Technical Report

UCAM-CL-TR-750
ISSN 1476-2986

Number 750

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

AtoZ: an automatic traffic organizer using NetFPGA

Marco Canini, Wei Li, Martin Zadnik,
Andrew W. Moore

May 2009

15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500

http://www.cl.cam.ac.uk/

Technical reports published by the University of Cambridge
Computer Laboratory are freely available via the Internet:

http://www.cl.cam.ac.uk/techreports/

ISSN 1476-2986

AtoZ: an automatic traffic organizer using NetFPGA

Marco Canini, Wei Li, Martin Zadnik, Andrew W. Moore

Abstract

This paper introduces AtoZ, an automatic traffic organizer that provides end-users with control of how their applications use network resources. Such an approach contrasts with the moves of many ISPs towards network-wide application throttling and provider-centric control of an application's network-usage. AtoZ provides seamless per-application traffic-organizing on gigabit links, with minimal packet-delays and no unintended packet drops.

The AtoZ combines the high-speed packet processing of the NetFPGA with an efficient flow-behavior identification method. Currently users can enable AtoZ control over network resources by prohibiting certain applications and controlling the priority of others. We discuss deployment experience and use real traffic to illustrate how such an architecture enables several distinct features: high accuracy, high throughput, minimal delay, and efficient packet labeling — all in a low-cost, robust configuration that works alongside the home or enterprise access-router.

1 Introduction

The evolution of the Internet has encouraged new data-intensive applications to emerge and quickly gain in popularity. Examples include Peer-to-Peer (P2P) file-sharing, online video services (e.g., YouTube), IPTV, wide-area file storage and network-based applications (e.g., Google Docs). The resulting high-rate of Internet traffic growth, 50–60% annually according to MINTS[1], drives the ever-higher Internet bandwidth requirements.

Simple over-provisioning is insufficient and comes with significant costs. The broadband *buildouts* which are expected to take the form of fiber to curb, premises and home[2], appears as part of the fiscal stimulus packages for many countries[3] and will only accelerate the growth. However, the increasing traffic, diversity of the applications and their intolerance to disruptions are posing an unprecedented challenge for the Internet infrastructure. Many business critical or highly interactive applications such as VoIP, financial trading platforms and real-time multi-player games are latency and loss sensitive, and thus prone to starvation caused by the hands of other data-intensive applications.

Several ISPs react by throttling certain traffic (e.g., BitTorrent), which is inappropriate for solving the wide-ranging problems in enterprise, organization, and domestic settings; and moreover triggers network neutrality concerns[4].

[1] `http://www.dtc.umn.edu/mints/igrowth.html`

[2] *High-Speed Internet Grants Find Support in House*, Wall Street Journal, January 22nd, 2009. `http://online.wsj.com/article/SB123265622888307299.html`

[3] `http://www.bloomberg.com/apps/news?pid=20601204\&sid=aJeIvPUk2vAQ`

[4] `http://uk.reuters.com/article/burningIssues/idUKTRE50R6W020090129`

In this paper, we present AtoZ, an automatic traffic organizer which gives control back to the end user: it allows flexible managing, shaping, balancing and distributing the traffic load on an access network based on actual applications. AtoZ is built upon a hybrid hardware-software platform and combines two significant elements:

- a hierarchical packet labeling scheme based on accurate behavioral application identification, and
- an implementation based upon the NetFPGA platform [1] permitting high performance packet processing.

This architecture can handle gigabit line-rates, while providing accurate and efficient labeling of packets, and to have the reliability and seamlessness in packet processing with minimum delay and no unintended packet loss[5]. The system exhibits substantial advantages over operating system-based solutions, such as the Linux netfilter[6], and commercial traffic shaping solutions, such as Qosmos QWork.

Our deployed prototype of AtoZ implements application-specific switching of traffic from different applications to different physical interfaces. We have conducted evaluations using real network traces from a university campus and a research institution to show how specialized hardware for packet-processing has a low impact on performance and complements the high accuracy application-labeling approach.

We begin by motivating our approach, followed by detailing design specifics and the results of our evaluations.

1.1 Motivation

Inefficient use of bandwidth is a significant problem for many enterprise, institution and even home networks. Business-critical traffic in an enterprise network can be congested by activities irrelevant to the core business, while critical applications may not receive even the simplest priority or service guarantees.

Our goal for traffic-organizing is to re-balance, reduce, or restrict specific traffic generated by particular network-applications. In a business scenario this may be motivated by financial common-sense, apportioning the cost of link usage appropriately among particular applications. But these need not imply weighty policies or heavy-handed restrictions. Our approach can dynamically provision network resources as new critical-applications come and go. Perhaps IPTV or data-replication causes unwanted service degradation? By providing a degree of resource-limiting, other applications such as VoIP can be shown to work without resulting in poorer service.

Even in the home context, balancing the uplink bandwidth, a bottleneck on ADSL and other asymmetric deployments common to the home, can provide a huge improvement in collective experience when the P2P clients are throttled permitting the VoIP even just a small quantity of constant capacity.

Figure 1 illustrates our prototype deployments of the AtoZ Automatic Traffic Organizer: as part of the access router. In our prototype deployments, the traffic organizer has been

[5]Intended packet-loss would result if the user configured AtoZ to discard certain applications.
[6]`http://www.netfilter.org/`

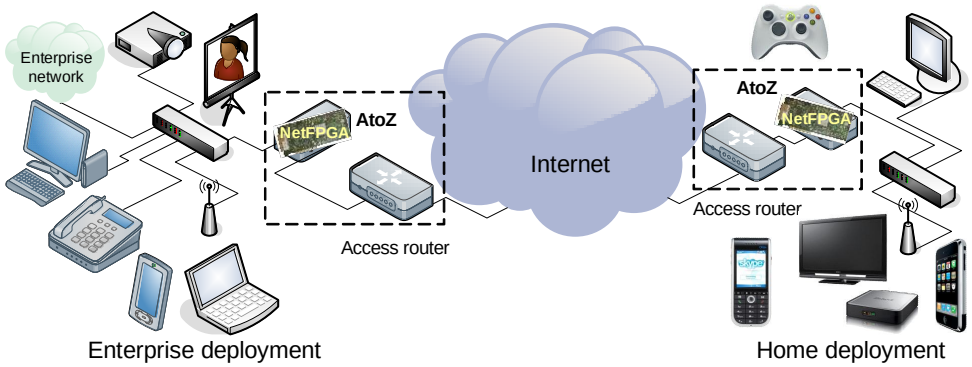

Figure 1: AtoZ in use as part of an enterprise or institutional gateway and as the access-router in a domestic setting.

placed alongside the existing access-routers, in the same manner as NIDSes (Network Intrusion Detection Systems) might be deployed. We foresee deployment of this system on the granularity of *access-policy* such as the department, hall-of-residence or, the increasingly well-connected home. With an access-router approach, the rewards for deployment are enjoyed by those bearing the cost.

We acknowledge specific requirements can differ both subtly and significantly in different scenarios. We focus on these quite different uses and these reflect our deployment experience.

Firstly, *Home Access Links* are commonly shared by family members or room-mates. Despite a comparably small number of simultaneous activities ongoing on the link, the traffic is rarely under effective management. The user experience of certain applications, e.g., online gaming or VoIP are often affected by resource consuming activities, e.g., P2P file-sharing or bulk downloading of video content. AtoZ provides "seamless home happiness" where different activities are neatly organized to optimize and utilization of the link and the overall user experience.

In contrast, *Enterprise*[7] *networks* are often intended to serve, as a priority, specific business goals. These may have clear functional-definitions: emails, electronic conferencing, customer services, or specific network computing needs, yet may be difficult to enumerate as network applications; particularly when practical and effective business work is done via Skype, Windows Live Messenger and other difficult to classify applications. In such an environment stringent policy may exist that embargo certain applications entirely. Our approach provides a combination of robust application identification and flexible control, implementing a range of access-policy: from understanding the (access-link) use, through prioritization of applications to the implementation of outright embargoes on unwanted applications. One outcome of our trial at a hall-of-residence has been a pending policy change permitting several previously banned applications including *Skype*. The reason has been that AtoZ removes the unpredictable impact of such applications upon more critical network services, and a user-pays situation can go some-way to reducing the unexpected cost to users of such applications.

