Automatically Extracting Fields from Unknown Network Protocols

Karthik Gopalratnam[1,2] karthig@cs.washington.edu
Sumit Basu[1] sumitb@microsoft.com
John Dunagan[1] jdunagan@microsoft.com
Helen J. Wang[1] helenw@microsoft.com

Microsoft Research (Redmond, WA) , [2]University of Washington (Seattle, WA)

ABSTRACT

There are thousands of network protocols in active use on the internet. System administrators often need to extract information from particular fields in such protocols without having sufficient information or time to programatically parse the packets. We propose an active learning framework to perform this extraction in an unknown protocol, in which the user presents the system with a small number of labeled instances. Our system then automatically generates an abundance of features and negative examples; we then use a boosting approach for feature selection and classifier combination. The system then displays its results for the user to correct and/or add new examples and iterate. In our preliminary experiments on DNS queries and responses, we achieve less than 0.1% generalization error using only a handful of labeled examples and thus a minimum of user effort. This translates to perfect retrieval from 86% of unlabeled packets.

KEYWORDS: network management, protocol analysis, active learning, boosting.

1. INTRODUCTION

Machines communicate with each other through a wide variety of network protocols. Indeed, there are a vast number of actual protocols that are in use at any given time. This number is magnified by the variety of implementations and version of each protocol. A quick survey in just five minutes of traffic for two buildings at Microsoft Research, there were more than 50 unique protocols. This diversity of protocols creates a huge headache for the network or systems engineer. For instance, if she's only interested it knowing what filenames are being transported via a filesharing application, perhaps to check for illicit activity, she has to find the specification for that protocol and then write code to decode the packet according to the specification. If said specification does not exist, she is forced to reverse engineer the protocol. While there are tools like Ethereal [4] that can understand many protocols, there are so many variations and subprotocols that it is difficult for such a tool to cover them all. Furthermore, extracting the field can be far from trivial: even relatively simple protocols have variable length fields, such as strings, which means the field of interest can be an arbitrary offset within the packet or TCP flow. In this paper, we examine a UDP-based protocol where each message is contained in a single packet, but our method applies to TCP flows as well.

Our work addresses precisely this scenario. Because network engineers often only want to extract a particular field from a protocol, a full decoding of the protocol is neither practical nor necessary. We thus propose a method in which we attempt to directly target the field of interest by learning a classifier. There is an additional twist, though, to this scenario – the engineer wants to do as little labeling as possible, and is unlikely to be willing to label a large number of packets. As a result, it is undesirable to use the typical supervised learning approach, i.e., batch learning a complex classifier on a fixed training set.

To deal with these constraints, we propose an active learning framework. The system works with as little or as much data as the user is willing to give, often beginning with only one labeled example. Given a starting point of a dataset of packets from the protocol of interest and at least one packet with the field(s) of interest labeled, the system automatically generates negative examples, since all other subranges in that packet cannot be the field of interest. It then proposes dozens of simple candidate features that are automatically generated based on the positive examples, each of which results in a simple classifier; we then use a boosting approach [6] to do both feature selection and classifier combination. Upon convergence, the system returns a set of qualifying subranges based on the classifier it has learned. The user then adds labels by either marking the false positives from that set or adding more positive examples, at which point the system learns a new classifier and brings back new results; these iterations continue until the user is satisfied or the system is unable to improve.

This active learning and classifier combination approach is beneficial for three reasons. First, it requires less effort from the user, since she only has to label as many examples as it takes to get the performance she desires. Second, the complexity of the classifier scales with the number of labels. If the user labels one example and one of the simple features is

sufficient to consistently return the field, only that one feature will have significant weight. On the other hand, if the problem is more complex, more features will be used in the final classifier. In contrast, a classifier of fixed complexity typically requires a fixed amount of labeling to avoid overfitting. Finally, if the network engineer speculates that new families of features that might be useful to a given task, she can add these to the bag without risking the error performance, even if the features are completely useless. Indeed, the bag of possible features can keep growing risk-free over time as people add custom features or feature families, making new problems easier to solve.

