Ecological Interface Design: Practical Applications to the Wireshark Network Analyzer

Evan Huus (Carleton University)
eapache@gmail.com

December 8, 2013

Abstract

The open-source Wireshark application is over fifteen years old and is widely hailed as the gold standard in network protocol analysis. Wireshark's user interface is in the process of being rewritten from scratch for both technical and usability reasons, making it an ideal candidate for research in human-computer interaction (HCI). This paper investigates practical applications of the Ecological Interface Design framework to Wireshark's design. We define the problem space, develop the necessary structures using Ecological Interface Design, then evaluate both the old and new versions of Wireshark's interface using those structures. Finally, we use the evaluation to provide concrete recommendations for future Wireshark development.

1 Introduction

1.1 Wireshark

The open-source Wireshark application[1] bills itself as "the world's foremost network protocol analyzer". It provides users with the ability to see and inspect the packets flowing through modern computer networks, supporting many hundreds of different network protocols. It is a mature, feature-rich project with over fifteen years of history as of July 2013[2].

In this time frame, Wireshark's graphical user interface design has not changed in any substantial way. It uses the same three-panel design used in many other network "sniffer" and protocol analyzer applications, a design dating back at least to 1989[3]. This layout consists of three vertically-stacked panes: the topmost pane contains a list of packets with basic summary information for each one, the middle pane contains protocol details for whichever packet is selected from the list, and the bottom pane displays the raw bytes of the selected packet.

[1] https://www.wireshark.org
[2] https://blog.wireshark.org/2013/07/fifteen-years/
[3] Laura Chappell, personal communication, November 3, 2013.

While this interface has served Wireshark well, it has begun to show its age. Additionally, the library originally used to write Wireshark's UI does not support several platforms which Wireshark wishes to support. In light of these factors, it was recently decided to rewrite Wireshark's GUI module from scratch using a new library[4]. Wireshark's long history and current rewrite make it an excellent subject on which to apply some modern theories in HCI.

1.2 Ecological Interface Design

In 1992, Vicente and Rasmussen proposed "a novel theoretical framework for interface design for complex human-machine systems" [6] and called it Ecological Interface Design (EID). Initially developed as an attempt to bring the powers of direct manipulation interfaces to new problem domains, EID became a fully-fledged framework in its own right and has been shown to have a number of practical applications [1, 2, 3, 4]. EID brings two distinct structures to bear on any given problem: the abstraction hierarchy, and the skills, rules, knowledge taxonomy. When combined, these structures provide substantial insight into interface design.

1.2.1 The Abstraction Hierarchy

The abstraction hierarchy in EID is a set of layers, each dealing with the same underlying system at a different level of detail and control. Hierarchies are a common theme in HCI already, but the abstraction hierarchy is distinguished by having, according to Vicente and Rasmussen, an "explicitly goal-oriented nature" [6]. Other hierarchies may be oriented around physical or temporal scale, or the flow of a particular resource (often information), but the abstraction hierarchy is oriented around ends, and the means of achieving those ends, bringing it in close synchronicity with users' real-world problem-solving patterns.

It is worth noting that the abstraction hierarchy is not a single specific hierarchy but a class of hierarchies fitting this general pattern. Each problem-space to which EID is applied must develop its own hierarchy specific to that problem, with its own layers and abstractions. There is a great deal of literature suggesting that users automatically develop their own abstraction hierarchies when problem-solving within a domain (several examples are cited in [6]), so EID recommends that system interfaces should generally be structured in the same way. This provides a natural mapping between the interface and the user's mental model.

1.2.2 Skills, Rules, Knowledge

The second structure used by EID is the taxonomy of skills, rules and knowledge, often abbreviated as SRK. First developed by Rasmussen in an earlier paper [5], the SRK taxonomy is concerned primarily with the three ways in which people process and react to information.

