A Survey on Packet Filtering

Nik Sultana
Illinois Institute of Technology

Hyunsuk Bang
Illinois Institute of Technology

Elena Yulaeva
CAIDA/UC San Diego

Ricky K. P. Mok
CAIDA/UC San Diego

kc claffy
CAIDA/UC San Diego

Richard Mortier
Cambridge University

Abstract

Packet filtering has remained a key network monitoring primitive over decades, even as networking has continuously evolved. In this article we present the results of a survey we ran to collect data from the networking community, including researchers and practitioners, about how packet filtering is used. In doing so, we identify pain points related to packet filtering, and unmet needs of survey participants. Based on analysis of this survey data, we propose future research and development goals that would support the networking community.

CCS Concepts

• **Networks** → **Network security**; *Network measurement*; **Network monitoring**;

Keywords

Packet Filtering, Network Monitoring, Traffic Classification

1 Introduction

Network monitoring is critical to managing network performance and security. It relies on the ability to selectively capture and analyze packets from the network, both on- and off-line. Packet filters [10] are functions that select packets from a stream. Filters are a key primitive that enable packet capture through two distinct features: (1) languages for expressing constraints on which packets should be captured, (2) techniques for capturing traffic at high data rates. Both have co-evolved alongside data networking over many years [2, 3, 12, 15]: convenient constraint expression was driven by the diversity of deployed protocols, while high-rate filtering was driven by increasing link capacities.

Our **research goal** was to form five inter-related pieces of knowledge based on survey data captured from the networking community: **(1)** which packet filtering *tools and techniques* are prevalent; **(2)** what are the perceived *pain points* of those tools and techniques; **(3)** how are those tools and techniques used in participants' *workflows*; **(4)** what are the filtering *needs* of participants that those tools and techniques do not address; and **(5)** how are responses *correlated* (e.g., "participants who do X are more likely to do Y"). Based on this knowledge, we formulated further packet filtering related research questions relevant to the networking community.

This paper describes data collection (§2), the results of that data collection (§3), and an analysis of correlations we could find within those results (§4). The paper concludes by discussing the survey's limitations (§5) and proposing future research directions to the broader networking community (§6). The questions used in this survey are provided in Appendix A.

2 Methodology

We designed an anonymous online survey for distribution among both research and practitioner data networking communities. Its design went through eleven iterations with small focus groups drawn from both communities, allowing us to calibrate the survey's structure and refine question wording to improve comprehension while keeping the survey reasonably short. Feedback received allowed us to scope our questions to address our points of interest within the broad ecosystem of relevant commercial and research tools.[1]

We obtained IRB exemption from IIT (the institution from which the survey was run) on the grounds that this was an online survey configured not to collect any Personally Identifiable Information (PII), and targeting a professional population. To maximize reach and usability, we used a well-established survey system.

The survey was advertised through various online channels used by network researchers and practitioners, at in-person events and through personal contacts to reach more potential participants. The survey ran from September to November 2023 and had 91 participants. When we released the survey we committed to releasing the resulting data, and this is now freely accessible online.[2]

We exclude from our analysis responses to a question that asked participants to describe the largest network that they managed, due to an error that was not caught before the survey went live: the offered options failed to cover the entire range ("Between 1 and 10 switches or routers" and "Between 100 and 1000 switches or routers" were offered, missing out the range 10—100).

3 Results

We now present a breakdown of survey responses relating to research goals **(1—4)** (§1). As not all participants answered all questions, we report two values in each case: $n\%$ indicates the percentage of participants who picked a specific answer *out of the subset of participants answering that question*, and $\widetilde{n\%}$ indicates the percentage of participants who picked a specific answer *out of the total number of participants in the survey*.

3.1 Population

69% ($\widetilde{62\%}$) of participants use traffic filtering tools in a professional capacity, with experience ranging from 4 months to over 25 years (Fig. 1). Many use these tools frequently (20% ($\widetilde{18\%}$) daily, 25% ($\widetilde{22\%}$) weekly, and 28% ($\widetilde{24\%}$) monthly). 28% ($\widetilde{24\%}$) report that they rarely use these tools. This likely explains a key pain point discussed later, "*reme[m]bering how to use them when I rarely do*".