Our approach builds a high performance but cost-effective traffic organizer, which is

[7]We evaluated AtoZ operating in a number of different sized organizations: including SOHO environments, departments, halls-of-residence, and entire research institutions of between 1,000 and 15,000 users.

capable of fulfilling the requirements for the current and next generation network, while being adaptable to different application scenarios.

2 Design Overview

The core component of a traffic organizer is its packet-labeling process, which attaches a label on each packet permitting application-specific management to be done using the label. We acknowledge a range of literature and practical implementations of packet scheduling and management such as the CSFQ [2], which this work may usefully inform. Our focus and main contribution is an architecture that provides accurate input to be used in application-specific management.

Several significant features allow our system to meet all our design objectives. The hybrid architecture (Section 2.3) allows the NetFPGA fast data path to support high-throughput, minimum packet-delay and non packet-loss of the system, while maintaining its intelligent functionalities (e.g., application identification) in software components. The hierarchical packet-labeling (Section 2.2) is combined with accurate application identification (Section 3.3) to maintain the high labeling accuracy while significantly reduces the overall time-to-bind[8] and occupation of the flow cache. Further, the modular structure (Section 2.4) of our system allows flexible updates and extensibility of the system.

We observe that there have been two main kinds of approaches in application-based traffic-shaping systems. The first is based upon a list of known IPs and services, such as the Linux netfilter, and a second approach is based upon individually-identified flows and through the use, for example, of deep-packet inspection (DPI) modules, which is common as part of commercial products (e.g., Qosmos QWork, Blue Coat PacketShaper, Riverbed Steelhead and D-Link Traffic Manager) and NIDSes (e.g., Snort [3], Bro [4]).

The first approach utilizes user-defined IP-based rules stored in the form of a static IP or service table. Despite having the advantage of simplicity, the table represents only a known part of the total traffic, and such a static definition will be error-prone, particularly with changes over time, and may have limited scalability if it is to describe, statically, every possible combination of IP address and service (port).

The second individual flow-based approach has been widely deployed, however, it has a number of architectural disadvantages:

1. a very accurate flow-identification mechanism is a must-have, but standard methods based on pre-declared port numbers or DPI have faced a challenging environment for some time [5],

2. application-specific operations can be taken only after a flow is classified, which introduces a period in which packets are left unlabeled; this makes the mechanism less effective for applications that involve many short, separate connections, and

3. it is non-trivial, introducing a non-negligible overhead to maintain a flow-table, with which each packet is matched to a flow entry in order to identify flow/application membership.

[8]We define time-to-bind as the period between the time a new flow is first observed and the time it is identified.

In contrast, several distinct observations allow us to develop a novel solution that meets all our design objectives – high throughput, high accuracy, swift packet-labeling, minimum delay and zero (unintended) packet loss.

Firstly, for any given traffic, there is only one unique application for a distinct {IP, port, proto.} 3-tuple within a certain time period[9]. We base this assertion on our observation and are confident it holds in most circumstances. This means that *dynamically* learned {IP, port, proto.} rules can be used to accurately label packets at a higher level of aggregation than the standard IP 5-tuple {src IP, src port, dst IP, dst port, proto.} which identifies a flow. This enables the labeling of packets of significant hosts and activities without time-to-bind, while also reducing the cost involved in maintaining the flow table and identifying each flow.

Secondly, past work has shown that behavior-based machine learning models can identify applications with very high accuracy using the first few packets of each flow [6, 7]. This mechanism can provide accurate and timely labeling result, and also be used to create and maintain 3-tuple rules.

Thirdly, the NetFPGA platform: a combination of FPGA and Gigabit Ethernet ports located on a PCI board, provides an ideal environment for implementing a packet-processing fast-path. In this way minimal-delay packet processing can be driven by specialized hardware. It is still difficult to support all the sophisticated requirements of classification within the NetFPGA. However, this allows for a natural division of work between the complex software suited to a general-purpose computer and the NetFPGA which focuses on accelerating the packet-processing pipeline.

This allows us to shape several key features of our design. In the following sections we present the design, discuss the tradeoff and show its advantages over standard approaches. We summarized earlier how we approach these design objectives.

2.1 Service Stability Assumption

Usually, packets with the same {IP, port, proto.} 3-tuple[10] are carrying out a similar service. For example, TCP packets to and from 207.46.107.73:1863 would be associated to the Windows Live Messenger. This observation leads to an assumption that packets associated to the same 3-tuple commonly belong to the same application in a certain time period. And so, most packets can be accurately labeled using a mapping table between the 3-tuple and an application. This can significantly reduce the memory cost of maintaining a mapping table based on the 5-tuple and accelerate the labeling process.

We base our assumption on observations of institutional networks (for several institutions) taken over multi-day periods covering periods since 2003 up until the current day. Within each day, we identify flows through pattern matching and manual inspection of the payload contents. The results show that traffic follows our stability assumption for the entire day although our work does not imply stability over longer periods. The only exceptions that exist are the different applications encapsulated in VPNs, tunnels or SOCKS proxies. In our current implementation these applications are identified as a special class of their own.

[9] We acknowledge the application-multiplexing that may occur within certain ports (80) and within services such as VPNs, these are specifically dealt with in Section 2.1.

[10] In either the server or client side.

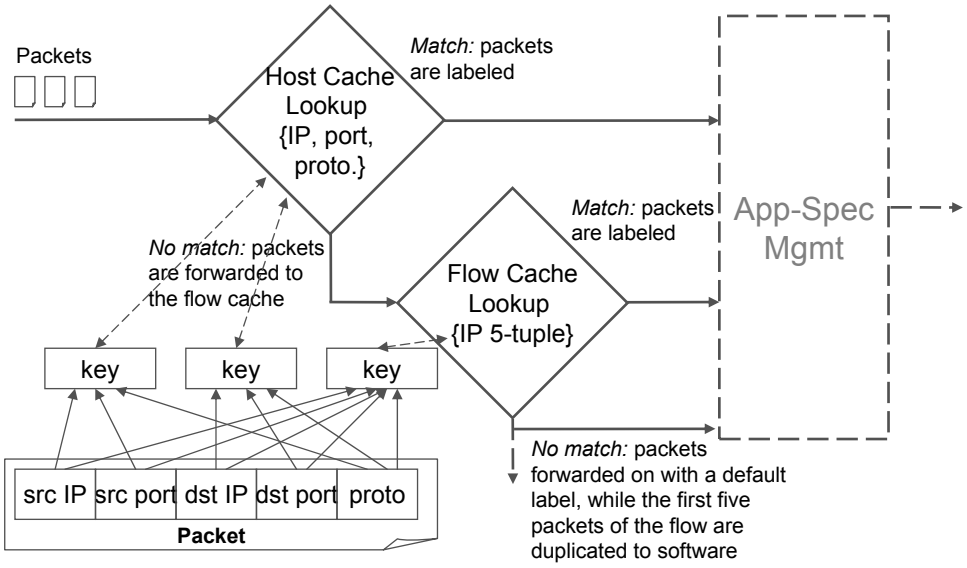

Figure 2: Hierarchical Packet Labeling

Others have discussed further mechanisms that can be applied to identify the encapsulated applications as needed [8].

2.2 Hierarchical Packet Labeling

The packet labeling in our system, illustrated in Figure 2, is hierarchically structured: it comprises of the Host Cache (HC) working at the host and service level, and the Flow Cache (FC) working at the flow level.

The HC stores entries that contain dynamically learned rules to label the packets associated to significant hosts and service activities ongoing on the link, which map an {IP, port, proto.} 3-tuple to an application class.

Packets not labeled by the HC are handled by the FC, which implements the mapping between a flow (identified by its 5-tuple) and an application class.

