Towards Monitoring Programmability in Future Internet: challenges and solutions

Francesco Fusco, Luca Deri and Joseph Gasparakis

Abstract Internet is becoming a global IT infrastructure serving interactive and real-time services ubiquitously accessible by heterogeneous network-enabled devices. In the Internet of Services (IoS) era, monitoring infrastructures must provide to network operators fine-grained service-specific information which can be derived by dissecting application level protocols. To accommodate these new monitoring requirements network probes must be flexible, easy to extend and still be capable of analyzing high-speed network streams. Despite the increased complexity, software and hardware technologies on top of which network probes are implemented have been designed when monitoring requirements were substantially different and almost left unchanged. As a result, implementing modern probes is challenging and time consuming. In this paper we identify desirable features for reducing the work required to develop complex probes, and we present a home-grown comprehensive software framework that significantly simplifies the creation of service-oriented monitoring applications.

1 Introduction

Recent advances in wireless networks and consumer electronics technologies changed the way we are using the internet. The future internet will become a global IT infrastructure providing interactive and real-time services, hence the name internet of Services (IoS), ubiquitously accessible by heterogeneous network-enabled devices.

Francesco Fusco
IBM Research - Zurich and ETH Zurich, Switzerland, e-mail: ffu@zurich.ibm.com

Luca Deri
ntop, Pisa, Italy, e-mail: deri@ntop.org

Joseph Gasparakis
Intel Corporation, Shannon, Ireland, e-mail: joseph.gasparakis@intel.com

Understanding service behaviour and measuring the services quality over time is necessary in order to reduce costs while preserving user satisfaction. The quality of service is affected by network performance metrics, such as latency and bandwidth, but also depends on the entire network infrastructure which includes server machines, software and so on. Therefore, network operators and service providers are gradually shifting from a network centric monitoring approach to a service centric monitoring approach that provides a comprehensive and integrated view of the network and allows them to discover the root causes of service quality degradation.

The paradigm shift substantially increased the complexity of monitoring infrastructures. In fact, network probes, which are the key measurement components in today's monitoring architectures, are not only responsible for measuring basic network-level performance metrics (e.g. number of packets), but also for providing detailed service-oriented metrics. Some of these metrics, such as transaction latency, can only be measured by performing flow-level analysis [6] up to the application layer and not by analyzing single packets out of a flow context. As services are often composed of several concurrent communication flows, it is also necessary to correlate them in order to compute service-dependent metrics. For instance, the overall download time of an HTML page has to include the time for retrieving all the external objects (e.g. images) referred from the main page.

The introduction of application layer protocols analysis drastically changed requirements in terms of flexibility, performance, and programmability. Flexibility is required in order to accommodate new requirements (e.g. new protocols) and changed monitoring conditions (e.g. an existing location-fixed service is migrated to a cloud architecture). Performance is necessary for preventing packet drops while coping with the increased packet processing costs due to service-level analysis. Programmability is desirable in order to reduce the work required to extend probes and to adapt them to changing monitoring requirements. As explained in the next section, the rush for performance might have a negative impact on programmability as the use of custom or closed hardware architectures often imposes severe limitations to software applications. As a matter of fact, hardware devices are driving application design and not the other way round, thus jeopardizing flexibility and limiting portability of software applications.

The increasing complexity of monitoring tasks imposed by service-oriented network monitoring did not result in any major evolution of hardware and software monitoring frameworks on top of which network probes are built. This happened because both industries and research communities focused on specific tasks (e.g. packet capture) rather than on the creation of a comprehensive architecture allowing pluggable modules to be included or replaced in order to satisfy new monitoring requirements. This has been the driving force for the definition of a novel monitoring framework that is modular, programmable by means of software components running on commodity hardware and still capable of exploiting modern hardware technologies.