[4] `https://blog.wireshark.org/2013/10/switching-to-qt/`

At the lowest cognitive level, skill-based behaviour consists of the nearly subconscious actions and muscle memory that make up many acquired skills. This behaviour is often very efficient as it requires almost no cognitive resources, but it can only deal with known, regular tasks. Rule-based behaviour sits one level up, consisting of known mappings between particular inputs and responses. It is slightly less efficient since the mapping requires some cognition, but it can deal with situations that are uncommon enough to have no skill-based behaviours, although it still cannot deal with situations that are entirely unforeseen. Knowledge-based behaviour is the higher-level, conscious problem-solving behaviour that we engage in when presented with unfamiliar scenarios. It is inefficient and relatively error-prone compared to the lower levels, but it allows us to deal theoretically with any potential problem-space.

Based on this taxonomy, EID suggests that interfaces should let users act at the lower, more efficient levels as much as possible. However, it explicitly notes that knowledge-based behaviour is sometimes unavoidable, and that interfaces should therefore still provide substantial support for that level of behaviour.

1.3 EID and Wireshark

In order to apply Ecological Interface Design to any application, we must first clarify the problem space and use cases targeted by that application. In broadest terms, Wireshark's goal is simply that of "helping the user to get information about their network". This is clearly far too broad to be useful, although it does provide a convenient umbrella under which Wireshark has several more specific use cases:

Exploring Wireshark is frequently used as an exploratory learning tool, allowing users to discover and watch the behaviour of networked systems in real time.

Auditing Wireshark can be used by auditors to inspect network traffic in order to ensure that protocol standards are being respected and to verify the absence of malicious traffic.

Monitoring Wireshark is also occasionally useful as a tool for monitoring specific sections of a network in order to locate problems or verify their resolution.

Debugging In perhaps its most common use, Wireshark permits users to debug networked applications and protocols by inspecting the specific structure of packets in network traffic.

While these four uses are distinct, it is notable that all but the first are concerned with network errors in one form or another. We shall therefore focus our attention on this dimension of Wireshark usage: the detection, identification and analysis of errors in a network. Importantly, we worry only about errors at the software level; unplugged network cables and other hardware issues are outside the scope of this problem space.

To provide a motivating example, consider a network engineer who has noticed unusual behaviour in certain connections to her web server. She uses Wireshark to audit the network traffic and notices some problematic

packets. She then uses Wireshark to inspect the contents of those packets and debug the problem. Having resolved the root cause on the web server, the engineer now uses Wireshark to monitor the network traffic and verify that the problem has actually been fixed.

2 Applications to Packet Analysis

With our problem space now clearly defined we can use it to explore applications of EID. In this section we consider several possible abstraction hierarchies for the domain. We then apply the skills, rules and knowledge taxonomy to the various forms of cognition that a user may have to perform during the problem-solving process. Along the way, we locate practical applications of these principles and use them to sketch components of a potential interface design.

2.1 Abstraction Hierarchies

2.1.1 Protocol-Based Hierarchy

Within the realm of packet analysis, several hierarchies already exist which might be drawn on to construct an abstraction hierarchy. One such hierarchy is the OSI model (ISO/IEC 7498-1), which breaks network communications down into seven layers. The bottom-most layer, officially called the physical layer, is not one that packet analysis tools usually have access to, but the remaining six layers might be used to derive an abstraction hierarchy.

In this hierarchy the highest level of abstraction is the application layer, where userspace applications can communicate over protocols like HTTP without worrying about lower-level details. Below this is the mostly-unused presentation layer, and below that is the session layer where protocols such as RTP handle the establishment and negotiation of communication sessions. Protocols like TCP and UDP are another level down at the transport layer, and protocols like IP are even further down at the network layer. The bottom-most relevant layer would be the data link layer, which is the realm of the Ethernet framing protocol and other details.

This hierarchy is well-established and seems like it might be a natural fit, but it has several problems that make it less than ideal. One issue is that particular layers may be missing or over-represented in specific packets. This can occur when HTTP is carried directly over TCP for example, skipping two layers of the hierarchy. Alternatively it is common for IPv6 to be tunnelled over IPv4 for compatibility reasons, effectively duplicating the network layer.