[1] Feedback example: *Any words that have to deal with "flows" may point to someone thinking about Sflow or Netflow. Words like "traffic" or "packet capture" point more towards using tools like tcpdump or Wireshark. If you want to stay neutral between the tools, I would say something along the lines of "filtering or visualizing the network data".*
[2] http://packetfilters.cs.iit.edu/

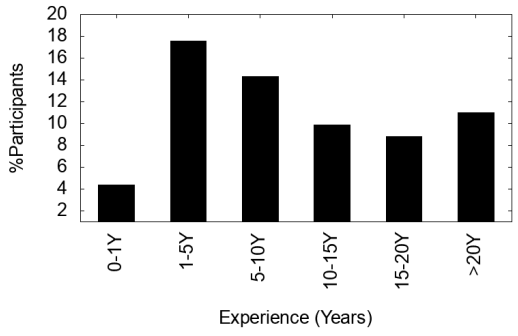

Figure 1: Experience reported by survey participants.

3.2 Tools and Techniques

86% (76%) of respondents used *Wireshark* or *TShark*, 84% (74%) used *tcpdump*, and 29% (25%) used custom programs reliant on *libpcap*.[3] 15% (13%) of respondents used custom-generated eBPF filters. Other tools mentioned that were not in the question set include NetFlow and FPGA cards. Several reported using one or more Intrusion Detection Systems (IDS), principally Snort (9% and 8%), Suricata (13% and 11%) and Zeek/Bro (11% and 10%).

3.3 Context

We sought to understand where and how filtering was carried out.

3.3.1 Environment. Most participants applied filtering tools in corporate or campus networks (50% and 44%), home networks (48% and 42%), datacenter networks (38% and 33%), ISPs (26% and 23%), and backbone networks (15% and 13%). Other options mentioned included experiment testbed networks.

3.3.2 Capturing method. 67% (57%) use an agent on an endpoint (a general-purpose network node), 41% (35%) use port mirrors, 21% (18%) use taps, and 14% (12%) use packet mirrors.[4]

3.3.3 Traffic feed. Most applied traffic filtering online, at 1Gbps (58% and 43%), 10Gbps (42% and 31%), 10—100Gbps (25% and 19%), 400Gbps (3% and 2%). Offline filtering, occurring when packet captures are analyzed after the fact, was applied to captures of less than 1GB (31% and 23%), captures of 1—10GB (25% and 19%), and 10—100GB (13% and 10%). One respondent mentioned targeting both high-capacity links and large files "*Multiple 40—100 Gbps deployments. PCAPs of all sizes, including some well north of 1 TB*".

3.3.4 Diversity of filters. Most respondents (67% and 29%) reported repeatedly using the same filter expressions because workloads and protocols do not change much over time. Specific reasons given included "*restricted scope of usage*", "*usually the same process every time*", "*generally troubleshooting the same issues (i.e. CPE stack implementation like DHCP options)*", and "*I am often performing a similar task, eg: following a protocol for a single application under diagnosis, or looking at all traffic from a physical device*".

[3]In this question, participants could indicate they used more than one tool. Therefore percentages can sum to more than 100%.
[4]Packet mirroring extends port mirroring by applying additional filtering to mirrored traffic [4].

The remainder (33% and 14%) tended to use diverse filter expressions. For example, one responded "*Each scenario I'm troubleshooting is different. Because of the volume of data I need to limit [the filter] to exactly what I need. Too much data just becomes noise.*".

3.3.5 Reasons for filtering. Most used filtering to diagnose connectivity (74% and 56%), followed by doing research (65% and 49%), diagnosing performance (51% and 38%), and diagnosing network configuration (45% and 34%). Free-form answers in this category included "*Protocol validation compliance*" and "*What the heck is breaking windows install? Oh. Something is trying to download that teenytiny little thingy from a non-fqdn-host.*"

Almost a quarter of participants (32% and 24%) also used filters for security-related diagnosis, to determine indications of compromise, detect scanning, detect exfiltration or data leaks, with 17% (13%) using filters as building blocks for firewalls, access control lists (ACLs), and to divert a subset of traffic to an IDS.

3.4 Pain Points

The main challenges participants identified with existing tools were limited performance and scalability (57% and 36%), followed by protocol support (33% and 21%), lack of stateful filter expression primitives (29% and 19%), poor extensibility or difficulty of using that extensibility in practice (21% and 13%), and limited expressiveness (17% and 11%). 17% (11%) felt the learning curve was too steep or documentation needed improvement, while 14% (9%) felt existing tools to be adequate.

