes proposed in [9].

Streams also introduce new complexities into the meter in several ways:

1. Streams require more complex data structures and dynamic memory usage.
2. Streams require timeout algorithms for recognizing when a particular stream has failed or is no longer relevant for analysis.
3. Streams require a greater degree of care when writing rulesets in order to avoid ‘ambiguous’ flows that skew analysis.

The following paragraphs discuss each of these stream requirements.

### A. Meter Data Structures

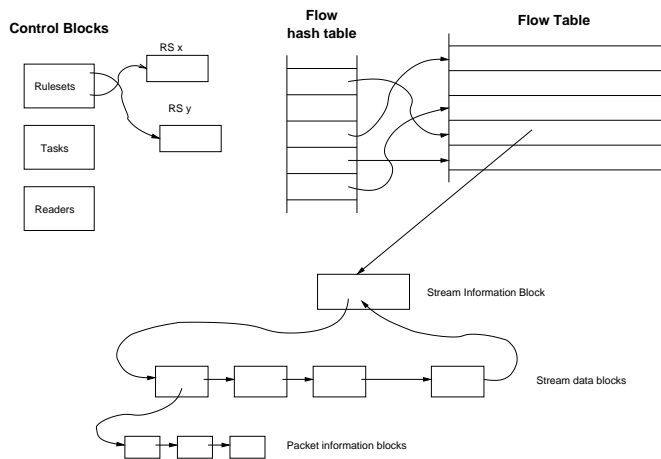


Fig. 1. NeTraMet meter data structures, showing meter control blocks, flow table, stream and packet information queues.

Figure 1 gives an overview of the Meter’s data structures. At the top left are control blocks, holding information about rulesets, tasks (rulesets being run for Managers) and Meter Readers (times when active rulesets’ flows were last read). One or more Managers set entries in the control blocks. The Meter can run many rulesets simultaneously, and many Meter Readers can read the resulting flow data.

The flow table is at the upper right; it is an array of flow data structures. Active flows form linked chains, one for each running ruleset. A meter reader makes Serial passes through the ruleset chains, reading flow data via SNMP *getblock* requests. The meter matches every incoming packet against each of the active rulesets. If the match succeeds, the Meter computes a hash of all its saved attributes (section 2), then looks up the flow in the flow hash table.

To implement recognition of streams, it was important that rulesets using streams not impose any penalties on rulesets that do not use them. To achieve this, a stream information block keeps information about a flow’s streams, and points to the flow. Each stream information block maintains a bidirectional chain of stream data structures,

making it easy to move a stream to the head of the chain of active streams. Finally, if packet-pair matching is required, each stream maintains its own queue of packet information blocks.

The NeTraMet Meter allocates memory space for all its data structures before it starts running to minimize memory management overhead. The user, via the Manager, specifies the maximum number of packet information blocks, streams, stream information blocks (i.e. number of flows that can use streams), and flows. The default numbers for each of these objects favour flow measurement over stream measurement, but are adjustable depending on what measurements are of interest.

### B. Timeout Algorithms

The RTFM architecture [7] discusses memory management for flow data. Essentially, a flow data structure must be made available when the flow becomes active, i.e. when its first packet is seen, and the memory must be recovered once the flow becomes inactive. RTFM uses a fixed timeout for its flows; its value, *InactivityTimeout*, is set by a Manager for each ruleset.

The default value for *InactivityTimeout* is 5 minutes. Flows time out if inactive for at least this period of time, and their data has been read by all interested Meter Readers. This simple approach works well for flows and it is simple to use; it requires only ensuring that upon startup a meter allocates a maximum number of flows sufficient to carry it through 3 to 5 meter reading intervals.

Streams, however, do not lend themselves to such a simple approach. Many streams are short-lived, with lifetimes of 0.5 seconds or less, and a flow will often be composed of many individual streams. When a stream terminates, the meter computes whatever metrics are specified by the ruleset, and updates the appropriate distributions. After that the stream’s memory must be recovered. In other words, while flows must persist until the Meter Reader reads its data, stream data structures must release their memory as soon as the corresponding stream terminates.