The rest of the paper is structured as follows. Section 2 lists the basic components that monitoring applications require as building blocks, and compares the state of the art of hardware and software frameworks for network monitoring. Section 3 de-

scribes the monitoring framework we have developed and positions it against similar approaches. Section 4 validates the framework against some general monitoring scenarios. Section 5 concludes the paper.

2 Towards Service-oriented Network Monitoring

The simplest traffic monitoring application is responsible for capturing traffic and computing packet-based metrics such as the total TCP traffic sent by a specific host. Flow-based monitoring applications, such as NetFlow/IPFIX [7] probes, go beyond this model by adding per-flow metrics which are derived from packet header information. Service-oriented network monitoring applications are capable of providing detailed information about services and not just about network communications. The following paragraphs describe common tasks that makes service-oriented monitoring applications substantially different from the ones listed above.

Payload Inspection. This activity is a prerequisite for properly decoding service primitives. This includes the inspection of tunnels (e.g. GRE and GTP) and encapsulations (e.g. PPTP) as well as the reconstruction of the original encapsulated payload. As of today, packet parsing is usually implemented from scratch in every application, as, beside rare exceptions [13], packet capture libraries such as libpcap [18] do not feature it, or do not release the source code such as NetBee [4] hence limiting their use to selected projects. Commercial libraries such as Hyperscan [1] feature high-speed DPI, whereas QosMOS [2] implements several protocol decoders but tight the application to their closed-source development environments. Wireshark [20] is the richest network protocol analyzer in terms of protocol supported but unfortunately packet decoding and flow analysis code are tight to the application and not available as library, making it unsuitable for integration into applications.

Service-level Packet Filtering. Most legacy filtering mechanisms such as Berkley Packet Filter (BPF) [19] do not allow the traffic to be filtered by using application-specific fields, whereas other frameworks such as FFPF [5] allow users to define application-specific filters but do not return to applications parsing information nor handle flows. Another limitation of the above technologies is the inability of efficiently adding/removing large number of dynamic filters, which is necessary for tracking services using dynamic ports such as VoIP and FTP. Contrary to BPF and FFPF, Swift [22] has been designed for offering low latency filter updates, but its scalability in terms of number of configurable filtering rules is limited.

Flow State Persistence. Maintaining protocol state and service-specific performance metrics (e.g. call setup time for VoIP) is necessary for service processing. This increases both processing workload and memory footprint. In addition, service-oriented monitoring applications require scalable, highly efficient and flexible (in terms of data types) storage architectures [15] capable of storing the retrieved service oriented metrics.

Flow Reconstruction. Per-flow packet sequence reordering, defragmentation, and IP datagram reassembly of PDUs spanning across multiple TCP segments must be performed before inspecting service primitives. Performing these tasks substantially increases both packet processing cost and memory footprint, and, therefore it must be enabled only when necessary. In addition, implementing robust and efficient TCP and IP re-assemblers is not trivial [11, 21]. Another important task is to partition the flow into sub-flows whenever several service communications are pipelined over the same connection. For instance in HTTP/1.1 peers can exchange several requests/responses over the same TCP connection.

Packet Capture. Packet loss during capture is not tolerated as it prevents service primitives from being interpreted. Instead, packet and flow-based applications can tolerate limited loss as it leads to inaccurate results while not breaking the overall monitoring system. It is worth noting that in service-oriented monitoring, packet capture is no longer the most resource consuming task, as this is a minor activity when compared to the increased packet processing costs.

Per-flow Traffic Balancing. Balancing the workload among processing units is necessary in order to leverage modern parallel architectures. When performing service oriented monitoring, packet processing costs depend on the particular traffic to be analyzed and, therefore, balancing packets across units may lead to workload unbalances.