Focusing on stateful filters – where subsequent matching behavior depends on one or more previously matched packets – 19% (9%) of respondents indicated use of such filters. This is less than those who indicated that support for such filters is lacking (29% and 19%). From participants' replies, stateful filters are used to "*follow SCTP or TCP payloads, and distinct streams which requires 'state'.*"

Another pain point concerned complexity: "*While I wouldn't necessarily call zeek scripts 'filters', they can get pretty complex. Especially when you start to track state across several workers nodes. Additionally, the DNS and SSL/TLS stuff is very complex due to the nature of the protocols and what's actually seen on the wire.*"

3.5 Workflows and Patterns of Use

Filter generation 52% (24%) of participants reported generating filters via a specialized language, e.g., *tcpdump* or Wireshark expressions, BPF or eBPF assembly, Zeek's filter language, or Snort's signature language, and 17% (8%) used a general purpose language, e.g., C++, while 14% (7%) generated filters automatically using custom, home-grown tools, and 5% (2%) automatically using third-party tools, e.g., *bpfc*.

Complexity Regardless of how they were expressed, 40% (19%) of participants reported that their filter expressions usually consisted of just a single line, 47% (22%) reported their filters were usually fewer than 5 lines, 7% (3%) reported usually 5—20 lines, and 7% (3%) reported usually more than 20 lines.

Free-form comments provided interesting insights into how tools are used, which we categorize as follows:

Expressiveness. Some used low-level languages to write fine-grained filters, as necessary, e.g., "*In rare cases, I observe things by*

manually writing eBPF programs for it." and "*Sometimes using BPF expressions to further filter subsets of the capture.*"

Staging. A number of respondents used iterative and multi-stage approaches when creating filters, e.g., "*Build them up in [an editor] and then paste them in*", "*Manual trial and error*", "*I will use wireshark to build and test the filter, then once it works reuse it with other tools.*" One described developing a filter to find those computers being controlled by an unauthorized user, starting from a set of computers that were known to have been under this user's control, and iteratively widening this set to include machines observed to have communicated with those in the set. During development, the filter itself was used to communicate the scale of the compromise between network and machine administrators.

Integration. Several described use of filters in a larger analysis system, e.g., "*On the rare occasion I'll use Python's SCAPY [to analyze pcap files]*", "*We may also use Splunk to search/filter through capture logs*", and "*We do complex things outside the filtering system.*"

3.6 Desirable Features

Based on the survey data, we believe the following features would be particularly relevant to the users of packet filtering.

W1 High performance. Many indicated performance was a key concern (§3.4).

W2 Clear and consistent filter specification. Although only 17% explicitly complained of limited expressiveness (§3.4), respondents also expressed a need for clear and consistent specification of filters, e.g., "*Not really, I'm pretty comfortable with existing syntax as long as the underlying parts work as advertised.*" and "*Something similar to Wireshark DisplayFilters would be nice, though I dislike gotchas like the NOT modifier https://wiki.wireshark.org/DisplayFilters#gotchas*".

W3 Community. Many of the comments concerning learning curve and documentation could be at least partially addressed by a community of fellow users, helping deliver "*Debugging support*", "*ease of use by newbies*", "*documentation*", "*examples and sample commands*", and even a "*professional support community*".

W4 Integration with standards. One respondent suggested that filtering syntax "*should be generated based on protocols. should be part of standards*" which might also address the points above.

W5 Modern interface tools. Two expressed an interest in using Large Language Models [6] to develop filters: "*Given the prevalence of LLMs, using natural language to describe a filter and have the LLM convert that to tool-specific syntax will be very useful.*" and "*A generative AI for tcp dump filters would be nice. While it's not really something syntax related, it would be useful. Being able to interactively drill down on with something that is "aware" of the goal could be useful.*"

4 Analysis

We next analyze the survey answer data (goal **5** from §1), dividing into two categories: **profiles** of particular types of participants, e.g., those who use many different types of filters, and **correlations** between data features, e.g., for network type X respondents reported using tool Y. Raw survey results were coded for analysis as follows:

Dummy variables. Each multiple-choice question option becomes a *dummy variable*, a new dimension in a vector encoded as "1" or "0" depending on whether the respondent selected it. For example, choice of tools and techniques offered "libpcap" and "Zeek/Bro" as options, each of which would be represented as an extra dimension in the vector representing a respondent's answers with the value indicating if that option was selected or not.

Enumerated values. Each choice is encoded as a small integer value in a single dimension. For example, frequency of use is represented as a single dimension taking values 0 ("daily"), 1 ("weekly"), 2 ("monthly"), 3 ("never"). Later analysis can scale or reorder these values as appropriate.

Grouped values. Responses are grouped into categories. The only example of this concerned participants' experience where "Up to 1 year" consisted of three answers: "10 months", "4 months", "1 year". This grouping was performed manually into the categories that are shown in Fig. 1.