For TCP streams, NeTraMet can monitor the session state. In particular, once the TCP stream sends a single TCP RST packet or pair of TCP FIN packets (in both directions) the Meter knows the stream is terminating, and can time it out (this is TCP’s FIN\_WAIT state). This method works well most of the time, but the meter can not be sure of seeing all packets for every stream (due to e.g., packet losses, asymmetric routes, load balancing between multiple links). To catch such ‘broken’ TCP streams the meter could use a fairly long timeout interval, on the order of tens of seconds.

Such a technique is inappropriate for use with UDP streams, which are simply an arbitrary sequence of packets in one direction, together with their set of answering packets. There are no flags to convey session state; instead it is up to the Meter to time them out. To make matters worse, most UDP streams (e.g. DNS requests) are very short-lived, requiring use of a short one or two second timeout interval.

These contradictory requirements for UDP and TCP streams suggest that a better approach would be to use a dynamic timeout algorithm. [5] suggests that most flows have a steady flow rate throughout their lifetime, and uses this fact to develop an adaptive flow timeout strategy. NeTraMet uses its own simple dynamic timeout algorithm, as follows:

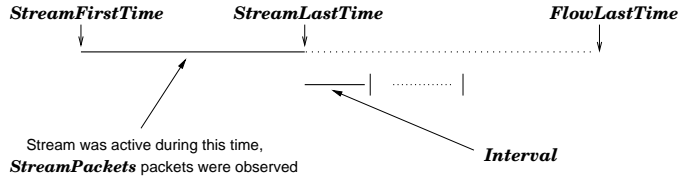


Fig. 2. Time relationships for a stream within a flow, showing the variables used by NeTraMet’s dynamic timeout algorithm.

For each flow:

$$FlowLastTime = \text{time last packet was seen}$$

For each stream contributing to a flow:

$$StreamFirstTime = \text{time first packet was seen}$$

$$StreamLastTime = \text{time last packet was seen}$$

$$StreamPackets = \text{total packets (to + from)}$$

Figure 2 illustrates these quantities. The algorithm has two parameters (implemented as compile-time variables)

$$FixedTime = \text{minimum stream timeout}$$

$$TimeMultiplier = \text{multiplying factor}$$

When a stream has been inactive for at least *FixedTime*, the meter computes its average packet interarrival time

$$Interval = \frac{StreamLastTime - StreamFirstTime}{StreamPackets}$$

The stream has been inactive for

$$InactTime = FlowLastTime - StreamLastTime$$

If  $InactTime \geq Interval \times TimeMultiplier$

the stream is considered inactive: the meter will time it out, update the flow’s stream distributions, and recover the stream’s memory.

Using a fixed minimum timeout, *FixedTime*, allows the meter to make a good initial estimate of the stream’s interarrival time. The meter revises this estimate each time it checks a stream’s activity.

Note that any dynamic timeout algorithm relies on assumptions about stream data rates. For NeTraMet’s algorithm, *FixedTime* is the longest lifetime a stream can have no matter how its packet rate varies. Beyond that, *TimeMultiplier* effectively specifies a space/mark ratio for stream traffic rates; a value of 1 allows a stream to remain active so long as its rate is reasonably constant. Higher values allow burstier flow rates.

NeTraMet uses values of 5 seconds for *FixedTime*, and 20 for *TimeMultiplier*, pragmatically chosen since they work well in practice, achieving reasonably low memory usage for streams, while allowing observation of streams with lifetimes from 10 ms to more than 60 seconds.

There is no need for a Meter to perform these calculations at fixed time intervals. Meters only need to check streams often enough to keep memory usage down to acceptable levels. Therefore, the Meter checks UDP streams after every 20th new stream observed, and TCP streams whenever the garbage collector examines the flow.

### C. Rulesets and Packet Matching

Rulesets are programs written in SRL, the Simple Rule-set Language [8]. An RTFM meter reads each packet header and executes the ruleset. A ruleset performs three functions:

1. Decide whether the packet belongs to a *required* flow. Compare the packet's address attributes to the values specified in the ruleset. If this test fails, the packet's source and destination address attributes are interchanged and the test repeats. If this 'reversed' test also fails, the Meter ignores the packet.

```
if SourcePeerAddress == (192.168.1/24, 192.168.2/24)
```

matches packets with source IP addresses in these two 24-bit subnets of 192.168/16.