The introduction of service analysis in monitoring infrastructures for high-speed networks raised the demand for flexible monitoring devices capable of speeding up traffic analysis applications. During the years monitoring device vendors focused mostly on performance, neglecting other aspects such as application programmability and portability across different devices designed for traffic analysis acceleration. The lack of common design guidelines across vendors has prevented the creation of a widely accepted and hardware transparent software layer beyond libpcap, which offers primitives limited to packet capture and network device management. Hardware vendors attempted to increase the processing performance in various ways including:

Capture accelerators. Packet capture accelerators such as DAG cards [12], are special purpose FPGA-based network interfaces that allow the incoming traffic to be captured and copied to the address space of the monitoring application without CPU intervention and without requiring packets to flow through the kernel layers. Often they also provide mechanisms for balancing the traffic among processor cores and filtering packets, although they are usually limited in features and are not meant to be changed in real-time as they require card reprogramming that may take seconds if not minutes. The main advantage of these hardware devices is the ease of programmability as applications can still run on commodity hardware while significantly improving their packet capture performance. For this reason capture accelerators have been widely accepted by the industry as they represent a simple solution for accelerating traffic monitoring applications, but at the same time they are of limited help in complex monitoring applications. This is because packet capture is no

longer the most resource intensive task, and therefore the speed-up achievable with packet accelerators is becoming less significant, but still not marginal.

Strong Multi-core Systems. Some vendors have embedded strong multi-core systems on network adapters in order to efficiently process packets as close as possible to the network link. Massive multi-core architectures, such as Tilera [3] use a mesh of up to 64 cores embedded directly on the network adapter. The result is that packet capture is no longer a cost as packets are processed directly on the network adapter without the need to move them to the main processor. Another advantage is that the card runs a Linux flavor and that applications can be implemented in standard C, thus significantly easing the development process.

Network Processors. Network processor boards, such as Intel 80579 [16], are special purpose monitoring devices that allow monitoring software to be executed by a processor specifically optimized for packet processing. The emphasis on speed resulted in unconventional hardware architectures providing coprocessors and several packet processing units. Developing applications for network processors is not trivial and requires a deep understanding of low-level architectural details which are usually vendor and model specific. Using external libraries for performing traffic analysis tasks is not always easy either, because applications must be implemented using languages which are similar to C, but not necessarily C compliant.

3 A Programmable Network Monitoring Framework

For a few years we have been developing an open-source kernel module for Linux systems named PF_RING [8] that we originally introduced for accelerating packet capture on commodity hardware. Over the years we have realized that it was necessary to go beyond the initial goals and to create a comprehensive framework able to tackle additional issues such as the one listed in the previous section. PF_RING is now a modular monitoring framework that allows developers to focus on implementing monitoring applications without having to deal with low-level details such as packets handling. PF_RING represents an easy to use, yet efficient monitoring framework for developing component based monitoring application. In addition, it provides a hardware transparent filtering mechanism that can be eventually accelerated by exploiting features available on modern commodity network adapters.

PF_RING substantially increases packet capture performance. Packets can be captured using standard Linux NIC drivers, but also using capture optimized drivers that allow the kernel to be completely bypassed and modern multi-core processors to be exploited. As of today, we have enhanced 1 and 10 Gbit drivers for popular network adapters by adding support for PF_RING. When running on modern servers and commodity network adapters, PF_RING can capture at wire rate from multiple Gbit links, and over 5 Mpps from 10 Gbit networks [9].

In addition to legacy BPF filters, PF_RING provides more advanced filtering mechanisms that can be used for filtering out unwanted traffic, but also for dis-

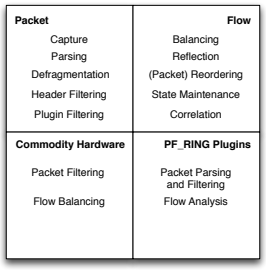

Fig. 1 Packet and Flow Management in PF_RING.

patching packets across analysis components. There are two families of filters: *exact* filters (i.e. all filter fields are specified) and *wild-carded* filters (i.e. at least one filter element has value 'any') where filter fields include MAC address, VLAN, protocol, IP v4/v6 addresses, and application ports. Exact filters are evaluated before wild-carded filters. Contrary to BPF, PF_RING parses the packet, and then checks filtering rules on it. Parsing information is returned as metadata to applications consuming the received packet. Packets are parsed once regardless of the number of consumers and filtering rules. Whenever a filter matches, PF_RING executes the action bound to it. Actions can range from simple packet dropping to more complex operations such as sending packets matching the filter to a network adapter for transmission (a.k.a. packet reflection).