A more serious concern with this model is that it seems somehow backwards when compared to how our hypothetical network engineer would actually approach her problem. If she wanted to narrow her search (or "zoom in" to use the terminology of Vicente and Rasmussen) to just a particular machine and then to just a particular stream of HTTP traffic on that machine, she would have to access the layers of this hierarchy

in reverse: different machines are distinguished at the low-level data-link and network layers, while traffic streams are distinguished at the mid-level transport layer. The actual HTTP traffic however, is located at the application layer.

2.1.2 Traffic-Based Hierarchy

While evaluating the previous attempt at a protocol-based abstraction hierarchy, the shape of another potential hierarchy revealed itself. This hierarchy, based more on the flow of information than on specific protocol structures, uses a byte of data "on the wire" as its atomic unit. Bytes can be grouped together to form packets of related data, providing our first layer of abstraction.

Subsequent layers of abstraction can be effectively generated by grouping elements of the previous layer together. Related packets can be grouped by request/response pairs, for example, or according to which web resource they were involved in accessing. These simple groups form what I will call conversations. Conversations can be further grouped and abstracted into sessions, collecting perhaps the numerous conversations that occur as a user browses through a website. These sessions may themselves be aggregated based on which machine(s) on the network were involved.

This abstraction hierarchy seems initially more useful to our fictitious network engineer. She already knows which sessions and machines are displaying the problematic behaviour, so this hierarchy more closely matches her mental model as she zooms in to focus on just the relevant packets. However she now hits a snag. This hierarchy says nothing about the protocols or their relations, leaving her unable to zoom in any farther. If the problem is in the HTTP protocol, she must now wade through a mass of unnecessary detail in the Ethernet, IP and TCP protocols in order to determine the actual error.

2.1.3 Combined Hierarchy

We have now considered two possible hierarchies, but both of them had non-trivial problems when viewed from the perspective of the network engineer. The traffic-based hierarchy provided excellent support for zooming in to just the relevant packets, but left the actual protocol layers undifferentiated. The protocol-based hierarchy did a decent job of abstracting the protocols, but provided no way to focus on only the relevant packets. In this way, it seems like the two hierarchies are actually somehow complementary.

Based on this appearance, we have developed a third potential abstraction hierarchy, built by combining simplified versions of the two previous attempts into a single structure. Starting at the most abstract this hierarchy moves from endpoints to conversations to packets, and then through a subset of the OSI model: the application layer, the transport layer, the network layer and the data link layer. When more than four protocols appear in a packet, as in the IPv6 tunnelling case, they are grouped appropriately.[5]

[5]In our example of IPv6 tunnelled over IPv4, both would appear at the network layer.

This combined hierarchy seems to do the job. Our network engineer will start her problem-solving process by selecting the endpoint and conversations of interest, presenting her with only the relevant packets. She then inspects the application-layer protocol for errors. If she cannot locate or identify the error there, she peels back another layer to view the transport protocol(s) and resumes her search.

2.2 The SRK Taxonomy

The skills, rules and knowledge taxonomy is the other primary component of EID. The applications of this taxonomy to the packet analysis domain are not immediately obvious, but we can gain some useful insights nonetheless. The primary problem in applying this taxonomy to packet analysis applications is that they provide what is effectively a read-only interface: the processes actually generating the traffic are far outside of the analyzer's control, so there is no way for the analyzer to manipulate the traffic being sent. As such, there is no way for it to provide a display structure isomorphic to the system's control structure.

Despite this problem, we can still take advantage of the SRK taxonomy. In particular, the user must still interact with the application's interface to navigate the abstraction hierarchy and control what information is displayed. According to Vicente and Rasmussen, any information given to the user "can be interpreted in three mutually exclusive ways — as signals, signs, or symbols" [6]. Each of these ways triggers its respective behaviour in the SRK taxonomy.

2.2.1 Signals and Navigation

We focus first on signals triggering skill-based behaviour, and their applications to navigation. Rasmussen wrote in 1983 that "in general, the skill-based performance rolls along without the person's conscious attention, and he will be unable to describe how he controls and on what information he bases the performance" [5]. We can infer from this that an interface containing good navigational signals will allow the user to navigate nearly subconsciously.