In the following sections, we describe the details of our approach to the field extraction problem and show our preliminary results. While our efforts are not complete, and we currently have only implemented the first iteration of the active learning process, the results are promising, and imply that with additional iterations we can achieve near-perfect extraction on typical fields.

2. RELATED WORK

There has been a variety of work on analyzing and parsing network protocols with varying degrees of automation. The work of Borisov et al. [3], GAPAL, is a general framework in which network protocols can be specified and parsed via hand-written grammars. This allows for precise recovery of all protocol elements, but requires significant manual effort to precisely describe the entire protocol. On the other hand, there have been fully automatic approaches like the RolePlayer system [2], in which the authors automatically modeled the server responses for various protocols to fool attackers into attempting to exploit vulnerabilities. In a similar vein is the work of Ma et al. [5], which attempts to classify packet streams into protocols based on learned Markov models. While these more automatic methods are able effectively to fool attackers and identify protocols, the models they learn are not sufficiently specific to extract arbitrary fields, leading us to our current work.

From the machine learning perspective, we draw our inspiration from the work of Tieu and Viola [1], in which the authors developed a system for image retrieval based on a small number of example images. For each query, they proposed a wide variety of features and used boosting to do classifier combination and feature selection. We take a similar approach to the field extraction problem, adapting the features to the space of network packets.

On the application side, our contribution has been to approach the field extraction problem directly – instead of trying to model the network protocols, we treat the problem as one of detecting special structures within unstructured data as in the Tieu and Viola work. On the learning side, we have not only applied the feature selection and combination approach to a new domain, but we have also incorporated it into an active learning framework, thus minimizing the amount of effort that the user must expend in order to achieve a sufficiently accurate classifier.

3. ACTIVE LEARNING FRAMEWORK

In the most general sense, active learning scenarios are those in which the learner can query the user for labels or additional examples which will guide it towards a better solution. This paradigm is particularly valuable in situations such as ours where it is difficult or time-consuming for the user to label a large dataset prior to beginning the learning. The learner can do the best with the data it has available and then demonstrate its performance to the user. If the performance is not satisfactory, the system can ask the user to mark its mistakes or add more examples and thus improve its performance. In this section, we describe the particulars of our active learning framework for field extraction.

The user supplies the system with a small number of packets, possibly just one, within which she has specified the field of interest by means of an offset and a field length. In addition, she supplies a dataset of packets that are from the protocol of interest. This requires little user effort: simply specifying a server/port allows the system to cull the relevant data from the network.

The system then proposes a large set of features for the classification problem (see section 3.1 for details) and creates a decision stump for every feature. The decision stump is a very simple classifier – if the feature is Boolean, the decision stump is the feature itself; if the feature is real valued, the decision stump is a simple classifier that provides a mapping from $\Re$ to *{0,1}*, e.g., a two class Gaussian model as in [1].

The system also generates a large number of negative examples. This is a relatively easy task given that there is only one correct answer per packet, so all other offset/length pairs are guaranteed to be negative examples. In fact, there are too many to be able to use them all. However, in addition to examples drawn randomly from this set, we make sure to include some examples that draw particularly sharp contrast with the positive examples. Specifically, we can take the correct offset of a labeled packet but use an incorrect length, and similarly an incorrect offset but choose a length such that it terminates at the same location as the true field. These "almost correct" negative examples prevent the system from overfitting to characteristics of a small number of labeled examples. For example, if *byte[0]* of the field is always 255, the system may propose that this feature is enough to determine the field, regardless of the length – simply because it has

seen no negative examples where this feature is true and yet the field is not correctly extracted.

At this point, we begin the feature selection and combination process. In a typical boosting scenario, we would train the same classifier (e.g., a simple hyperplane) over and over again, reweighting the individual data points and learned classifiers on each iteration as in [6]. In our case, as in [1], we have a large number of classifiers to choose from, each based on a very simple feature. At each boosting step, we thus choose the *best* of these features/classifiers, i.e., the classifier that will most improve our classification accuracy on the weighted data. We then weight the resulting classifier as in standard AdaBoost [6]. This process thus performs both feature selection (by choosing the best classifier at each step) and feature combination (via the weighted combination from boosting). Due to the relative bias in the number of positive vs. negative examples, we scale the weight of the data points relative to the number of examples in each class.