Finally, all labeled and unlabeled packets are forwarded for application-specific management.

The application-class labels in both caches are automatically learned. Specifically, the HC is dynamically updated by inferring the binding of an application to a certain {IP, port, proto.} 3-tuple via aggregation of the application information of flows derived from early flow application identification. Once the association for a 3-tuple has been established, subsequent packets carrying that 3-tuple can be immediately labeled at this higher aggregation. This means that *new flows* involving a *known 3-tuple* are handled without requiring to track and identify them; and their the time-to-bind is reduced to zero.

In common with past-authors [9], we capitalize on the observation that host interactions on the Internet exhibit a heavy-tail behavior, that is, the majority of the traffic is associated to only a small proportion of the interacting hosts [10, 11].

The HC is intended to deal with such heavy tail – the services most frequently used or with the heaviest traffic load. Clearly, the existence of the HC also effectively reduces the

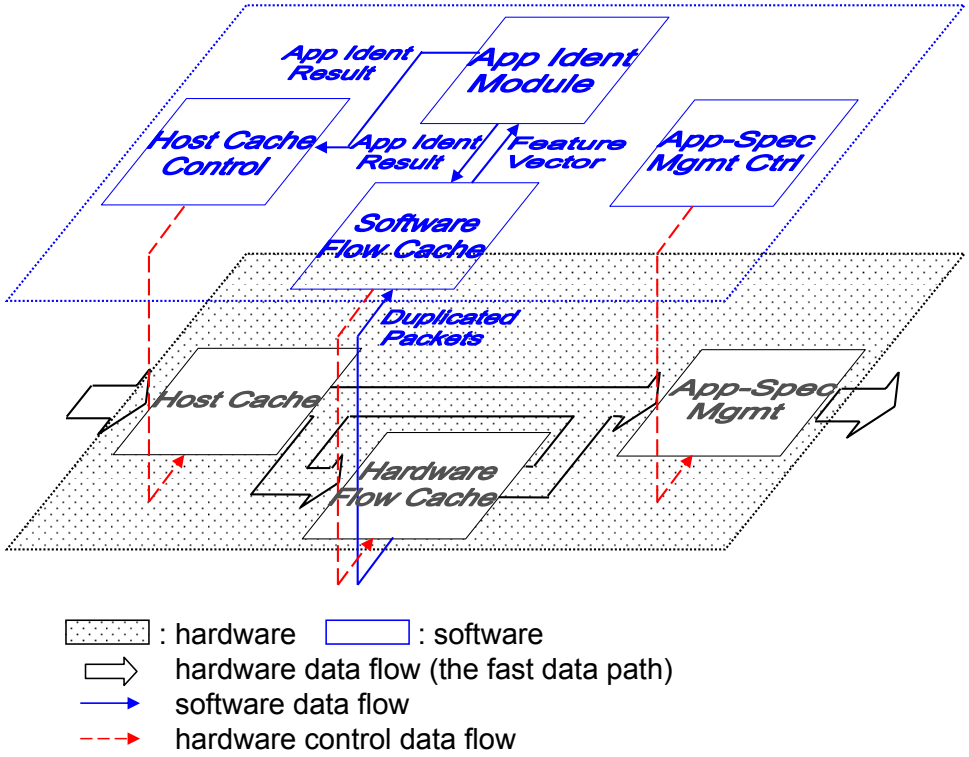

Figure 3: The hybrid architecture and data flow in the traffic organizer.

utilization of flow records in the FC, since a significant number of flows may be handled by fewer HC entries; otherwise, each flow will result in one FC entry.

However, the HC does not label packets carrying 3-tuples that have not appeared before and for which an association has not been established. Therefore, we complement it with the FC in order to guarantee that the rest of traffic is also timely labeled. In common with traditional flow-monitoring mechanisms (e.g., Cisco NetFlow), the FC supports the maintenance of per-flow state information, but here it only handles traffic which has not already been organized by the HC.

The application-identification module provides accurate flow identification to initiate instant updates of the FC, and updates the HC at certain time intervals as further discussed in Section 3.1. The module adopts a light-weighted behavior-based early-flow identification method [7]. The method focuses upon the beginning of flows, enabling application identification to be done promptly after the flow is observed.

Our resulting hierarchical packet labeling process is able to maintain labeling accuracy of the application identification mechanism. More significantly, it removes the time-to-bind of up to 79% of the flows and saves up to 75% the FC occupation.

2.3 Hybrid Architecture

The NetFPGA has meant we are able to implement the data path for packet processing at full-duplex Gigabit line rates with minimal processing latency measured in only a few clock cycles [1]. Its capabilities are only limited by the complexity of the logic and the size of available memory; therefore, our system adopts a hybrid architecture that integrates

the NetFPGA as a fast-path packet processor with a more comprehensive, albeit slower, software slow-path executed on a host machine to overcome memory and complexity limitations.

Several design challenges exist for such a hybrid architecture. The first challenge is the gap between sophisticated, dynamic, organizing-functions and heavily constrained complexity and memory availability in hardware. The second challenge is to be disruption-free: that is, to minimize the impact on performance, at speed, latency and (unintended) packet-loss. The third challenge is to find the right balance between the high-speed hardware and the lower speed software. Meanwhile, the communication between hardware and software is limited, operating at rates far slower than the total fast-path throughput and removing any opportunities for the simple off-loading of hardware-workloads into the general-purpose software systems. One final challenge is that the hardware ought allow dynamic rule updates, of the FC and the HC, on-the-fly.

These challenges are resolved by properly designing the data paths and distributing the processing tasks between hardware and software components, while considering the following design principles:

- build a complete, uninterrupted packet processing data path in hardware (which only duplicates packets to software as it is able to),
- minimize transfers between NetFPGA and host machine,
- limit the complexity of gateware, the synthesized logic downloaded into an FPGA,
- reduce memory usage in hardware, and
- modularize by function (not by process) each system component to preserve the flexibility of various traffic organizing sub-systems, permit mobility between hardware and software module implementations and allow for future extensions of the system.

All the core packet processing components are implemented in hardware, including the packet labeling process based on the HC and FC hierarchy, the Application-specific Management logic, and the packet-receive and transmit units. These form the fast data path of our system, from receiving packets from an interface to delivering them to where they should be organized. Currently, our prototype implements modification of the destination MAC address and-or switches traffic on different physical interfaces. Extensions to traffic-shaping (e.g., rate-limiting) for specific applications is also available.

The components not located on the fast data path are implemented in software. These are the software flow cache, the application identification module, the hardware controllers which dynamically update and maintain the caches, and further extensible modules that allow for other specific operations.

Moreover, the computational cost in this architecture can be significantly reduced by limiting the number of packets to be processed in software. Packets having been matched by the HC or the FC do not need to be processed again by the software. And so, we design a packet filter in hardware that forwards only the first few (typically five) packets of each flow to the software, so that a fairly small proportion of total packets will go through the software data path.

Between the hardware and software, data-driven communications permit the system to work smoothly and efficiently. For example, immediately after identifying a flow, the software forces an update of the corresponding hardware FC entry on-the-fly.

The packet filter combined with the de-synchronization between hardware and software guarantees that the processing capability of software components will not be a bottleneck for the entire system. In this way, the whole system is able to work at a constant high speed: it handles up to 8 Mpps with an average latency of less than 17 μs.

2.4 Modular Structure

Owing much to the modular *vision* of the Click Modular Router [12], we implement our system as functional modules. Each software and hardware component in our system can be flexibly reorganized and easily tuned, in order to (a) facilitate further research, (b) allow using alternative algorithms to achieve specific tasks or to optimize performance, and (c) to enable easy, frequent update of identification models or traffic organizing policies due to novel development of Internet applications.

For example, the application identification module can be replaced, such that any other machine-learned early-flow identification models or DPI methods can be used, or even a combined classifier using multiple methods. Host-based classification models, for example BLINC [13], can also be used alongside the flow identification results to derive HC rules.

The modularity also allows re-assembling the pipeline for many more purposes than in this prototype. This allows the system to be extended to other applications, which are further discussed in Section 6.