Consider the abstraction hierarchy from the previous section. As in most abstraction hierarchies, the primary navigational direction will probably be "down", to reveal the next layer of detail. However, moving up the hierarchy may also occur if the user travels too far and realizes that the information they need is more clearly presented at a higher level. Movement in other directions may also be convenient if, for example, one is tracing an event from machine to machine in the network. In this case it may be very useful to stay at the application level of the hierarchy, but jump sideways from host to host.

At the traffic-based levels of the hierarchy, the layers have a container-contained relationship, where for example one endpoint may be involved in many conversations. One way to match the user's mental model in this case might be to display a spatial "map" of the elements at the current layer, with a brief summary next to each one. Selecting an element could zoom in to the set of sub-elements it contains.

At the protocol-based levels of the hierarchy things are somewhat more complicated. Many protocols such as TCP permit payloads to be spread arbitrarily across actual packet boundaries, meaning that there may be no consistent relationship between the apparent "packets" at two different layers. A better choice for this level might be a stack-like structure similar to that commonly used for software architecture diagrams[6], such that higher-level packets are shown on top of the lower-level packets from which they are derived.

2.2.2 Signs and Errors

While skill-based behaviour seems primarily useful during navigation, signs and rule-based behaviours are more directly applicable to the problem of error detection. Many if not all of the anomalies a user might want to detect can be precomputed by the analyzer software without human assistance. For example, the TCP protocol header contains a checksum field which is relatively straightforward to verify once located. The application's job in this case is to "provide a consistent one-to-one mapping between the work domain constraints and the cues or signs provided by the interface" [6].

The obvious first step is to visibly flag detected errors in a way that draws the user's attention to anomalous packets, but an application should do more. Different errors can have different levels of importance which should be distinguishable: critical errors should stand out even amidst a sea of lesser issues. It is also important to aggregate errors at different levels of the hierarchy. While TCP checksum errors must obviously be visible at the TCP level, the display of endpoints should presumably still flag an endpoint that is experiencing many such errors.

This display logic becomes more complex when errors exist at multiple levels. If there is an error in the current application-level packet, for example, but also in the underlying transport-level packet, then both should be flagged. The application-level error must be more obvious as that is the layer currently in focus, but the transport-level error must also be visible as it may in fact be the cause of the higher-level error. The display of errors with different severities at different levels of abstraction requires a careful balancing act.

2.2.3 Symbols and Decoding

The final prong of the SRK taxonomy is the use of symbols and knowledge-based behaviour. People use knowledge-based behaviour when they encounter a situation that is both unfamiliar to them and unanticipated by the application. In packet analysis these scenarios generally take one of two forms: an unanticipated error in a known protocol or an entirely unknown protocol. While the application should attempt to minimize occurrences of either situation, it should still support the user in both.

[6] See for example the following diagram of open-source display and UI toolkits: `http://en.wikipedia.org/wiki/File:Free_and_open-source-software_display_servers_and_UI_toolkits.svg`

In the case of an unanticipated error condition, the violated constraints can usually be described by relations between various protocol fields in one or more packets. For example, the TCP checksum field is defined to be the 16-bit checksum of the remaining packet bytes[7]; the relationship between these two components is a constraint on both, and any violation of that constraint is considered an error. To aid the user in locating and calculating these violations, the application should therefore permit the user to compare, list and operate on field values as flexibly as possible, both within and between packets.

In addition to allowing the user to identify these violations, the application should also allow the user to easily locate other instances of the same type of error. This may not always be possible if, to continue our previous example, the application is unable to calculate checksums at all, but in simpler cases the user should be able to formally describe the constraint violation and have the application do the heavy lifting. These additional constraints should be persistent, so that in subsequent sessions the user does not have to remember or re-enter them.

In the case of entirely unknown protocols, the only data that the user has to work with are the raw bytes of the packet and often some external protocol specification. In this case the application should allow the user to add meaning to the bytes by interpreting and labelling them. The user should be able to easily apply these structures across multiple unknown packets, and should be able to operate on them with the same tools used for working with built-in protocols.