PF_RING analysis components are plugins implemented as dynamically loadable kernel modules, and identified by a unique numeric identifier that can be associated with one (or more) filtering rule. When a packet matches a rule, the corresponding plugin callback is executed. Developers can define plugin hooks for filtering packets up to layer seven, and forwarding parsing information to the user-space as part of the packet metadata. Therefore, by combining filters and analysis components users can specify L7 filters such as "return only HTTP packets with method POST", which, contrary to what happens for instance in Wireshark, are executed directly at the kernel layer. Filtering rules can specify a plugin id, thus packets matching such a rule are then passed to the specified plugin for further processing. So far, PF_RING plugins include support for VoIP (Voice Over IP) [14], HTTP, and multimedia streaming.

In PF_RING, flows are used to identify and maintain state for packets matching an exact filter. They can be created automatically by means of plugin actions that are executed whenever a received packet matched. For instance a FTP monitoring application dissecting the control connection by means of a plugin, can add a new exact filtering rule for the tracking data connection as soon as the FTP client initiates a file transfer. For each flow, plugins can keep the state and maintain information about the flow being analyzed. For instance the HTTP plugin maintains information about response code and throughput, whereas the RTP plugin computes jitter and packet loss of voice packets. In a nutshell, the combination of filtering rules and

plugins, enables application developers to create powerful monitoring applications by means of simple configuration rules.

PF_RING is implemented as a Linux kernel module that can be compiled without patching the kernel source. A user-space library named libpfring, communicates with the kernel module by means of PF_RING socket and allows applications to transparently interact with the kernel module. Packets are copied by the kernel module into a circular memory buffer that is memory-mapped to user-space. This means that user-space applications can read packets from the buffer without issuing system calls. PF_RING sockets are bound to one physical network interface on which packets are received. As modern network adapters support NIC virtualization and MSI-X (message signaled interrupts), on multi-core systems PF_RING can give applications access to the various virtual RX queues contrary to vanilla Linux, which merges all queues into one. This means that hardware-based mechanisms such as RSS (Receive-Side-Scaling) for balancing network flows among RX queues mapped on processor cores, can be exploited by PF_RING applications to bound to a virtual RX queue in order to receive a portion of the traffic. This solution enables scalability as applications can be partitioned into threads or processes, each bound to a RX queue, that can process a portion of the traffic as highlighted in the Figure 2.

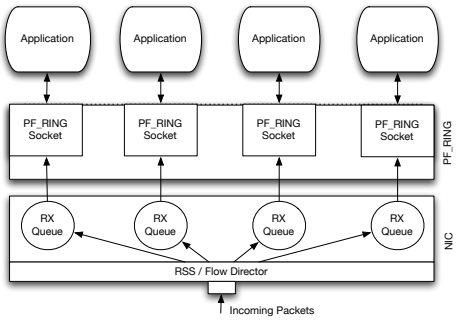

Fig. 2 Flow Balancing and RX Queues in PF_RING.