3 System Component Design

The traffic organizer is comprised of several modules, as shown in Figure 3. The modules in the system may combine codes on two different platforms: gateware on the NetFPGA card and software on the host machine. For convenience we refer to them as hardware components and software components.

Hardware components focus on supporting a fast data path providing minimal packet-processing delays and no unintended packet loss. Accordingly, software components are designed for two objectives: firstly, to support sophisticated functions such as rule creation and flow identification, and secondly, to manage and control the hardware components where the data structures have a higher complexity than that can be managed by themselves.

In the following subsections, we describe a system design which is based upon several functional modules. We show how the modules connect with each other, and how software and hardware components collaborate in each module to achieve the design goals. Here we focus on the high-level functional design and the software implementation, while hardware design and implementation is further discussed in Section 4.

3.1 Host Cache

As mentioned in Section 2, by focusing upon the subset of hosts which are responsible for large amounts of traffic flows, the Host Cache serves to label packets to and from these hosts with the corresponding application class.

HC is a two-stage mechanism, comprising a fast packet labeling stage in hardware, and a management stage in software. The packet labeling stage operates at the per-packet level, matching three packet header fields, either {dst IP, dst port, proto.} or {src IP, src port, proto.} against a set of existing rules based on {IP, port, proto.} 3-tuples. If it finds a match, the packet is labeled using the application class associated with the matching rule; otherwise it retains a default empty label.

These rules are created and managed by the HC management stage. This operates by defining a time window (and thus the time scale is several orders of magnitude larger than the per-packet level) during which the newly classified flows are aggregated by the server end-point: the {IP, port, proto.} 3-tuple based on the server address. Now, to each end-point corresponds a flow count x and a set of application classes. Let p be the proportion of flows belonging to the class with the majority of memberships.

Note that only flows starting after the beginning of the measurement interval are counted. Furthermore, only flows that are not matched by the HC rules currently in use are considered because the matched flows are not being processed by the application-identification module.

At the end of each time window, we score each candidate rule using a utility metric, and we consider just those rules whose utility exceeds a threshold U_{th}. If the total number of rules is larger than the HC capacity, the ones with higher utility are selected. For a given end-point, we define the *utility* metric as

$$U = x \times f(p)$$

where $f(p)$ is a function increasing with p[11]. The idea here is that the utility is proportional to the flow count and is also affected by the consistency of the application identification results. For active rules, we estimate their utility as $U = n$ where n is the number of flows matched by this rule during the past time window. As the active rules are stored in hardware memory, it is the hardware that measures n (approximately).

Finally, the management stage identifies the most useful rules by comparing the candidate with the active rules. Active rules that are not considered sufficiently useful are evicted and replaced by more useful ones.

The utility score serves to describe the usefulness of a rule and by quantifying the contribution of the candidate rule we provide a mechanism for selecting the entries for eviction. Using the utility metric in this way allows us to directly trade uncertainty in classification-stability (i.e., the applicability of previous classifications to future traffic) for speed-gains made through use of the HC. Recall that if an entry is evicted from the HC then the full-classification path is used.

We now discuss some details of the complexity encountered in our implementation.

[11]The $f(p)$ in this prototype is a linear function, and further investigation is planned in the future work.

Firstly, the initial aggregation of identified flows requires us to keep state information about the interacting hosts. However, as we are only interested in relatively popular end-points, we want to avoid wasting memory tracking every single host's activities. Thus, the first step of the aggregation involves a particular technique based on multistage filters to efficiently detect the heavy-hitter end-points (in terms of flow count). A detailed description of multistage filters is offered in [14], but in essence, a multistage filter is a variation of counting Bloom filter [15] that represents a subset of elements and their individual occurrences with just a small, fixed-size memory block. This technique effectively reduces the memory consumption of the management stage because it only has to track the state for the small set of popular end-points that pass the filter, i.e., their flow count is above a certain threshold.

The management stage then maintains a hash-table that associates each popular end-point with an array of counters of identified flows in each class. The table supports an `update(`f`)` operation that is invoked upon the identification of a flow f involving an end-point that passes the filter. When invoked, it increments the counter corresponding to f's class for the table entry indexed by f's 3-tuple (the end-point).

The filters and hash-tables are reset at the beginning of each time window.

Secondly, the estimation of the active rules' utility requires that some statistics are kept for each rule in hardware. Since we want a low gateware complexity, we use an implementation based on Bloom filters to arbitrarily trade-off estimation accuracy for space-efficiency [15].

3.2 Flow Cache

In our system, flow level operations are supported by a dual flow cache, combining two data structures: hardware Flow Cache (HFC) and software Flow Cache (SFC), located on the NetFPGA and on the host machine, respectively. The role of the dual flow cache is to maintain flow state information for currently active flows (inactive flows are identified using a timeout). Indeed, this is similar to standard flow monitoring mechanisms such as Cisco's NetFlow. However, some notable differences remain.

Firstly, an important difference is that such dual flow cache is specifically designed for traffic organizing instead of traffic measurements (Section 6 discusses how traffic measurements can be made in our system). Thus, the HFC only observes and processes packets which are not matched by the HC. Furthermore, the SFC processes an even smaller proportion of the packets than the hardware counterpart, typically the first five packets of each flow, according to the principle that the hardware supports larger throughput than software.

Secondly, the HFC and the SFC are bi-directional, i.e., there is only one entry for packets of the same flow in both directions.

Thirdly, the HFC is devoted to packet labeling, and maintains minimal information about the flows (stored in a space-efficient format). On the other hand, the SFC supports the flow identification by collecting the flow features.

Furthermore, the HFC is not self-managed. Instead, its management operations are driven by software, with the hardware providing an interface to support these operations. However, all the communications are data driven and de-synchronized, in order to minimize the latency and overhead. For instance, the software updates the application class of a

certain flow once it is classified, and manages the flow timeout[12]. Finally, it removes the FC entry when a flow expires.

Importantly, stricter hardware constraints require that the HFC has smaller capacity than the SFC. When changes in the traffic patterns cause an increase in the number of flows leading to a saturation of its capacity (e.g., during DoS attacks), the HFC is not able to label continuously all packets for the set of identified flows in the SFC. As described in Section 4.3, we include a simple LRU-based eviction policy to maintain the most active flows in the HFC. In such situations, counter-measures such as elephant flow detection [14] or reducing the flow timeout can overcome the impact of such an attack with the drawback of (potentially) increased inaccuracy in classification performance.

3.3 Application Identification

As already noted, the hierarchical packet labeling mechanism is supported by the early-flow application-identification module.

Here, we implemented the method described in [7] in C++. It utilizes 12 behavioral features and C4.5 decision tree algorithm to build machine-learned models that are able to classify traffic into 13 application classes with both high accuracy and low computational overhead.

The 13 classes group applications based on their purpose, and are intended to cover all the Internet traffic, with each application categorized into one of the classes (e.g., Windows Live messenger being classified into "CHAT"). Such definition tends to capture the inherent property of each type of application and may exhibit better stability over time than individual protocols or applications.

The method supports both TCP and UDP traffic; furthermore, the early-flow features used in this method are collected from the packet headers of no more than the first five packets of each flow, allowing a short time-to-bind.

In [7], the authors show that the models are fairly accurate over a considerably long period of time, and across different sites; they also suggest that a model built up with traffic from various sites can be more accurate in overall for all sites. Notably, the worst result (classifying traffic using a model from a different site 1 year ago) using the classifier is still better than conventional DPI and port number-based approaches.

Besides the accuracy, the method also benefits from the extremely low complexity of the C4.5 decision tree classifier [16], the implementation comprises of several nested `if` statements. This makes it a very efficient flow classifier that is well-suited to handling online traffic.

4 Hardware Design

While the software system provides the control plane of the whole system, the hardware implements the entire packet data path on a NetFPGA card. The gateware in the NetF-

[12]The flow timeout is managed in software only to maintain a simple hardware logic. When a flow times out, the software resets the packet filtering counter to test whether the flow is still active. If no packets are observed for this flow during another timeout period, the flow is considered expired.