Table 1: Coding process converted responses into three types of variables (§4): D (Dummy), E (Enumerated), G (Grouped). Question Categories are drawn from §3. For example, *Environment* refers to the question that offered 5 choices of network environments: Home network, Corporate network/campus, Datacenter, ISP network, Backbone network. *Population* encompasses 3 questions, each of a different type.

Question Category		# of coded variables
Dependent variables (DVs)		
Tools Used	(§3.2)	9×D
Population	(§3.1)	2×E and 1×G
Environment	(§3.3.1)	5×D
Capturing	(§3.3.2)	4×D
Reasons for filtering	(§3.3.5)	8×D
Traffic feed	(§3.3.3)	8×D
Independent variables (IVs)		
Pain points	(§3.4)	5×D
Filter diversity	(§3.3.4)	1×E
Filter generation	(§3.5)	5×D
Complexity	(§3.5)	1×E
Statefulness	(§3.4)	1×E

4.1 Profiles

We analyze conditional frequencies in multi-dimensional contingency tables using Determinacy Analysis [8, 9], computing properties of *explanatory rules*—statements of the form "if X then Y" that describe the degree of dependence of one variable (Y) on any combination of one or more independent variables (X). Rule *accuracy* (A) represents the proportion of respondents for whom the rule is true, computed as the count of responses with a respective variable value divided by the count of responses with that dependent variable value, while rule *completeness* (S) reflects the proportion of cases explained by the rule, computed as the count of items with both X and Y divided by the count of items with Y. Finally, *contribution* (C) of each factor in a rule is defined as a difference between the accuracy of this rule and the accuracy of the rule with the respective factors removed.

We used the SuAVE online platform [17] for survey analysis and visualization to examine the conditional frequencies of respondents' profile characteristics. SuAVE also includes a collection of

Jupyter notebooks that implement various survey analysis and data management operations, including Determinacy Analysis.

4.1.1 Does any set of characteristics distinguish the 33% who have high *filter diversity*, needing to generate new filters often, from the other 67%? Participants who frequently generate new filters tend to write more complex expressions than participants who reuse filters. Experience plays a pivotal role, with novices more likely to reuse filters and more experienced individuals more inclined to generate new filters, although this effect varies. Filter type, whether stateful or stateless, also influences this distinction, with each contributing differently to the rules' accuracy. Notably, the use of automatic tools for generating expressions/filters significantly boosts the accuracy of rules for participants who prefer not to change filters often. Table 2 quantifies this analysis.

4.1.2 What are the characteristics of users of *stateful* filters, and are they distinguishable from those using *stateless* filters? Differences between participants using stateful and stateless filters center on expression complexity, experience, expression diversity, the tools used, and the method of generation. Stateful filter users typically write longer expressions and use a broader range of tools, including advanced tools like Zeek/Bro and Suricata, indicating a need for complexity and diversity in their filtering practices. They also tend to create new filters more frequently. In contrast, stateless filter users often write shorter expressions, rely on a narrower set of tools, and are more likely to use the same filters consistently. Automatic generation of expressions or filters, especially using third-party tools, is particularly influential for those using stateless filters. Table 3 quantifies this analysis.

4.1.3 What characterizes the participants experiencing each pain point? Examining the groups defined by each pain point reveals similarities and differences in their experiences and behaviors. **Protocol Support** affects $\widetilde{21\%}$ of respondents, primarily those using stateful filters, involved in gathering traffic samples for analysis, and online monitoring of 400 Gbps links. They often cite poor extensibility and limited expressiveness as accompanying concerns. **Limited Performance and Scalability** affects $\widetilde{36\%}$ of respondents, notably those monitoring 400 Gbps links online and/or 10—100 GB PCAP files offline. They state that their filter complexity is usually more than 20 lines. Other characteristics include the use of such tools as custom BPF generation and Zeek/Bro. Those respondents tend to generate filters using automatic custom tools and use various tools for building filters for other systems such as a downstream firewall. **Poor Extensibility** affects $\widetilde{13\%}$ of respondents, and refers more to the context of use than specific factors, for example online monitoring of 400 Gbps links. Other characteristics of respondents experiencing this pain point include the need to build filters for other systems and to write lengthy filter expressions. **Lack of Stateful Filter Expression Primitives** affects $\widetilde{19\%}$ of respondents, and is closely linked to the perception of limited expressiveness. Those affected typically write long expressions, use filters for monitoring 10 Gbps links, and build filters to gather traffic samples for analysis, and for use in other network monitoring tools. Those respondents usually have a moderate level of experience with using tools and generate expressions automatically using custom tools or manually using command line tools or GUI. **Limited Expressiveness** affects $\widetilde{11\%}$ of respondents, many of whom also struggle with the lack of stateful filter primitives and poor extensibility. Characteristics include the automatic generation of expressions with third-party tools, building filters for other systems, and usage across different monitoring scales. Experience levels tend to be lower.