In some cases it might be useful to overcome RSS and assign selected flows to a specific RX queue in order to create specialized traffic analysis applications each sitting on a specific queue. In order to achieve this goal, we have recently added into PF_RING support for the latest generation of network adapters such as Intel 82599 controller that allows the driver to force flow balancing to cores by means of a mechanism called flow director (FD) [17]. Binding specific network flows to a non-existing core (e.g. to a core id that is greater than the number of available processor cores) instructs the adapter to drop such flow, hence implementing a wire-speed traffic filter and balancer. PF_RING comes with a specialized driver for this adapter that allows applications to transparently set FD rules [10] whenever a filtering rule is set. This means that whenever an application adds/removes a filtering rule, the PF_RING filtering engine attempts to transparently set the filter in hardware if the

adapter supports it. The result is that unwanted packets are dropped before they hit the driver, hence reducing the amount of packets that need to be processed in software. Captured packets are still filtered in software as the network adapter might not support at all or feature limited hardware filtering capabilities with respect to the filtering rules supported by PF_RING.

The combination of native multi-core/virtual RX support, support of hardware flow filtering/balancing, and in-kernel protocol dissection and analysis, makes the PF_RING framework ideal for the creation of modular and efficient traffic monitoring applications. The following section shows how this technology can be efficiently used for creating service-oriented monitoring applications.

4 Using PF_RING For Network Service Monitoring

Over the years, network applications have been constantly updated to implement the latest innovations in security. Although firewalls and IPS (Intrusion Prevention Systems) have been deployed at network borders in order to prevent unwanted communications, it is still necessary to deploy monitoring applications for discovering traffic that circumvents the security policies. This trend is driven, for example, by the use of generic protocols such as HTTP for transporting data and by the spread of technologies for creating network overlays on which freely exchange data. Security threats are also caused by unauthenticated service requests, user service abuse, misbehaving clients and permissive access rules. Web-services technologies and cloud computing are examples of traffic that needs to be carefully inspected in order to implement what is generally called trustworthy Internet. Although most Internet protocols are managed by many security systems already available on the market, it is often necessary to implement fine-grained tools for controlling selected protocol requests and also checking those protocols (e.g. network database communications) that are often not supported by monitoring appliances. Given this, it is necessary to move from packet to service-oriented monitoring in order to monitor the expected service agreements and usage policies. This requires:

- Going beyond packet header monitoring and inspecting the packet payload in order to analyze the service parameters and validate the responses.
- Computing detailed service metrics in addition to generic metrics such as throughput, latency and used bandwidth.
- Correlating various service requests in order to create a unique service data record rather than several service access requests all related to the same master request.

PF_RING simplifies the process of building service-oriented applications as it provides:

- Filtering, balancing and packet reflection capabilities for implementing simple packet filtering and balancing devices. This allows network administrators to bal-

ance the monitoring workload across processor cores which is a key requirement for performing complex resource consuming analysis tasks. To the best of our knowledge, PF_RING is the only open-source framework that can successfully exploit native hardware flow prioritization mechanisms implemented by modern network adapters in the context of traffic monitoring.
- An extensible plugin-based architecture that can be used for inspecting various protocols including Internet (e.g. web and email) and transactional (e.g. database) communications. Developers can focus on dissecting and analyzing packets while leaving the duty of dividing packets per-flow, reordering and discarding duplicates to the framework. The framework is responsible for maintaining per-flow information including protocol metrics and service request parameters.
- Filtering rules for early discarding packets that are not due to be analyzed, and for dissecting selected flows using a specific plugin.
- Correlating flows by exploiting the intra-flow framework mechanisms, for alerting specific plugins whenever a certain flow is created, deleted or updated.

As of today, the PF_RING framework has been successfully used for simplifying the development of complex and yet efficient monitoring software for real-time services [14] and HTTP-based applications. The performance evaluation of the filtering infrastructure can be found in [10], whereas [9] reports the packet capture performance.

5 Conclusions

In this paper we showed that network monitoring goals have changed in the past years and that the focus shifted from packet-level analysis to fine-grained service monitoring. The shift requires easy to develop and extend monitoring probes capable for performing complex analysis tasks on modern high-speed networks by leveraging the latest innovations in computer hardware. High-performance and ease of extensibility can be achieved by creating simple building blocks for handling various low-level activities which allows application developers to focus only on the specific problem they are tackling. From a survey of the various software and hardware technologies available, we came to the conclusion that even if there are several solutions available for tackling specific monitoring problems, there is not a comprehensive framework that can be used as a foundation for developing complex monitoring applications. This has been the driving force for creating PF_RING, an open-source flow analysis framework developed by the authors, which has been successfully used to tackle different service monitoring problems including real-time analysis of multimedia streams and web communications.