3 Evaluating Wireshark

In the previous section we explored and developed many design principles and structures related to the general problem of packet analysis, with a particular focus on detecting, identifying and analyzing errors. In this section, we extract and summarize a set of domain-specific design criteria. We then use these criteria to evaluate both the old and new versions of Wireshark's user interface and to provide recommendations.

Every effort was made to minimize unnecessary differences between the two versions; both were built and tested on the same machine with the same set of packets and default settings. The "old" version was built from revision 53547 of the `trunk-1.10` branch using the GTK-based interface, while the "new" version was built from revision 53607 of the `trunk` branch using the Qt-based interface. Both revisions were the most recent available as of the time of writing. It is worth noting that the Qt-based interface was not entirely complete at this point — all core functionality had been implemented, but it was far from a finished product.

3.1 Evaluation Criteria

The following evaluation criteria have been extracted and summarized from the previous sections for simplicity's sake:

[7] The actual checksum algorithm is rather unusual and somewhat more complicated as it involves calculating an extra pseudo-header. For a full specification see IETF RFC 793.

1. The interface should have a clear abstraction hierarchy, supporting endpoints, conversations, packets, and the different protocol layers.
2. The interface should permit easy navigation between the layers of abstraction, and make obvious the relationships between the different layers. If the user's focus is on a particular layer, the other layers should not get in the way.
3. The interface should visibly flag errors, differentiate between errors with varying levels of importance, and summarize errors appropriately at different layers of the abstraction hierarchy.
4. The interface should allow the user to compare, list and operate on protocol fields within and between packets. It should allow the user to describe and mark constraint violations, and should persist these descriptions across sessions.
5. The interface should allow the user to interpret and label bytes of unknown protocols, reuse these interpretations across multiple packets and sessions, and operate on them as if they were built-in.

3.2 Old Interface

3.2.1 Primary Window

The primary window for Wireshark's old interface (Figure 1) follows the traditional three-panel layout with the addition of a display filter toolbar. Each of these four elements has been highlighted and numbered in red.

The display filter (element `1`) is a powerful feature that allows users to combine comparison operators, protocol fields and Boolean logic to display specific subsets of packets. For example, a user could enter `(ip.src == 192.168.1.101 and http)` and Wireshark would display only the HTTP packets being sent by that particular IP address[8]. Frequently used filters can be saved by the user so they persist across multiple sessions. However, while the display filter fulfils a good part of criterion #4, it does not support multi-packet filters and lacks operators for calculating (as opposed to simply comparing) packet fields.

The second component of Wireshark's interface is the packet list (element `2`). It lists the subset of packets matching the current display filter, if any. For each packet, a number of columns display summary information, and the packet's row is coloured according to certain rules. The packet list is extremely flexible: users can configure custom columns and colouring rules using the same syntax as the display filter[9]. This flexibility allows the packet list to satisfy parts of criteria #3 and #4; with the appropriate colouring rules, problematic packets will stand out in the list, and custom columns allow the user to perform basic multi-packet comparisons. One limitation of the protocol list is that it always displays exactly one row per packet of data "on the wire". If, for example, a packet contains multiple

[8] See `www.wireshark.org/docs/man-pages/wireshark-filter.html` for a complete description of display filter syntax.

[9] See `www.wireshark.org/docs/wsug_html_chunked/ChCustColorizationSection.html` for a description of how to set up custom colouring rules.