Across these groups, the intersection of technical challenges (e.g., protocol support and performance) with tool-related issues (e.g., expressiveness and filter primitives) creates a complex space where factors – including the type of monitoring activity, the complexity and generation method of filter expressions, and the specific tools and techniques employed – distinguish participants' experiences. However, characteristics common to several groups of participants do exist, such as the need for better extensibility and expressiveness, and the need for better user experience. This suggests areas for improvement in tool and filter design.

4.2 Correlations

We study the correlation between the IVs and DVs (Table 1) using multinomial regression. For each IV coded into multiple dummy variables, we construct multiple models (one per dummy variable) with the same set of DV(s). Our implementation uses the `fitnmr` function in Matlab, employing the ordinal model type and the logit link function. Our analysis focuses on understanding how various user characteristics may lead to different pain points and the creation of filters. Table 4 shows the sets of models we tested. We inspected the statistical significance of *both* the model coefficients and the regression models to derive correlations.

4.2.1 Pain points. Model set M1 revealed several correlations between participants and pain points. We found that participants who generate custom eBPF filters reported a lack of protocol support as a pain point. Given the low-level nature of eBPF, supporting additional protocols can involve substantial development and testing effort. Participants using *libpcap* did not report performance and scalability issues as a pain point, presumably because there is community-wide awareness of other frameworks that provide better performance and scalability (e.g., DPDK [3]). Participants with less experience reported lack of stateful filters as a pain point.

M3.4 in Table 4 revealed correlations between pain points and reasons for using packet filters. We found that the protocols supported by current packet filtering libraries can satisfy the needs of participants who deployed filters for conducting research and diagnosing network connectivity. Furthermore, lack of stateful filter expression primitives was not a pain point for gathering traffic samples and building filters for another system (e.g., firewall, IDS) presumably as the other system can be highly provisioned and horizontally-scaled to perform filtering, compensating for the inability to filter *upstream*—at an earlier point in the network.

4.2.2 Filter generation. M3.2 in Table 4 revealed that how participants generated their filters is affected by their purpose. Filters for network configuration were less likely to be created manually using a special language, presumably as such tasks are well-established and simple/standardized filters exist to detect misconfiguration. Similarly, connectivity testing did not prefer automatically-generated filters in custom tools, presumably for the same reason.

Table 2: The profile of participants who need to generate new filters often, in terms of different factors (shown in the left column). For example, we find that participants who often need to generate new filters tend to use filters whose size is >5 lines. The (A=·,C=·) notation shows the accuracy and contribution values (§4.1), to indicate significance. A high-accuracy but low-contribution result is weaker than one with a high contribution. If the contribution is larger, then accuracy increases.

Factor	Need to generate new filters often	Use the same filters
Complexity	(A=67,C=52) Expressions exceed 5 lines.	(A=65,C=36) Expressions under 5 lines.
Experience	(A=45,C=30) 11—15 years.	(A=67,C=38) Less than 1 year.
Statefulness	(A=63,C=48) Yes	(A=62,C=33) No
Filter generation	(A=33,C=19) Automatically, using custom tools.	(A=100,C=71) Automatically, using third-party tools.

Table 3: Analyzing the profile of participants who use stateful and stateless filters. This table uses the same notation as Table 2.

Factor	Stateful Filters	Stateless Filters
Complexity	(A=33,C=25) Expressions contain 5-20 lines. (A=33,C=25) Expressions contain >20 lines.	(A=88,C=51) Expressions contain 1 line. (A=80,C=43) Expressions contain 2-4 lines.
Experience	(A=40,C=31) >20 years.	(A=88,C=50) 16-20 years.
Filter diversity	(A=38,C=30) Create new filters often.	(A=81,C=43) Use the same filters.
Tools Used	In addition to tcpdump and Wireshark, use of (A=44,C=36) Zeek/Bro and (A=30,C=21) Suricata.	General use of tcpdump and Wireshark. No significant contribution.
Filter generation	No significant contribution.	(A=100,C=62) Automatically, using third-party tools. (A=77,C=40) Manually, using a specialized language.