The system then classifies all possible candidate packets and fields therein using the learned classifier, and returns the instances that are classified as correct fields. Note that this is an expensive operation – there are $O(N^2)$ possible fields in a packet of length N: all offsets and all lengths for each offset. For instance, identifying the field of interest from 10000 packets of average length 100 would involve $5*10^7$ classifications. We will discuss more efficient methods in Section 5.

The user then can then provide feedback on the extracted fields in two ways – she can provide negative examples (by identifying some of the extracted fields as wrong) or provide more positive examples if the recall rate is perceived to be too low. These active learning steps are repeated until the user receives satisfactory results.

3.1. Feature Generation

The learner's feature proposal process for field extraction is completely agnostic to the actual protocol. The proposed features can take a number of forms. The simplest of these is the case where each feature corresponds to whether or not a byte relative to the field location takes on a particular value. However, the naïve method of proposing these features is prohibitive, since that would involve proposing a separate feature for each byte taking on each value in the range {0…255}. Instead, we use the positive examples as a guide to propose these single byte features in the following manner. We first align the positive examples along the start of the field (see Figure 1).

We then use this alignment to propose features corresponding to each byte value within some window (+/-10 in our case) around the beginning of the field. For example, in Figure 1, we would propose features such as *"byte[offset-1] == 0"*, *"byte[offset-2] == 2"*, *"byte[offset-2] == 7"*, *byte[offset+1] == 48"* and so on, where *offset* (a variable) is the starting position of the field for the given packet. We then do the same for the end of the field. In our preliminary evaluation on the DNS protocol, we found that these simple features performed very well.

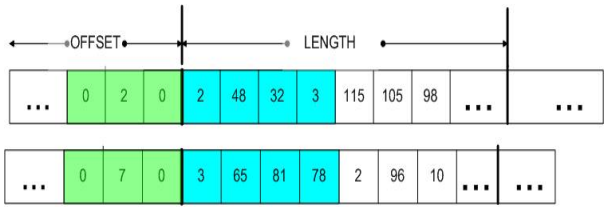

Figure 1: Aligning the positive examples along the beginning of the demarcated field in order to propose features. Note *offset* and *length* are both variables and are different for different packets.

In addition to these single byte equality features, there are a number of other features that can be proposed. We plan to explore this in the context of more complex protocols.

4. EXPERIMENTAL SETUP AND RESULTS

This section describes the preliminary evaluation of our system on extracting fields from the DNS protocol, using traces collected from a gateway router at a building on the Microsoft Redmond campus. This NetMon trace was collected over 5 minutes while traffic was flowing at 95% capacity on a 100Mbps link. We extracted all the DNS traffic (Port 53). We separated the DNS queries and responses and focused on determining if our system could reliably extract the *domain name* from DNS queries, and the *returned IP address* from the DNS responses.

DNS is a relatively simple protocol, but we argue that being able to extract these in a completely protocol agnostic fashion makes a compelling proof of concept for our active learning and classification system for the following reasons. The domain name in the DNS query is an example of a variable length field in a binary protocol, which makes it an interesting case, since a classifier has to somehow find the bounds of the field. In the case of the IP address in the DNS response, there can be multiple responses following the variable length query (the query is repeated in the response).

We chose to use just the simple byte-equality features based on lining up the positive examples (Section 3.1) for our classification. To evaluate of the classification algorithm, we created a test data set of both DNS queries and responses – each consisting of packets with the query and response IP fields marked out. We used Ethereal's parsing of the packets to label our test data.