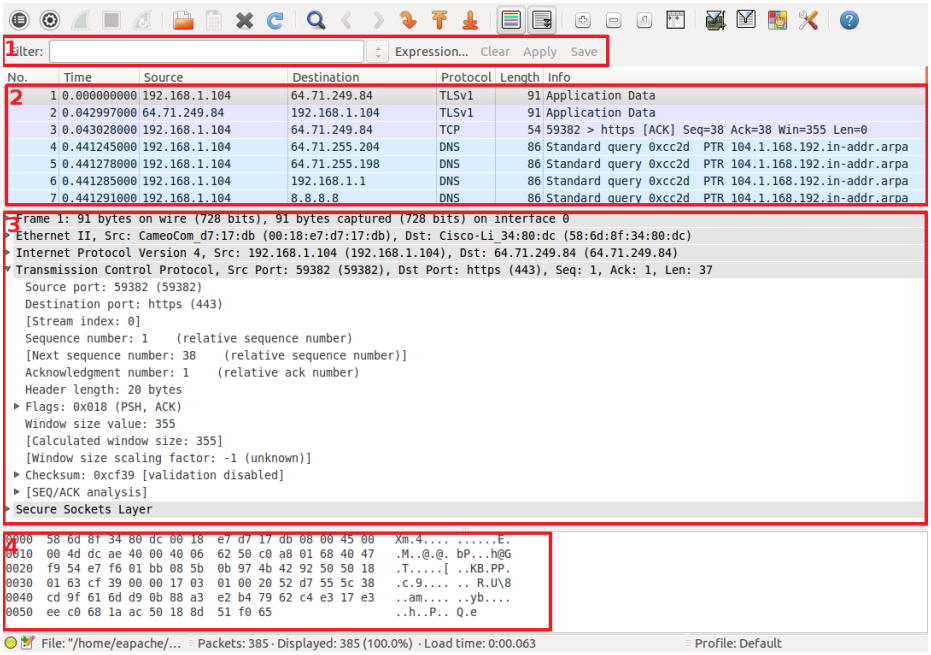

Figure 1: The GTK-based user interface of Wireshark 1.10.

HTTP requests, then there is no way to have each request displayed on its own row.

Below the packet list are the two packet-specific parts of Wireshark's interface: the protocol tree (element 3) and the byte pane (element 4). The protocol tree contains a list of every protocol, field and interpreted value detected in the currently-selected packet, in the form of an expandable tree. The bytes pane simply lists the raw bytes of the current packet in several formats. The two elements are linked: selecting a field in the tree highlights the bytes representing that field in the packet, and clicking on a byte in the packet automatically focuses the relevant field in the tree.

The tree provides a relatively clean view of the protocol-based levels of the abstraction hierarchy, however it suffers from the same limitation as the protocol list: there is no way to hide lower-level framing or separate cohabitating higher-level payloads. Despite this problem, it is substantially more flexible than the pure abstraction hierarchy with respect to nested and tunnelled protocols, and partly satisfies criteria #1 and #2.

3.2.2 Other Elements

The traffic-based levels of the abstraction hierarchy can be found amongst the plethora of additional windows accessible via Wireshark's menus. These windows include one providing a list of identified network endpoints and another providing a list of identified conversations; both windows will automatically generate display filters for focusing on the relevant

packets. While this partly satisfy criterion #1, it utterly fails to satisfy criterion #2: the relationships between the levels are not apparent, and the windows are difficult to discover and navigate.

Another of these extra windows provides a comprehensive list of Wireshark's calculated "expert information" for the current packets. This information consists of all interesting or problematic events identified in each packet, grouped together by type and severity. These can include everything from error-level TCP checksum mismatches to chat-level events indicating the normal closing of a connection.

Wireshark's expert information appears in many other elements of the interface as well, comprehensively satisfying criterion #3. In the very bottom-left corner of Figure 1, the yellow circle indicates that "Warning" is the most severe level found in the current set of packets. The filter engine also supports expert information, providing three special fields (`expert.group`, `expert.message` and `expert.severity`) for use in display filters, columns and colouring rules. In the protocol tree, expert information shows up as a special generated field, coloured according to its severity.

The only place where Wireshark falls entirely flat is on criterion #5. The interface provides no way to interpret, label or group the unknown bytes of a packet, making it very difficult to use when a protocol is not supported. Currently the only way to add new protocols to Wireshark is to write code in C or Lua, which requires a great deal of additional knowledge on the part of the user. Fortunately, Wireshark supports over 1000 protocols in its latest release, so hopefully the number of missing protocols is low.