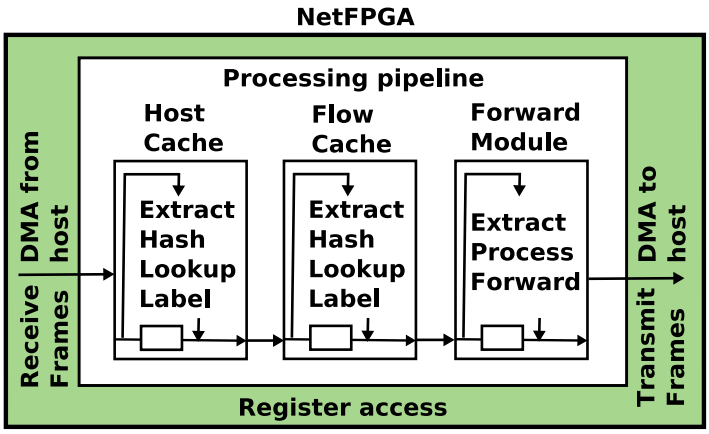

Figure 4: FPGA gateware – Fast data pipeline and the NetFPGA environment.

PGA is described in Verilog[13] and can be easily ported to any network interface card equipped with FPGA chips.

4.1 The Fast Packet Data Path

The fast data path consists of a modular pipeline with multiple modules, as shown in Figure 4. All hardware modules are data-driven: each processes frames (Ethernet encapsulated packets) immediately upon arrival and independently of any other module. Modules receive and transfer data using a 64-bit wide data bus with a FIFO-like interface, and are designed to process up to 8 Mpps which is sufficient to handle more than four fully utilized Gigabit interfaces irrespective of packet sizes.

As can be seen in Figure 4, while running independently, each module consists of several actions including extraction of packet header fields, hash computation, table lookup, and packet labeling or forwarding.

The life of a packet in the pipeline begins in the HC. Firstly, the IP addresses, ports and protocol are extracted from the packet. Then two hash values are computed, the first using {src IP, src port, proto.} and the second using {dst IP, dst port, proto.}. These hash values are used to look up the application class label in the HC table. The mechanics of the hash-table is discussed in detail in Section 4.3. If the lookup is successful, then the frame header is annotated with the class label. In this case, the frame passes the HFC without any further processing. Otherwise, the HFC extracts the IP 5-tuple and computes a single hash value (since we consider bi-directional flows) which is used to look up the flow record in the HFC table. If found, the flow record contains the class label and a packet counter. If the label is valid, the frame header is annotated with it. The counter is only used to keep track of the first few packets for this flow (typically 5) so that they can be duplicated to software. When the counter reaches its maximum, aside from TCP packets with FIN or RST flags set, no more packets for this flow are duplicated to software. However, the software can reset the counter when required (e.g., to handle a flow-timeout). When the lookup fails, a new flow entry is created in the HFC table, with counter of zero and an empty class label. The class label is updated as soon as the new flow is identified.

[13] `http://www.verilog.com/`

Both labeled and unlabeled frames are forwarded for application-specific management. In our implementation, this is done by the Forward Module which is capable of forwarding traffic to different output ports and/or to modify destination MAC addresses based on the application class. It uses a uniform interface to implement both the packet transmission and packet duplication to the SFC on the host machine. Non-IP packets, such as ARP and ICMP are simply forwarded to the default output port.

The implementation of this module can vary depending on the desired functionality. Examples include priority queues to accelerate critical traffic, filters and rate-limiters to throttle undesired traffic, intelligent load balancers or specific modules that apply different Service Level Agreements to different application classes.

4.2 Hardware – Software Communication

The fast data path and software components cooperate to organize traffic at a high speed. Due to its limited complexity the hardware utilizes results from software to accurately label incoming packets, and at the same time it realizes load shedding for the software by handling the majority of the traffic.

Hardware and software sub-systems communicate via the PCI interconnect.

From the fast data path perspective the NetFPGA platform provides an interface to receive or transmit Ethernet frames on physical interfaces or to transfer frames using DMA operations in either directions between the card and the host machine. In addition, it provides a register-based interface which is used to receive the control commands.

From the software perspective the interface to receive the duplicated packets is provided by a Linux kernel driver which exposes the NetFPGA to the operating system as a standard NIC. Further, the software components have access to memory-mapped registers that are used for both configuring the data pipeline and updating the HC and HFC tables.

4.3 Hash Table Lookup Scheme

The lookup scheme in both the HC and HFC is based on a Naïve Hash Table (NHT) that consists of an array of buckets each of which contains a fixed number of entries. We are aware of other schemes, such as those based on Bloom filters [17, 18, 14], but NHT is readily mapped on FPGA on-chip memory structure and inherently has characteristics that other schemes might miss or for which it is hard to achieve: NHT allows to (a) add and remove items quickly and continuously, and (b) store and update per-entry state information.

The lookup procedure works as follow. Given an item to be searched, we compute an hash value and use this value to select the corresponding bucket. Then, instead of doing a sequential scan through the bucket's entries, we search all entries simultaneously by exploiting a function of the Xilinx Virtex-II-Pro FPGA which allows parallel access to multiple banks of on-chip memory in one clock cycle. Also at no extra hardware or time cost, the entries in each bucket are kept in least recently used (LRU) order so that when a bucket becomes full we can maintain the most active flows and evict old entries to allow new items in the bucket.

Parameters	HC	HFC
Entries	2048	32768
Buckets	256	4096
Fingerprint	40	36
Length of hash	48	48
F.p. probability p_f	7.5×10^{-9}	1.9×10^{-6}

Table 1: Configuration of Host Cache (HC) and Hardware Flow Cache (HFC)

As both tables are stored in the on-chip memory, space is a major limiting factor. At the expense of accuracy, we store in each entry only a small fingerprint of the original item instead of its complete descriptor (e.g., the IP 5-tuple). This greatly increases the number of entries, but it introduces the possibility that two different items (e.g., flows) collide into the same entry if they have the same hash value and fingerprint. This undesired situation is called a false positive. However, by dimensioning the bit-length b of the fingerprint, the probability of false positives p_f can be reduced to an acceptable rate. This probability can be estimated by the birthday problem. Suppose there are 2^h buckets, then the bit-length of an hash value is $h + b$, therefore we have:

$$p_f = 1 - \frac{m!}{m^n(m-n)!} = 1 - \frac{2^{(h+b)}!}{2^{(h+b)^n}(2^{(h+b)}-n)!}$$

$$\approx 1 - e^{-\frac{(n-1)n}{2\times 2^{(h+b)}}}$$

where m is the total number of hash values and n is the number of entries in use.

4.4 Hardware Setup

The total on-chip memory available in NetFPGA would allow us to accommodate approximately 128 K entries if each entry has a 36 bit fingerprint and a few bits reserved for the application class and the packet counter. Unfortunately, the NetFPGA infrastructure (MAC cores, etc.) consumes over a half of the memory resources. In our prototype, we use the configuration of the HC and the HFC as shown in Table 1. Each bucket in the HC and the HFC contains 8 entries; with a total length of 48 bits for the hash, the false-positive probability of the NHT is negligible for our experiments, as shown in the table.

We are planning to use the external SSRAM memory available on the NetFPGA board which is currently used for packet buffers and queues. We estimate it can accommodate approximately 500 K entries which will make it more suitable for deployment in larger enterprises.

Moreover, we show that our approach works sufficiently well with these limited numbers of table-entries even when contrasted with the total number of flows present in our evaluation traces. The HC effectively labels a significant number of flows and the HFC gracefully degrades the labeling performance when the number of concurrent flows exceed its capacity. In no case packets are dropped.

Trace	Dur.	Packets	Bytes	Avg. Util.	Avg. Concur. Flows
R.Inst.	30m	202 M	142 G	624 Mbps	38 K
Campus	30m	268 M	22 G	98 Mbps	17 K

Table 2: Working dimensions of our traces.