References

1. Hyperscan. http://sensorynetworks.com/Products/HyperScan/.
2. Protocol plugin library. http://www.qosmos.com/products/protocol-plugin-library.
3. A. Agarwal. The tile processor: A 64-core multicore for embedded processing. Proceedings of HPEC Workshop, 2007.
4. M. Baldi and F. Risso. Using xml for efficient and modular packet processing. In *Proc. of Globecomm*, New York, NY, 2005.
5. H. Bos, W. de Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis. Ffpf: fairly fast packet filters. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 24–24, Berkeley, CA, USA, 2004. USENIX Association.
6. N. Brownlee, C. Mills, and G. Ruth. Traffic flow measurement: Architecture. RFC 2722, 1999.
7. B. Claise. Specification of the ip flow information export (ipfix) protocol for the exchange of ip traffic flow information. RFC 5101, 2008.
8. L. Deri. Improving passive packet capture:beyond device polling. In *SANE 2004: Proceedings of the 2004 System Administration and Networking Conference*. USENIX Association, 2004.
9. L. Deri and F. Fusco. Exploiting commodity multi-core systems for network traffic analysis. Technical Report, 2010.
10. L. Deri, J. Gasparakis, P. J. Waskiewicz, and F. Fusco. Wire-speed hardware-assisted traffic filtering with mainstream network adapters. In *NEMA '10: Proceedings of the First International Workshop on Network Embedded Management and Applications*, page to appear, Niagara Falls, Canada, 2010.
11. S. Dharmapurikar and V. Paxson. Robust tcp stream reassembly in the presence of adversaries. In *SSYM'05: Proceedings of the 14th conference on USENIX Security Symposium*, pages 5–5, Berkeley, CA, USA, 2005. USENIX Association.
12. S. Donnelly. Dag packet capture performance. http://www.endace.com, 2006.
13. K. Fuji. Jpcap. Homepage http://netresearch.ics.uci.edu/kfujii/jpcap/doc/.
14. F. Fusco, F. Huici, L. Deri, S. Niccolini, and T. Ewald. Enabling high-speed and extensible real-time communications monitoring. In *IM'09: Proceedings of the 11th IFIP/IEEE international Symposium on Integrated Network Management*, pages 343–350, Piscataway, NJ, USA, 2009. IEEE Press.
15. F. Fusco, M. Stoecklin, and M. Vlachos. Net-fli: On-the-fly compression, archiving and indexing of streaming network traffic. In *Proceedings of the 36th International Conference on Very Large Data Bases (VLDB)*, page to appear, 2010.
16. Intel. 80579 integrated processor. http://www.intel.com/design/intarch/ep80579, 2010.
17. Intel. 82599 10 gbe controller datasheet. Rev. 2.3, 2010.
18. V. Jacobson, C. Leres, , and S. McCanne. Libpcap. Homepage http://www.tcpdump.org.
19. S. McCanne and V. Jacobson. The bsd packet filter: a new architecture for user-level packet capture. In *USENIX'93: Proceedings of the USENIX Winter 1993 Conference*, pages 2–2, Berkeley, CA, USA, 1993. USENIX Association.
20. A. Orebaugh, G. Ramirez, J. Burke, and L. Pesce. *Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security)*. Syngress Publishing, 2006.
21. T. Ptacek, T. Newsham, and H. J. Simpson. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, 1998.
22. Z. Wu, M. Xie, and H. Wang. Swift: a fast dynamic packet filter. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 279–292, Berkeley, CA, USA, 2008. USENIX Association.