2. Determine the packet's direction within its flow. In most cases the tests above answer this question. In cases where a packet matches in both directions, the first packet seen by the meter is assumed to be travelling from source to destination. For example,

```
if SourcePeerAddress == 192.168/16 &&
    DestPeerAddress == 192.168/16}}
```

matches packets travelling between any two hosts in 192.168/16

3. Specify the address granularity of the flow. For example, for packets matching the expression in (2) above we could have the meter produce separate flows for each 24-bit subnet by saying

```
save SourcePeerAddress/24;
save DestPeerAddress/24;
```

When all information needed to define the flow has been specified in 'save' statements, a `count` statement is executed. The Meter updates that flow's packet and byte counters for the matched direction.

The meter determines the direction of each packet within its flow by attempting to match each packet against a ruleset. Rulesets must keep packet matching unambiguous; this is usually not difficult to achieve. Suppose, for example, that we have a meter at the gateway between our local network, 192.168/16, and our Internet gateway. Packets going to the Internet will have *SourcePeerAddresses* within 192.168/16, packets coming from the Internet will have *SourcePeerAddresses* outside 192.168/16. We might write SRL statements to meter our traffic to and from the Internet, using the following SRL statements:

```
if SourcePeerAddress == 192.168/16 {
    save SourcePeerAddress/24; # 24-bit subnet address
    count;
}
```

When the meter sees a packet from our network, it saves the source address (using its first 24 bits) and updates the flow's 'to' counters. A packet travelling in from the Internet

will fail the test; the meter will retry it with source and destination addresses interchanged; that match will succeed, so the meter will update the flow's 'from' counters.

What will happen if the meter sees a packet travelling from one host to another inside our network? Since both source and destination address are inside 192.168/16, the packet will match in both directions! In this case the meter will use the direction of the first packet it sees for the flow as the 'to' direction. The remaining packets of the flow match properly, i.e. the meter is able to resolve this kind of ambiguity itself.

It is, however, easy to write rulesets for which a source-to-destination packet matches one flow, but an answering destination-to-source packet matches another. This is an ambiguous situation which the meter cannot resolve. If both sides of the flow use streams, the meter has no way to discern to which flow a stream belongs. In such a situation, rather than risk corrupting its stream data structures, the Meter disables both flows and writes an error message to its log file. Such disabled flows time out after their data has been read; the meter will attempt to create them again if more of their packets appear.

To see how ambiguous flows can occur, consider the following example. We have a Meter at the Internet gateway of our HOME network, and want to investigate stream lifetimes for web flows. It is important to distinguish between flows to servers located outside HOME and flows occurring within HOME. Our first attempt at a corresponding ruleset:

```
if SourceTransType == TCP save;
else ignore; # Only interested in TCP flows

if SourcePeerAddress == HOME {
    if DestTransAddress == WWW
        store FlowKind := 1; # Server outside HOME
    else if SourceTransAddress == WWW
        store FlowKind := 2; # Server inside HOME

    save FlowTime = 50.0.0!0 & 2.4.1!12000;
    # 50 buckets, log transform, 10**4 scale factor
    # => buckets from 10 ms to 120 s
    count;
}
```

The test on *SourcePeerAddress* ensures that all streams have their source at hosts within the HOME network. The tests on *DestTransAddress* and *SourceTransAddress* (IP destination and source ports) set the computed attribute *FlowKind* to indicate whether the packet's flow is to an Internet web server (*FlowKind* 1), a HOME web server (*FlowKind* 2) or is not a web flow (*FlowKind* 0, its default value).

Unfortunately, this ruleset has a flaw that allows it to create ambiguous flows. While the ruleset correctly handles all packets with one endpoint inside HOME and the other outside, it is ambiguous for packets from a HOME host to a HOME web server. For example, a packet

HOME.1 port 12345 → HOME.2 port 80

sets *FlowKind* to 1, but its answering packet

HOME.2 port 80 → HOME.1 port 12345

sets *FlowKind* to 2. Since the meter uses *FlowKind* (as well as any other attributes the ruleset has saved) to define flows, these packets cause the Meter to create two (unidirectional) flows.