5 Performance Evaluation

The experiments aim at validating our hierarchical packet labeling approach, and demonstrating the performance advantages of simple and cost-effective hardware.

5.1 Experimental Setup

For this system, the packet and flow rate have more significance than just the data rate. The data path is designed to handle multi-Gigabit/s data streams. However, each packet and flow undergoes a number of operations (e.g., table lookup, feature extraction, classification) that must be able to keep up with packet and flow rates, respectively, so it is important to include realistic traffic in the evaluation methodology. Moreover, as AtoZ derives and uses information of behavior of applications we conclude that real traffic traces (with pre-determined ground truth) are fundamental to the system evaluation.

For these experiments, we use two data traces of real traffic, one from a research institute (R.Inst.), and one from a large university campus (Campus). Table 2 summarizes the working dimensions of our traces.

Our testbed consists of two high-end server machines which can replay the traffic using `tcpreplay`[14]. We partitioned the traces into two halves, using flow hashing to ensure that packets belonging to the same flow are in the same trace, to allow them being replayed from the two machines with fidelity. The traffic from the two machines is aggregated in a Cisco C6509 switch to provide high data-rate traffic. The NetFPGA is connected to an optical port on the switch via a media converter. By means of an optical splitter located on the path to the optical port, we also connect a DAG card downstream from the switch so that measurements are not impacted by jitter and delay introduced at aggregation. `tcpreplay` provided support to reproduce the timings of the original trace with sufficient fidelity.

The NetFPGA board is hosted on a machine with an Intel Quad Core CPU (2.40 GHz) with 4 GB of RAM, running CentOS Linux 5.0 (kernel version 2.6.18). However, our software implementation is only single threaded.

In order to compare AtoZ with an equivalent software-only solution, we combine our software components with elements from the standard Click distribution that provide packet-receive and transmit functionalities. In this case, we run the software on the same PC hosting the NetFPGA, but we use a high-end dual port NIC (Intel e1000) to receive and retransmit packets, because the NetFPGA is not optimized to operate as a standard NIC.

We point out that our prototype uses research-grade code and many opportunities exist for further improvements to be made. We simply intend to show the performance gap

[14] http://tcpreplay.synfin.net/

Data rate (Packet rate)	300 Mbps (51 Kpps)	600 Mbps (107 Kpps)	1 Gbps (137 Kpps)
AtoZ	13 ± 4 μs	16 ± 3 μs	17 ± 2 μs
SW only	304 ± 120 μs	12 ± 25 ms	-

Table 3: Packet processing latency of AtoZ compared to software-only solution.

between using a software-only solution and a system that builds on that same software but combines with functionalities implemented in specialized hardware.

5.2 System Capacity

We start by examining whether AtoZ is able to handle a fully loaded link, as our simulation predicted. To test the system's throughput, we generate traffic by replaying the R.Inst. traffic traces at maximum speed. This produces a 1 Gbps data stream. Our experiments confirm that no packets are dropped, even when traffic is sent at maximum speed. We measure the throughput using 10 s intervals. The mean throughput is about 137 Kpps and achieves a maximum of 173 Kpps, while the software processes on average 1.6 Kpps with a peak of 12 Kpps (duplicated packets).

5.3 System Latency and Packet Loss

We simulate each hardware component to measure its processing time, using minimum and maximum IP packet lengths of 64 and 1500 bytes respectively. Summing the individual contributions, we obtain 227 and 281 clock cycles which correspond to about 1.8 and 2.3 μs for the minimum and maximum packet length, respectively, excluding buffering and transmission delays. At the expense of complexity, these times can be improved by increasing the clock frequency[15].

To measure the actual latency, we generate traffic by replaying the traces at 300 Mbps, 600 Mbps and maximum speed. Table 3 shows the measured processing latencies with standard deviations (which do include buffering and transmission delays). AtoZ maintains a very low latency (at worst 17 μs) with no observed packet loss and CPU utilization always below 6%. The software sustains a data rate of 300 Mbps, but its processing latency is already an order of magnitude higher than AtoZ. At 600 Mbps, the software drops about 4% of the packets and the CPU reaches 100% utilization, causing an increase in latency of tens of milliseconds. As packet loss already starts to occur, there is no need to test the software with 1 Gbps.

5.4 Packet Labeling

For these experiments, we replayed the traffic traces at their original speed.

Host Cache Efficiency: We evaluate the Host Cache (HC) efficiency by measuring the number of flows that it matches over the total number of flows in each time window. We

[15]The NetFPGA has a core clock that runs at 125 MHz, but modern FPGA architectures readily support 200 MHz.

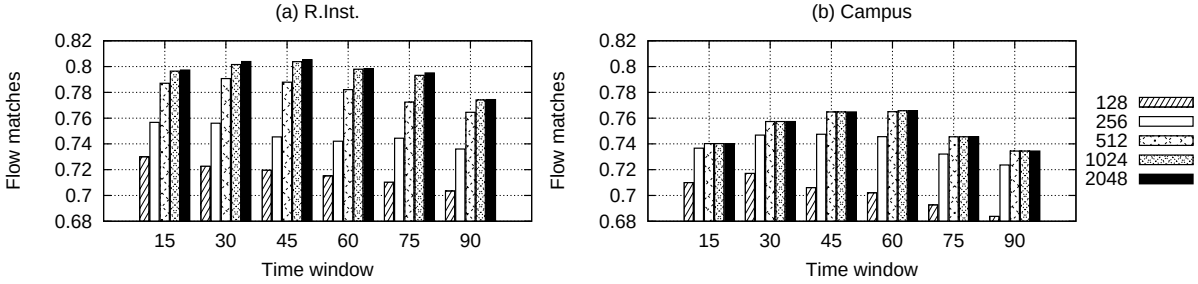

Figure 5: Host Cache efficiency.

vary the time window from 15 to 90 s, and we vary the capacity from 128 to 2048 entries by powers of two. Figure 5 shows the average efficiency for some significant time window lengths. The value of U_{th} is 10. We experimented with several values and we found that this value is a sufficient trade-off between avoiding spurious rules (which happen with lower values) and requiring longer time windows.

As expected, the number of flows matched by the HC increases with a higher HC capacity. The optimum value of the time window is 30 to 45 s for both traces to reach the highest number of matches. Although a longer time window allows establishing with higher confidence which end-points are the most significant and stable over time, extending the time window for too long also reduces the opportunity for applying the rules just derived.

Time-to-bind: An important benefit of the HC is that the system can promptly organize the traffic of new flows involving known hosts: the time-to-bind is zero for all new flows matched by the HC, which means that their packets are labeled from the time the flows start. To demonstrate this improvement, we measure the time-to-bind when the HC is disabled and compare it with the case when it is enabled. Figure 6 shows the time-to-bind histogram obtained for the R.Inst. trace.

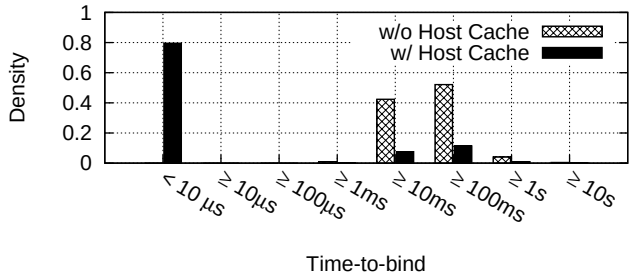

Figure 6: Time-to-bind distribution, with Host Cache vs. without Host Cache.

Labeling Accuracy: By comparing with pre-determined application ground-truth, we evaluate the overall accuracy of the packet labeling process.

Two metrics are considered: the flow accuracy and the packet labeling accuracy. To measure the flow accuracy we export the application identification results and HC rules on the host. For the packet labeling accuracy, we use the DAG card to capture only the packets that are not labeled. By combining this with the results above, we determine which packets are correctly labeled. We compare our labeling accuracy with a number of methods, including the original behavioral method [7], IANA port-based identification, the l7-filter[16] and Qosmos QWork, considering all the TCP and UDP traffic. Table 4

[16]Application Layer Packet Classifier for Linux. `http://l7-filter.sourceforge.net`.