3.3 New Interface

Wireshark's new, Qt-based interface (Figure 2) has a core layout which is clearly derived from the old GTK-based interface. It also inherits much of the old interface's behaviour, as the underlying engine has received only incremental improvements between the two versions. However, there are still several differences which promise improved usability.

The first and most obvious change is that packet conversations are now visibly grouped in the packet list (see label `1` in Figure 2). This is quite difficult to see due to its small size and poor contrast, but the left-most column of the list now displays a bracket indicating which other packets are in the current conversation. If the currently selected packet includes links to other packets — for example the response corresponding to the current request — then those rows are marked with an additional dot. Unfortunately these interface elements are not currently interactive; if the user wishes to focus on a particular conversation they must filter it out some other way.

Another change, albeit not a visible one, is that individual expert information items can now be used as fully-fledged filterable fields in display filters, custom columns and colouring rules. In the previous version, the only expert-related filters were the three special ones, thus requiring the user to write regular expressions on the `expert.message` field to filter out specific errors. In this version, each individual error type is filterable, al-

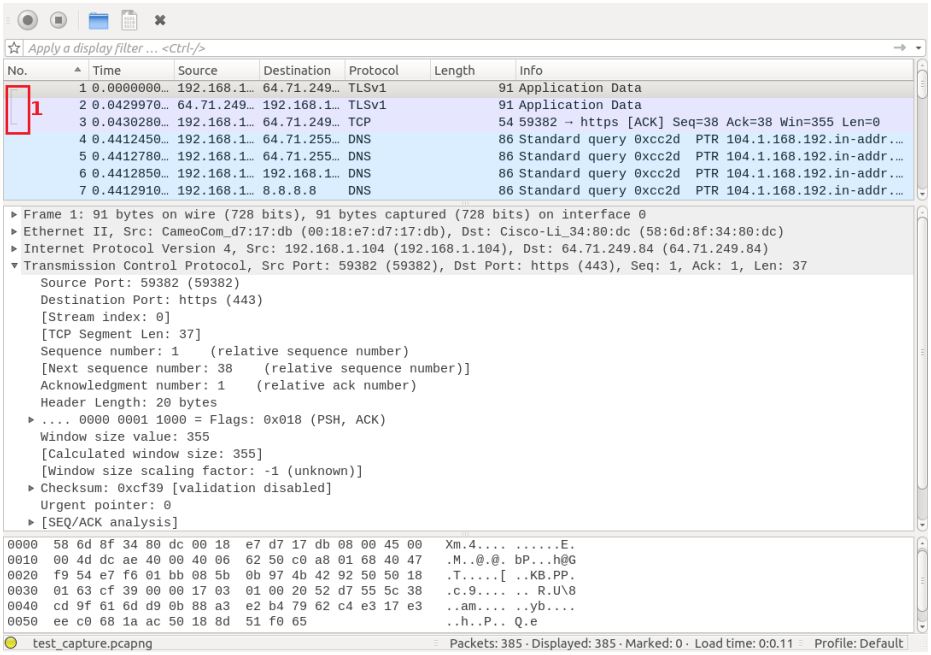

Figure 2: The Qt-based user interface of Wireshark under development.

lowing the user to filter directly on fields like `tcp.analysis.out_of_order`, for example[10]. Although technically an engine improvement and not an interface change per se, it substantially improves Wireshark's conformance to criteria #3 and #4.

The final apparent change in the new interface is technically more of an absence; many of the additional windows present in the old interface are currently unimplemented, including the list of endpoints, the list of conversations, and the list of expert information. This may just be a result of the new interface's incomplete status, but it means that there is substantial room to rethink the layout of these functions before implementing them in the new interface, particularly with respect to criterion #2.

3.4 Recommendations

Having evaluated Wireshark's two interfaces according a set of specific design criteria, we are now in a position to make several concrete recommendations for future Wireshark development.