To avoid this problem, one needs to write rulesets that distinguish all possible paths packets can take. For example, we could add a third *FlowKind* value to handle HOME-HOME packets, as follows:

```

if SourcePeerAddress == HOME {

  if DestPeerAddress == HOME
    store FlowKind := 3; # Host, Server both HOME

  else if DestTransAddress == WWW
    store FlowKind := 1; # Server outside HOME
  else if SourceTransAddress == WWW
    store FlowKind := 2; # Server inside HOME

  save FlowTime = 50.0.0!0 & 2.4.1!12000;
  # 50 buckets, log transform, 10**4 scale factor
  # => buckets from 10 ms to 120 s
  count;
}

```

The Meter will ignore any flows that have neither Source nor Destination in the HOME network. The modified ruleset produces three unambiguous flows. Note that systematic examination of all possible paths between networks of interest is good discipline, making a useful contribution to the design of network traffic measurements.

### III. EXAMPLE: DNS RESPONSE TIME DISTRIBUTIONS

RFC2724 [9] defines *TurnaroundTime* attributes, and points out that for this to be useful, the meter has to use streams, and perform packet-pair matching, i.e. maintain a queue of information about packets and match responses with their originating request. *TurnaroundTime* is the time interval, in microseconds, between a request and its response.

We have used this feature to investigate the behaviour of the global root and gTLD (top-level domain) nameservers. For this purpose we installed a NeTraMet meter so as to monitor all traffic between three UCSD netblocks and the Internet, and developed the following ruleset to collect DNS *TurnaroundTime* data.

```

if SourcePeerType == IPv4 save;
else ignore; # Not IP
if SourceTransType == UDP save;
else ignore; # Not UDP

TestDestAddress; # Sets FlowKind
if FlowKind == 0 nomatch; # Not root server
else {
  if DestTransAddress == DNS save;
  else ignore; # Dest not DNS port

  save ToTurnaroundTime = 50.11.0!0 & 2.3.7!700;
  # 50 buckets, log transform, 10**3 scale factor
  # => buckets from 7 ms to 700 ms
  count;
}

```

Most of the code in this ruleset concerns matching UDP packets going to the DNS port on a global nameserver.

*TestDestAddress* is an SRL define that compares *DestPeerAddress* with a list of the global nameserver addresses. If one of them matches, *FlowKind* takes a value indicating which server is the packet's destination. If the destination is not a root server, *FlowKind* is left with its default value: zero.

Results from this investigation are discussed in detail in [1], we give a brief glimpse of them here. The meter is read every 5 minutes, producing *TurnaroundTime* distributions for each reading interval.

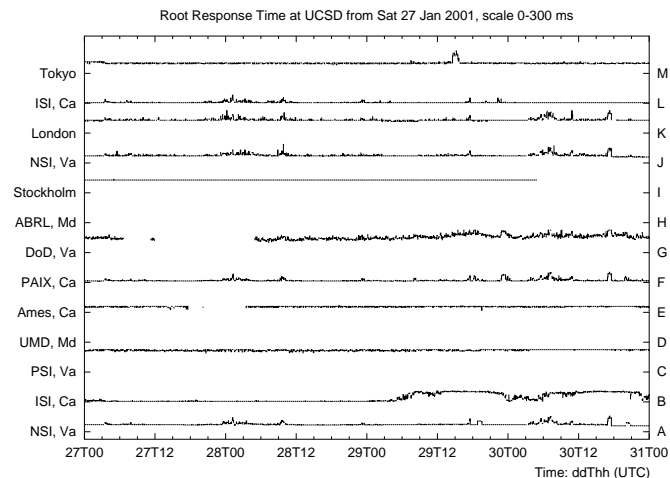


Fig. 3. Response time for the 13 root nameservers. Different behaviour for various roots reflects different routes between each root and UCSD.

Figures 3 and 4 show response times over four days late in January 2001. The 5-minute median response times for each server are plotted in small strips, scaled so that their maximum value (indicated in the plot title) lies just below the next strip. Points are not plotted for intervals with insufficient data, leaving blank sections in some of the traces.