Trace	Metric	AtoZ	Original	Port-based	l7-filter	QWork
R.Inst.	Flows [%]	99.63	99.73	64.66	73.03	86.88
	Packets [%]	99.46	99.71	21.07	27.34	46.39
Campus	Flows [%]	99.59	99.66	89.67	94.28	95.04
	Packets [%]	99.36	99.50	79.39	86.61	87.92

Table 4: Overall packet labeling accuracy.

summarizes the results. (The packet-accuracy for all the other methods is the *gross* accuracy without considering the time-to-bind.) It shows that our approach maintains the high accuracy of the application identification method and outperforms other methods.

Load Shedding: We track the packet rate in the HC, HFC and SFC to evaluate the workload distribution in the hierarchical labeling process. Using the R.Inst. trace, Figure 7 shows the packet processing rate at several stages: the total traffic load, the traffic load of HFC and the traffic load of SFC during its steady-state operation. The HC sustains the entire traffic load and labels about a third of the total packets. The remaining two thirds are mostly processed by the HFC, whereas the SFC receives only a very small fraction of the load. It can be seen that the SFC curve closely resembles that of the traffic load except for the point marked (1) in the figure. This corresponds to the expiration of a certain HC entry due to timeout. As desired, the system periodically expires HC entries even though they are well performing. This is done to maintain accurate traffic classification. In this case, after one time window the HC management restores the HC entry.

Further, the packet filter in HFC combines with the HC effectively to reduce the traffic processed in software to about 0.1% of the original traffic. This result agrees with the findings of other works that also leverage the heavy-tailed nature of traffic. For instance, in [9] the authors reduce the data to less than 6% of the original traffic from their sites by collecting the first 15 KB of each flow. In our experiments, the reduction is even more drastic due to the use of just the first 5 packets and the efficiency of the HC. In this case, disabling the HC increases by 5 times the amount of traffic processed in software.

Figure 8 presents the temporal evolution of the SFC size for R.Inst. We compare the case of HC disabled with HC enabled. When the HC is disabled, every active flow is present in the SFC, causing greater load on the system (e.g., longer table lookup times) and high utilization of the (limited) HFC resources. In this case, the SFC maintains on average 38.5×10^3 entries which is 17% larger than the HFC. However, when the HC is enabled only about 25% of the active flows need to be maintained in the SFC, reaching a maximum of 25×10^3 entries and less than 77% utilization of HFC.

The two curves have similar profiles except for point marked (2) in the figure. Here, in correspondence with (1), the number of flows entering the SFC increases. These are the flows that are matched by the HC until (1), but when the HC entry is removed, the successive 5 packets for these flows are processed by software. AtoZ now has two alternatives: it can label these flows based on the HC entry or it can classify the flows using mid-flow features. In our implementation, we chose the simpler solution which is to label the flows using the HC entry. Therefore, once the HC entry is removed from the HC we still keep it in software for one flow timeout period.

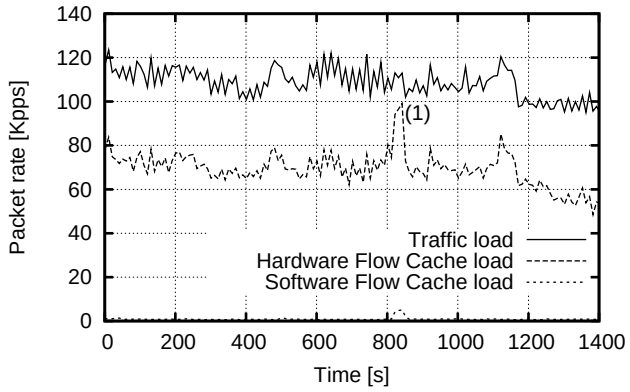

Figure 7: Load shedding of offered traffic load by virtue of HC and packet filtering in HFC.

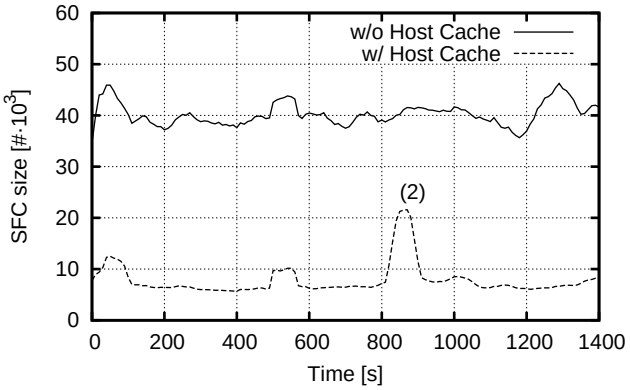

Figure 8: Temporal evolution of SFC size, with HC vs. without HC.

6 Discussion

Robustness

The robustness of the whole system is guaranteed by the hardware fast data path, in which each component is designed to support full line-rate packet switching.

System failures may impact other traffic organizer implementations due to resource exhaustion or system-overloading. In our system, there is no interruption to the fast data path which serves as the default path for all data, so that even in exceptional circumstance only the software system can be overloaded. The worst case is that not every packet is labeled. We could conjecture that the resulting performance will still be an improvement over no device, since at least some traffic labeling would take place. In common with many high-end network device designs, e.g., NIDS[19], the hierarchical packet labeling scheme also provides more resilience against attacks which lead to poisoning of the flow cache.

Quality of Experience (QoE)

QoE is part of our motivation and our prototype can be naturally extended into a QoE device. Space limitations prevent discussing this specific topic in depth, but simple approaches include applying a rate limiter to the lower-priority application classes, or apply-

ing a different packet filter for each class. Further, our device is able to provide a flexible classification of network-traffic for the benefit of other systems that can provide QoE such as diffserv [20] and MPLS [21].

Extensibility

The traffic organizer presented uses only automatically-generated rules learned from the flow classifier. However, thanks to its modular structure, it is readily extensible to allow input information from other sources such as an organization's NIDS or input from a manual interface, or to provide network utilization information to specific traffic-handling systems such as firewalls and NIDS implementations.

Implementation and Costs

Our current software implementation is based on extensions to Click and makes extensive use of the NetFPGA platform, which permits rapid prototyping of our system and demonstrates the feasibility of the hybrid architecture to fulfill the design objectives including packet processing performance, accuracy and flexibility.

For future implementations, one tempting approach is to extend the Linux netfilter to use the NetFGPA by adding hardware implementations of appropriate functional modules, as we have done in the AtoZ traffic organizer. This would improve the performance and efficiency of netfilter in the same manner as the NetFPGA-based acceleration for the standard Linux router[17].

We suggested earlier that the AtoZ traffic organizer be part-of or used alongside the access-router. The advantages of traffic organization are immediately provided to those who incur the deployment costs. For an academic target-audience the NetFPGA card cost is low, at $580. Further, the implementation we have made is portable and may be ported to existing systems (e.g., Netronome[18] and NetCOPE[19]), to provide 10 Gbps implementations at reasonable cost.

7 Related Work

Our work on the AtoZ automatic traffic organizer relates to a wide set of fields ranging from systems-architectures to application-identification based upon machine-learning methods.

We are indebted to the creators of the Click modular route [12], and extend this idea with an approach that preserves functionality between modules. While this does not permit the same level of *plug'n'play* that Click provides, our approach still permits high-performance hardware modules to be co-designed alongside software implementations.

An important theme, and one we share with several past works is the the placement of network functionality. Aside from the obvious example of the NetFPGA itself [1],

[17] http://netfpga.org/wordpress/?p=124
[18] Netronome. `http://www.netronome.com/`.
[19] NetCOPE. `http://www.liberouter.org/netcope/index.php`.

the work of U-net [22], SPINE [23] and ARSEnic [24], — while focused specifically on network interface cards — each provides valuable insight into the choices that are faced in hardware-software co-design.