4.1 Performance vs. Number of Training Examples

The first evaluation we made was to understand the effect of the number of positive and negative examples

on the accuracy of the classifier. This was measured by training the boosting algorithm on varying numbers of positive and negative examples, and testing them on a test set of 1,000 positive examples (a subrange corresponding to a field) and 1,000 negative examples (a subrange not corresponding to a field) from DNS queries. Figure 2 shows the percentage error of the classifier vs. the number of positive examples, averaged over 20 sets of 200 randomly selected negative examples. The plot shows the error rates on the positive and negative instances. In this experiment, the error drops to zero after labeling less than ten examples, which implies minimal effort from the user. We observed similar performance on the DNS responses. We also observed that the algorithm was not particularly sensitive to the number of random negative training examples, and that very similar curves were obtained on the testing set.

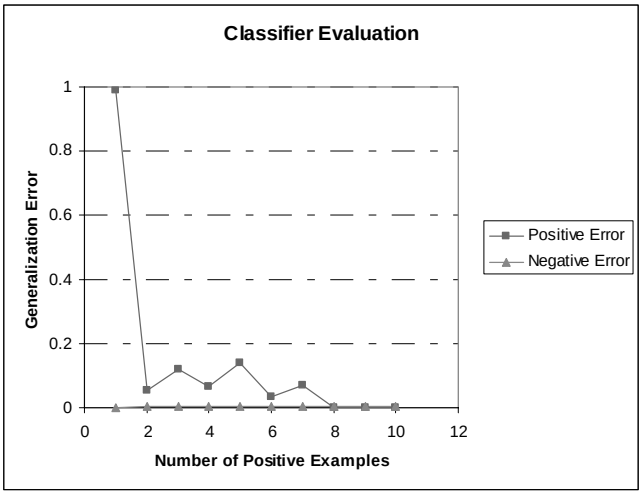

Fig. 2: Generalization Error for Boosting vs. number of positive examples on extracting DNS queries. The positive and negative test sets are the same size (1,000 of each).

4.2 Features Identified by the Classifier

Our next evaluation was to see which features were identified as important by the classifier, and we aimed to use our knowledge of the DNS protocol to evaluate whether the features with high weight made sense. For this evaluation, we ran the classifier on a set of 10 positive examples and 10 sets of 1,000 randomly generated negative examples, and made a list of the selected features and their average weight. Table 1 shows the top 6 features with the highest weight.

FEATURE	WEIGHT
byte[offset+length+5] == 1	4.52919
byte[offset – 7] == 1	3.304362
byte[offset+length-8] == 116	3.213337
byte[offset – 10] == 1	3.184982
byte[offset+7] == 73	1.851104
byte[offset + 26] == 67	1.483588

Table 1: Top 6 Most important features for identifying the DNS query

For instance, the above table indicates that the most important feature is that five bytes after the candidate field, the byte value is '1' and that seven bytes before the offset, the byte value is '1.' In reality, the DNS query always starts after the header, (which has a '1' seven bytes before the start of the query field), and the first '0' indicates the end of the query. However, several bytes after the final delimiting '0' are constant, and thus were effective in finding the end of the query, explaining the high weight for the first feature.

To understand why the final delimiting '0' was not a highly weighted feature, we constructed a histogram of the various byte values in DNS queries. This showed a disproportionate number of '0's. Therefore, with a relatively high probability, our randomly chosen negative examples contained instances where the byte after the candidate field was a '0', thereby lowering the weight of that feature during classification training. This does not particularly hurt us for the DNS queries/responses case, but for more complex protocols, it could be important to find the exact features that delimit the field of interest.

4.3. False Positives

The third evaluation was to understand the extent of false positives while actually searching through a set of given packets for the field of interest. To do this, we chose a set of 100 packets with the DNS query field. We then generated every possible candidate field (i.e. every possible combination of offset and length) for each of these packets, and using a learned classifier, classified each of these candidate fields. We then counted the number of packets for which the classifier correctly extracted (and only extracted) the query field. The evaluation was done for either two or four positive examples, and then repeated over 20 sets of 200 randomly selected negative examples. This establishes the extent to which the search for the field of interest among all the candidate packets is likely to yield false positives. Table 2 summarizes this result.