Table 4: ●/ ◐/ ○ indicate that all / some / none of the models in that model set were statistically significant ($p < 0.05$). E.g., considering M3.3, "reasons for filtering" correlates with the statefulness variable; that is, those reasons given for filtering significantly determine whether to use stateful or non-stateful filters. In contrast, M4.3 indicates that the type of network has no significant bearing on whether stateful or non-stateful filters are required.

Model sets	IV(s)	DV	Results
M1	Tools Used & Population	Pain points	◐
M2	Population	Filter generation	○
M3.1	Reasons for filtering	Complexity	●
M3.2		Filter generation	◐
M3.3		Statefulness	●
M3.4		Pain points	◐
M4.1	Environment	Filter generation	○
M4.2		Complexity	○
M4.3		Statefulness	○
M4.4		Pain points	○

M3.1 showed that filters for security or performance purposes were more complex, presumably due to higher specificity as they must cover everything from simple worm signatures, to exfiltration indicators (e.g., for HeartBleed [14]), to IDS evasion [7].

M.3.3 showed sampling traffic for analysis was more likely to use stateless filters, albeit only with marginal significance (p=0.066). We believe that this follows similar reasoning to the stateless filtering described in §4.2.1, i.e., developing complex upstream filtering is not needed if downstream resources can support high-quality filtering.

4.2.3 Other. Finally, we could not find significant correlations between network environment and several DVs (M4.1—4.4), probably as many participants managed multiple networks including their home network. Re-running the survey with more diverse participants might help examine this further. Manual generation of filters was the most popular approach (§3.5) but we did not find a significant relationship between the approach used to generate filters and the level of experience reported by participants (M2). This might mean that there are well-established "best practices" in packet filtering that are quickly adopted by newcomers. We observed that those reporting lower filter diversity (§3.3.4) tended to use longer filters (§3.5), and those reporting higher filter diversity used fewer lines in their filters.

5 Limitations

The data from the survey indicates the views of a relatively small sample of the population, and is not necessarily representative of the total population. Another limitation is that analysis is hamstrung by lack of knowledge of participants' backgrounds. This could have been mitigated by adding more questions, but we tried to keep the survey brief to encourage participation. If the survey is repeated in the future, we hope that the outcomes from the first survey will encourage more participants to engage.

6 Concluding Proposals

We propose some research objectives based on the survey's findings.
P1 Flexible upstream filtering. Observations in §4.2.1 and §4.2.2 suggest that the abundance and affordability of downstream processing resources disincentivize the development of flexible upstream filtering, as a 400 Gbps link could be split into 4×100 Gbps streams for analysis on a cluster. However, as link capacities continue to increase steadily even outside of hyperscale datacenters [1, 13], network operators will need to balance performance (**W1**) against expense. We suggest a research goal might be to decompose filters so that parts could be pushed upstream to reduce the load on downstream processing resources, and thus the expense in procuring and maintaining those resources.
P2 Integrated behavior of distributed filters. **P1** divides labor between upstream and downstream filters rather than migrating filters upstream wholesale, as complex and stateful filters would be challenging to move upstream and made to work at high data rates. This division of labor necessitates coordination between filters, not an entirely new idea (workflow descriptions in §3.5 already indicate that some run distributed filtering infrastructure) but it underscores the need for coordination of high-performance filtering points. A research goal would be to compose and coordinate distributed filters, without causing "surprising" behavior (**W2**) [11], and to serve different participants (§3.3.4) and their workflows (§3.5). **P3** Explainability of *performance*. Independent of level of experience, participants tend to manually generate filters (§4.2.3). As well as controlling what they capture, this affords control over resources used by a filter, ensuring high performance (**W1**). Even simple filters can have unwanted behavior that wastes resources, e.g., checking for transport headers that do not occur in your network [14]. A research goal would be to provide per-filter, ahead-of-time performance estimates, e.g., how many clock cycles a filter would take on a specific target architecture. This enables the principle of "paying" for what you use, as seen in performance-vs-security trade-offs in extensible access control [16]. For packet filtering, this performance estimate could be provided as part of a capacity plan that is parametrized by total data rate and expected traffic composition.
P4 Explainability of *behavior*. Widely-used filtering languages have several quirks (**W2**), more likely to be triggered by complex filters. We observed that security-related filters tend to be more complex (§4.2.2), increasing the risk of unwanted filtering behavior leading, in turn, to outages or gaps in visibility. A research goal would be to convert filters into more legible equivalent forms, perhaps using natural language interfaces (**W5**); Net2Text explored a similar idea for connectivity [5]. In this case, this goal would help network operators and administrators better understand the behavior of complex or third-party filters. This interface would also provide ease of reference when a community is not available (**W3**), such as when scenarios, filters, or protocols are highly specific.