Issues of co-design were also faced by the designers and users of the PCI-based Pamette FPGA board from the DEC Systems Research Center. A general-function system, this board saw use as a (reprogrammable) engine for graphics offloading, encryption and performance-instrumentation [25].

The architecture of AtoZ shares a few common characteristics with some other FPGA-based architectures that "shed load" for network-intrusion detection and prevention systems ([19, 26, 27]), in particular, the way that it distributes the workload and functionalities between hardware and software. These approaches are specifically designed to offload the subset of traffic which is large in volume but of little interest to intrusion detection and prevention systems. In contrast, AtoZ aims at organizing the entire traffic on the link, and solves different problems that arise associated with traffic organizing. However, due to the architectural similarity, there are certainly possibilities that these systems can be integrated in the same framework.

The hierarchical packet labeling scheme is inspired by an observation of the application binding with IP, port pairs. Similar assumptions have also been used for application identification [13].

We also acknowledge the variety of work on behavior-based application identification and early flow fingerprinting [6, 28, 29]. The biggest difference among these methods is the feature set used, and we argue that with a strong and compact feature set, we can make advantageous trade-offs between classifier complexity, accuracy and time-to-bind, and couple high identification accuracy with system performance by using simple and efficient algorithms (such as the C4.5 decision tree).

We have recently seen numbers of proposals intended to tackle the challenges of supporting the diverse range of Internet applications. Some examples include virtual enterprise-network provisioning with Da Vinci [30], co-ordinating the Home network in HomeMaestro [31], and new proposals such as network exception handlers [32], which more closely integrate the end-host into the networks' operation.

8 Conclusion and future work

We have described the design and implementation of the AtoZ traffic organizer: a small, intelligent and high-speed network device. It enables seamless, application-specific traffic management on edge-network, benefiting from a highly-efficient packet-labeling mechanism based on intelligent behavioral flow identification, and high performance packet processing using NetFPGA.

We deal with several technical challenges: how to support the entire system functionality under maximum throughput, without delay or packet loss; all while incorporating effective intelligent traffic classification.

In Section 4, our implementation worked well for smaller instutions ($\approx$1,000 hosts) but was not best-suited to the largest enterprise (15,000 users and close to 100,000 hosts). However, there is no reason for the current constraints of on-chip memory to exist in a

future implementation. Furthermore, we showed the feasibility of a simple, cost-efficient hardware-software hybrid implementation that facilitates seamless and sophisticated traffic management operations on multiple Gigabit-rate links. We are making our implementation available to the community at `http://anonymized` and at the NetFPGA project page `http://www.netfpga.org`.

Limitations and Future work

Our implementation is an early-day prototype, and while deployment experience has been positive, we can identify several areas for future work. Our classification method is neither definitive nor complete: we next intend to investigate replacement schemes that use hybrid methods. Our implementation could logically be integrated into an existing access-router; although current practical limitations make this a challenge for a NetFPGA-based solution. We also note that our work does not provide full traffic-accounting functionality at line rate. However, there is no reason (aside from NetFPGA gate-resource limitations) that AtoZ could not be integrated with the NetFlow probe also developed on the NetFPGA platform[20].

We restate that AtoZ is also not supposed to be a network intrusion detection system or a firewall in its own right. Our prototype has no sophisticated mechanism for detecting attack patterns (although that may be a logical addition). We suggest that this system operates in tandem with existing NIDSes and firewalls.

References

[1] J.W. Lockwood *et al.* "NetFPGA–an open platform for gigabit-rate network switching and routing". In *IEEE International Conference on Microelectronic Systems Education (MSE'07)*, 2007.

[2] I. Stoica *et al.* Core-stateless fair queueing: achieving approximately fair bandwidth allocations in high speed networks. In *Proceedings of ACM SIGCOMM '98*, 1998.

[3] M. Roesch. Snort — Lightweight Intrusion Detection for Networks. In *Proceedings of USENIX LISA'99*, 1999.

[4] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23–24):2435–2463, 1999.

[5] A. W. Moore and D. Papagiannaki. Toward the accurate identification of network applications. In *Proceedings of the Sixth Passive and Active Measurement Workshop (PAM'05)*, March 2005.

[6] L. Bernaille *et al.* Early application identification. In *Proceedings of the ACM CoNEXT'06*, December 2006.

[7] W. Li *et al.* Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks*, 2009. doi:10.1016/j.comnet.2008.11.016.

[8] M. Dusi *et al.* Tunnel Hunter: Detecting application-layer tunnels with statistical fingerprinting. *Computer Networks*, 53(1):81–97, Jan 2009.

[20] NetFlow Probe. `http://netfpga.org/wordpress/?p=165`

[9] G. Maier *et al.* Enriching network security analysis with time travel. In *Proceedings of the 2008 ACM SIGCOMM*, Aug 2008.

[10] Vern Paxson and Sally Floyd. Wide-area traffic: The failure of poisson modeling. *IEEE/ACM Transactions on Networking*, 3:226–244, 1995.

[11] D. Lee and N. Brownlee. On the variability of internet host interactions. In *Proceedings of the 2008 IEEE GLOBECOM*, pages 1–6, Nov 2008.

[12] R. Morris *et al.* The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.

[13] T. Karagiannis *et al.* BLINC: multilevel traffic classification in the dark. In *Proceedings of the ACM SIGCOMM 2005*, pages 229–240, 2005.

[14] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proceedings of the ACM SIGCOMM 2002*, Aug 2002.

[15] A. Broder & M. Mitzenmacher. Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4):485–609, 2003.

[16] N. Williams *et al.* A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *SIGCOMM Computer Communication Review*, October 2006.

[17] H. Song *et al.* Fast hash table lookup using extended bloom filter: an aid to network processing. In *Proceedings of ACM SIGCOMM 2005*, pages 181–192, 2005.

[18] A. Kumar *et al.* Space-code bloom filter for efficient per-flow traffic measurement. In *Proceedings of IEEE INFOCOM*, Mar 2004.

[19] M. Attig and J.W. Lockwood. SIFT: Snort intrusion filter for TCP. In *Proceedings of the 13th Symposium on High Performance Interconnects (HOTI05)*, 2005.

[20] K. Nichols *et al.* RFC 2475: Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers, December 1998.

[21] E. Rosen *et al.* RFC 3031: Multiprotocol label switching architecture, January 2001.

[22] T. von Eicken *et al.* U-net: a user-level network interface for parallel and distributed computing. *SIGOPS Oper. Syst. Rev.*, 29(5):40–53, 1995.

[23] M.E. Fiuczynski *et al.* On using intelligent network interface cards to support multimedia applications. In *Proceedings of the 8th NOSSDAV*, 1998.

[24] I. Pratt and K. Fraser. Arsenic: A user-accessible gigabit ethernet interface. In *Proceed. of IEEE INFOCOM*, 2001.

[25] L. Moll *et al.* Systems performance measurement on PCI pamette. In *Proceedings of IEEE Workshop on FPGAs for Custom Computing Machines*, pages 125–133, 1997.

[26] H. Song *et al.* Snort offloader: A reconfigurable hardware NIDS filter. In *Proceedings of International Conference on Field Programmable Logic and Applications (FPL'05)*, 2005.

[27] N. Weaver *et al.* The shunt: an fpga-based accelerator for network intrusion prevention. In *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays (FPGA'07)*, 2007.

[28] M. Crotti *et al.* Traffic classification through simple statistical fingerprinting. *SIGCOMM Computer Communication Review*, January 2007.

[29] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. In *Proceedings of the 2007 ACM SIGCOMM*, August 2007.

[30] J. He *et al.* Davinci: Dynamically adaptive virtual networks for a customized internet. In *Proceedings of ACM CONEXT 2008*, Dec 2008.

[31] T. Karagiannis *et al.* HomeMaestro: Order from chaos in home networks. *Microsoft Research Technical Report MSR-TR-2008-84*, May 2008.

[32] T. Karagiannis *et al.* Network exception handlers: host-network control in enterprise networks. In *Proceedings of ACM SIGCOMM 2008*, Aug 2008.