NUM. POSITIVE TRAINING INSTANCES	AVG. NUM. PACKETS CORRECTLY MARKED	MAX.. NUM. PACKETS CORRECTLY MARKED	MIN. NUM. PACKETS CORRECTLY MARKED
2	70	95	42
4	86	100	68

Table 2: Extent of False Positives: Packets with exactly one query field found during exhaustive enumeration of fields

We observed over multiple runs of the classification algorithm that there is a high degree of variance in the number of packets that are marked correctly. This is owing to the fact that the random set of negative training examples may not contain examples of the type that resolve the ambiguity that is reflected in these misclassifications. This underlines the promise of the active learning framework; we can rectify this by

adding the misclassified fields as negative examples for the next active learning iteration. However, we may have to ascribe a greater weight to the user-labeled negative examples relative to the randomly generated negative examples, since otherwise these specific examples will not generate enough of a penalty for misclassification during training, and therefore be effectively ignored. In fact, we can iteratively increase the weights until the classifier either gets these examples correct or can no longer improve its performance. This is an extension to the classification algorithm that we have not fully implemented, and leave this for future work.

5. DISCUSSIONS AND FUTURE WORK

We have shown that our initial implementation of an active learning system for field identification is able to achieve reasonable success; it is still an open question as to how well it will generalize to other protocols. As we described above, though, it seems clear that by making better use of our negative examples, we can drive the errors down to still lower levels.

In our future work, we first want to evaluate more explicitly the advantages of active learning over batch labeling, as is traditional in supervised learning scenarios. We can do this by measuring performance against the number of examples a user has to label in rounds of active learning, as compared to the same number of (randomly selected) examples labeled as a batch. Furthermore, we would like to investigate how we can optimially choose *which* labels the system presents to the user for labeling, given that those that are the most ambigious (i.e., closest to the decision boundary) are the most likely to need user feedback.

We would also like to extend our system in a number of other directions. First, we would like to implement the "feature cascade" used in [1] to efficiently filter the input with successive passes of the component features instead of enumerating all possible subpackets as we described in Section 3. Second, we would like to model the cost to the user of labeling negative examples vs. adding more data, allowing us to optimize in terms of user effort as well as classifier performance. Finally, we would like to explore how to generalize our work as a means of automatically extracting the structure of a protocol – if we can learn extractors for a number of tokens in a protocol, it is likely we can build models for how these tokens relate to each other.

From an applications perspective, we would also like to explore the range of possible tasks for which our methods could be helpful. For instance, one may want to extract the filenames of files sent over the network to enforce company policies, debug distributed systems problems related to particular field values, track the content of traffic to particular servers to understand their usage for management purposes, or identify whether applications with known vulnerabilities are being used on the network. We are optimistic that our methods could be helpful in many such scenarios.

6. ACKNOWLEDGEMENTS

We would like to thank the reviewers for their helpful comments and suggestions.

7. REFERENCES

[1] Kinh Tieu and Paul Viola. "Boosting Image Retrieval." *International Journal of Computer Vision*. 56 (1), 2004.

[2] Weidong Cui, Vern Paxson, Nicholas Weaver and Randy H. Katz. "Protocol-Independent Adaptive Replay of Application Dialog." *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS)*. San Diego, CA, February 2006.

[3] Nikita Borisov, David J. Brumley, Helen J. Wang, John Dunagan, Pallavi Joshi, and Chuanxiong Guo, "Generic Application-Level Protocol Analyzer and its Language." *MSR Technical Report MSR-TR-2005-133*. February 2005.

[4] Ethereal: A Network Protocol Analyzer. http://www.ethereal.com/

[5] Justin Ma, Kirill Levchenko, Christian Kreibich, Stefan Savage, and Geoffrey Voelker. "Automatic Protocol Inference: Unexpected Means of Identifying Protocols." *UCSD Computer Science Technical Report CS2006-0850*. February 2006.

[6] Yoav Freund and Robert Schapire. "A Short Introduction to Boosting." *Journal of Japanese Society for Artificial Intelligence 14(5):*771-780. September, 1999