The only criterion fundamentally unfulfilled by the interface was the fifth, dealing with unknown protocols, however this may not be as large a problem as it appears. Wireshark does provide the opportunity to code

[10] Certain protocol dissectors used to generate extra "fields" just to permit filtering on common expert information, so you *could* technically filter on `tcp.analysis.out_of_order` in the old interface, but it required additional work and an extra layer of indirection.

protocol plugins in two languages, and while this is not something we can reasonably expect from the average layperson, Wireshark's target user-base is substantially more technically savvy. Nevertheless, it would be highly beneficial if Wireshark provided basic functionality for interpreting and labelling unknown bytes.

In contrast, the fourth criterion was much better satisfied, however there is always room for improvement. Wireshark's filter engine would be much more useful if it supported comparisons across multiple packets, as it is currently very difficult to locate inter-packet constraint violations if they are not already marked with expert information. Additionally, it may be useful to permit basic operations for calculating new values out of the existing fields.

The third criterion was the most completely covered in both versions, and yet there was a peculiar lack of useful defaults. While the expert information framework was well-integrated and could be used in both columns and colouring rules, neither took advantage of it. The default columns relied on the protocol to add the appropriate summary information, a hit-and-miss situation at best, while the default colouring rules cherry-picked certain common errors instead of using more general filters. At the very least there should be a colouring rule to highlight packets that match `expert.severity == "Error"`.

The first two criteria dealt with the implementation and navigation of the abstraction hierarchy. While this was a more complex area to evaluate, two things stand out. First, the packet list and protocol tree should be capable of focusing on higher levels of abstraction instead of always being stuck at the lowest level of on-the-wire packet. Second, whatever design ends up being used to implement the conversation and endpoint functionality in the new interface should make every effort to provide a more unified hierarchy and a more discoverable and navigable interface.

The above recommendations range broadly in specificity and scope. Not all of these are easy, and some may be entirely unfeasible, but all of them would, in principle, contribute to Wireshark's usability according to Ecological Interface Design.

4 Conclusion

Ecological Interface Design was originally created and used for the interfaces of complex industrial systems like nuclear power plants [6], however we have shown in this paper that its underlying principles have applications to more traditional software interfaces as well. We have used these principles to develop structures for evaluating packet analysis software, and have analyzed two versions of the Wireshark application's user interface. In addition, we have been able to make concrete recommendations for future Wireshark interface development.

There are many opportunities for future work in this area. Making predictions is one thing, but testing them is quite another: future work could involve implementing and empirically evaluating some of the changes suggested in this paper. If this proves successful, there are many

other applications in various fields that might benefit from the EID approach. Additionally, there are several commercial applications targeting the same problem domain; evaluating them according to the criteria developed in this paper and comparing them to Wireshark would be a useful exercise.

Acknowledgements

Many thanks are due to Laura Chappell, Betty DuBois and Guy Harris who provided me with first-hand accounts of historical network sniffer and analysis tools. Thanks are also due to Wireshark developer Chris Maynard, who reviewed this paper and pointed out several mistakes with respect to Wireshark's design and behaviour. Any remaining inaccuracies are entirely my own. Finally, I wish to thank Dr. Robert Biddle who taught the course for which this paper was written.

References

[1] C. Borst, H. C. H. Suijkerbuijk, M. Mulder, and M. M. Van Paassen. Ecological interface design for terrain awareness. *The International Journal of Aviation Psychology*, 16(4):375–400, 2006.

[2] N. Dinadis and K.J. Vicente. Ecological interface design for a power plant feedwater subsystem. *IEEE Transactions on Nuclear Science*, 43(1):266–277, 1996.

[3] Pierre Duez and Kim J. Vicente. Ecological interface design and computer network management: The effects of network size and fault frequency. *International Journal of Human-Computer Studies*, 63:569–586, 2005.

[4] G.A. Jamieson, C.A. Miller, W.H. Ho, and K.J. Vicente. Integrating task- and work domain-based work analyses in ecological interface design: A process control case study. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 37(6):887–905, 2007.

[5] Jens Rasmussen. Skills, rules, and knowledge; signals, signs, and symbols, and other distinctions in human performance models. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(3):257–266, 1983.

[6] Kim J. Vicente and Jens Rasmussen. Ecological interface design: Theoretical foundations. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(4):589–606, 1992.