Figure 3 shows response times for the 13 root name-

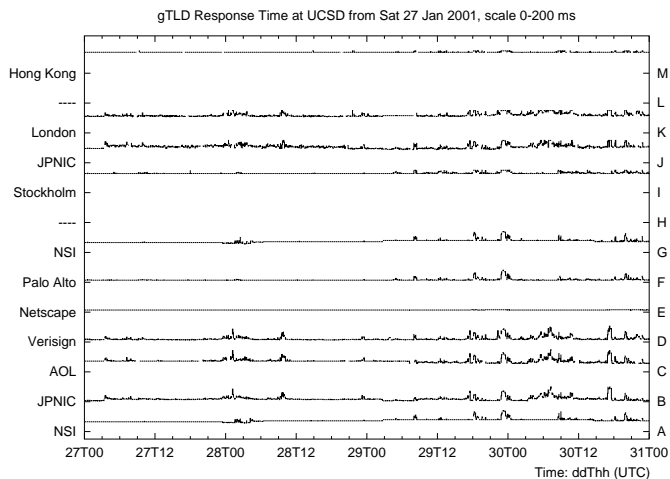


Fig. 4. Response time for the 11 gTLD nameservers. Groups of servers show similar behaviour, indicating that servers in such groups share much of the route between them and UCSD.

servers. Most of the roots have steady traces, with short bursts of increased response time. A, F, J, K and L show many of the same bursts, indicating that they share a common route. Other servers show different behaviour. C's trace, for example, is clamped at its maximum, and is much greater than A or J (other servers in Virginia), suggesting a server problem rather than network delays. B was well-behaved during the weekend, but was much poorer on Monday and Tuesday. Lastly, note the hour-long step at 1800 (UTC) on Tuesday 30 January (0900 Local Time). This behavior was common to A, F, G, J and K. The paths to these five hosts all use the same initial sub-path on transit provider CERFnet; the step on their traces suggests a denial of service attack on our CERFnet link during that hour.

Figure 4 shows response times for the 11 top-level domain nameservers. The gTLD plots are generally better-behaved than those for the roots. B,C and D shared the step at 1800 discussed above for the roots. A similar step at 2300 (UTC) on Monday 29 January was common to all 11 servers, suggesting an attack on the link being metered. Note the steps on the A and G at 0300 (UTC) on Monday 29 January, which indicate a routing change for one or more links common to those servers.

#### IV. CONCLUSION

NeTraMet is a useful research tool for studying the behaviour of streams within flows. It allows one to compute and collect distributions summarising stream behaviour in near real time, without the overheads of collecting, storing and processing trace files. Creating NeTraMet rulesets to describe flows of interest, and the distributions to be collected, requires considerable care. In particular, one must be careful that a ruleset does not allow ambiguous flows, i.e. that for every packet which matches a flow, NeTraMet can determine that packet's direction within that flow.

We presented a brief summary of an investigation of DNS response time for the global nameservers. This highlighted the need to be careful in designing the experiment, and creating a ruleset to gather the data. We gathered all data for this study using purely passive measurements at a single point. The results provide worthwhile insights into the behaviour of the Internet paths from our campus to the global root and gTLD nameservers. Realtime production of these plots could be useful for production monitoring of campus (or enterprise) Internet connections.

#### REFERENCES

- [1] Nevil Brownlee, kc claffy and Evi Nemeth, *DNS Damage*, submitted to SIGCOMM 2001
- [2] CoralReef website, <http://www.caida.org/tools/measurement/coralreef/>
- [3] Nevil Brownlee, *Using NeTraMet for Production Traffic Measurement*, Intelligent Management Conference (IM2001), May 2001
- [4] Netramet website, <http://www.auckland.ac.nz/net/NeTraMet/>
- [5] Bo Ryu, David Cheney and Hans-Werner Braun, *Internet Flow Characterization - Adaptive Timeout and Statistical Modeling*, PAM2001 workshop paper
- [6] RTFM website, <http://www.auckland.ac.nz/net/Internet/rtfm/>
- [7] Nevil Brownlee, Cyndi Mills and Greg Ruth, *Traffic Flow Measurement: Architecture*, RFC 2722, Oct 1999

- [8] Nevil Brownlee, *SRL: A Language for Describing Traffic Flows and Specifying Actions for Flow Groups*, RFC 2723, Oct 99
- [9] Sig Handelman, Stephen Stibler, Nevil Brownlee and Greg Ruth, *New Attributes for Traffic Flow Measurement*, RFC 2724, Oct 1999
- [10] Nevil Brownlee, *Network Management and realtime Traffic Flow Measurement*, pp 223-227, Journal of Network and Systems Management, Vol 6, No 2, 1998
- [11] WAND group website, <http://wand.cs.waikato.ac.nz/>