Finally, as networks continue to evolve, we suggest that this type of survey might usefully become census-like, running regularly to continually understand how better to provide the primitives on which our networks rely.

Acknowledgments

We thank the anonymous reviewers and the following for feedback: Jeronimo Bezerra, Adrian Bucurica, Richard Clayton, Jon Crowcroft, Peter Dordal, Adrian Farrel, Babar Kamran, Hyojoon Kim, Jeff Mogul, Denis Ovsienko, Ben Pfaff, Nishanth Shyamkumar, Italo Da Silva, Jim Tufts, Anthony Ulloa, Vinod Yegneswaran. This work was supported by a Google Research Award, the Defense Advanced Research Projects Agency (DARPA) under contract HR0011-19-C-0106, and National Science Foundation (NSF) under awards 2120399, 2131987, 2319959, 2346499. Any opinions, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funders.

References

[1] 2023. NSF FABRIC project announces groundbreaking high-speed network infrastructure expansion. https://learn.fabric-testbed.net/knowledge-base/nsf-fabric-project-announces-groundbreaking-high-speed-network-infrastructure-expansion/. (2023).
[2] 2024. Corsaro 3: the Parallel Edition. https://github.com/CAIDA/corsaro3/tree/master. (2024). Accessed: 2024-02-26.
[3] 2024. Data Plane Development Kit. https://www.dpdk.org/. (2024). Accessed: 2024-02-26.
[4] 2024. Juniper Networks mirror encapsulation (Jmirror). https://wiki.wireshark.org/jmirror. (2024). Accessed: 2024-02-26.
[5] Rüdiger Birkner, Dana Drachsler-Cohen, Laurent Vanbever, and Martin Vechev. 2018. Net2Text: query-guided summarization of network forwarding behaviors. In *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation (NSDI'18)*. USENIX Association, USA, 609–623.
[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://arxiv.org/abs/1810.04805
[7] Mark Handley, Vern Paxson, and Christian Kreibich. 2001. Network intrusion detection: evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th Conference on USENIX Security Symposium - Volume 10 (SSYM'01)*. USENIX Association, USA, 9.
[8] R. Kuusik and G. Lind. 2010. Some developments of determinacy analysis. In *Advanced Data Mining and Applications: 6th International Conference, ADMA 2010, Proceedings, Part I (Lecture Notes in Computer Science)*, Vol. 6. Springer Berlin Heidelberg, Chongqing, China, 593–602. November 19-21.
[9] Philip A. Luelsdorff and Sergej V. Chesnokov. 1996. Determinacy form as the essence of language. *Prague Linguistic Circle Papers* 2 (1996), 205–234.
[10] J. Mogul, R. Rashid, and M. Accetta. 1987. The Packer Filter: An Efficient Mechanism for User-Level Network Code. *SIGOPS Oper. Syst. Rev.* 21, 5 (nov 1987), 39–51. https://doi.org/10.1145/37499.37505
[11] Eric S Raymond. 2003. *The Art of Unix Programming*. Addison-Wesley Professional.
[12] Luigi Rizzo. 2012. netmap: A Novel Framework for Fast Packet I/O. In *2012 USENIX Annual Technical Conference (USENIX ATC 12)*. USENIX Association, Boston, MA, 101–112. https://www.usenix.org/conference/atc12/technical-sessions/presentation/rizzo
[13] Michael Smitasin. 2023. Network Tapping for Zeek: A Deep Dive. Presentation at ESnet's CI Engineering Lunch & Learn Series. (2023).
[14] Nik Sultana. 2019. What we talk about when we talk about pcap expressions. In *Proceedings of the 4th ACM International Workshop on Real World Domain Specific Languages, RWDSL@CGO 2019, Washington, DC, DC, USA, February 17, 2019*, Robert J. Stewart and Greg J. Michaelson (Eds.). ACM, 2:1–2:9. https://doi.org/10.1145/3300111.3300113
[15] Gerry Wan, Fengchen Gong, Tom Barbette, and Zakir Durumeric. 2022. Retina: analyzing 100GbE traffic on commodity hardware. In *Proceedings of the ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. Association for Computing Machinery, New York, NY, USA, 530–544. https://doi.org/10.1145/3544216.3544227
[16] Robert N. M. Watson. 2013. A decade of OS access-control extensibility. *Commun. ACM* 56, 2 (feb 2013), 52–63. https://doi.org/10.1145/2408776.2408792
[17] Ilya Zaslavsky, M.M. Burton, and T.E. Levy. 2017. A new approach to online visual analysis and sharing of archaeological surveys and image collections. In *Heritage and Archaeology in the Digital Age: Acquisition, Curation, and Dissemination of Spatial Cultural Heritage Data*. 133–150.

A Questions

This appendix lists the questions in our survey. The survey dataset includes answers for each question.[2]

Q1 Which tools/libraries do you use for traffic filtering, packet capture, deep inspection or intrusion detection, or traffic monitoring as part of network monitoring?
- libpcap with your custom program
- libnids with your custom program
- tcpdump
- Wireshark or tshark
- Custom generation of BPF
- Custom generation of eBPF
- Zeek/Bro
- Suricata
- Snort
- Other (Describe)

Q2 Do you use these tools in a professional capacity now?

Q3 Approximately for how long have you used these tools?

Q4 Approximately how frequently do you use these tools?
- Daily
- Weekly
- Monthly
- Rarely

Q5 On which types of networks do you use these tools?
- Home network
- Corporate network/campus
- Datacenter
- ISP network
- Backbone network
- Other (Describe)

Q6 How big is the largest network you used these tools on?
- Between 1 and 10 switches or routers
- Between 100 and 1000 switches or routers
- Between 1000 and 10,000 switches or routers
- More than 10,000 switches or routers

Q7 For capturing traffic on network links, which techniques do you use?
- Agent on the end-point (i.e., a general-purpose network node)
- Port mirrors on switch/router (e.g., using SPAN or RSPAN)
- Packet mirrors on switch/router (e.g., using ERSPAN)
- Optical/Electrical Taps
- Other (Describe)

Q8 For what types of activities do you use the tools mentioned above?
- Diagnosis: Connectivity
- Diagnosis: Network configuration
- Diagnosis: Security (indication of compromise, scanning detection, detecting exfiltration or data leaks)
- Diagnosis: Performance
- Gathering traffic sample for analysis (e.g., for compliance with regulation)
- Building filters for another system (e.g., firewall, ACL, to divert a subset of traffic to an IDS)
- Research
- Curiosity (e.g., seeing chatter from IoT devices)
- Other (Describe)

Q9 "Pain points" you experience when using these tools.
- Protocol support (e.g., IPv6, legacy protocols, etc).
- Performance and scalability is too limited.
- Poor extensibility, or difficulty of using that extensibility in practice.
- Lack of stateful filter expression primitives (whose behavior is influenced by one/more previously-matched packet/frame(s)).
- Expressiveness is too limited.
- Other (Describe)

Q9.5 Can you give concrete examples of the type of pain points you chose in the previous question?

Q10 When filtering network traffic, what is your typical usage scenario?
- Online (live / real-time) monitoring of 1Gbps link
- Online monitoring of 10Gbps link
- Online monitoring of 10-100Gbps link
- Online monitoring of 400Gbps link
- Offline (not real time) monitoring of <1GB pcap file
- Offline monitoring of 1GB-10GB pcap file
- Offline monitoring of 10-100GB pcap file
- Other (Describe)

Q11 Expression diversity you typically encounter when filtering traffic.
- I often need to write/generate new filters. (Describe why)
- I mostly use the same filter expressions/specifications. (Describe why – e.g., workloads and protocols don't change much.)

Q12 How do you generate expressions/filters?
- Manually using a command line tool or GUI. (Please give details.)
- Manually using a general purpose language (e.g., C++). (Please give details.)
- Manually using a specialized language (e.g., tcpdump or Wireshark expressions, BPF or eBPF assembly, Zeek's filter language, Snort's signature language, etc). (Please give details.)
- Automatically by using third-party tools (e.g., bpfc). (Please give details.)
- Automatically by using custom, home-grown tools. (Please give details if publicly available)

Q13 Typical filter expression complexity (regardless of whether expressed in C++, tcpdump's language, etc)
- One line
- Fewer than 5 lines
- Between 5 and 20 lines
- Greater than 20 lines

Q14 Do you tend to need stateless filters (i.e., filters that behave uniformly across all frames/packets) or stateful filters.
- Stateless
- Stateful (Please provide examples)

Q15 Do you have examples of challenging (e.g., tricky to understand or change) but frequently-used filter expressions or scenarios you can share?

Q16 Do you have examples of complex filter expressions or scenarios you can share?

Q17 Do you have examples of filtering tasks that existing tools or languages struggle with, or cannot handle?

Q18 Do you have examples of what your ideal syntax would look like for some filter expressions? And what sort of behavior would you expect for that syntax?