Network Traffic Classification for Cybersecurity and Monitoring

Luca Deri <deri@ntop.org>
@lucaderi

ntop

Who am I

- ntop founder (http://www.ntop.org): company that develops open-source network security and visibility tools:
  - ntopng: web-based traffic monitoring and security
  - nDPI: deep packet inspection toolkit
  - nScrub: software-based DDoS scrubber
  - n2n: peer-to-peer VPN
- Author of various open source software tools.
- Lecturer at the CS Dept, University of Pisa, Italy.

nDPI at FOSDEM ‘21

FOSDEM '21 About News Schedule Stands Volunteer Practical

Online / 6 & 7 February 2021 visit News Sponsors Contact

FOSDEM 2021 / Schedule / Events / Developer rooms / Network monitoring, discovery and inventory / Using nDPI for Monitoring and Security

Using nDPI for Monitoring and Security

nDPI in practice

Track: Network monitoring, discovery and inventory devroom
Room: D.network
Day: Saturday
Start: 16:20
End: 16:55
Video with Q&A: D.network
Video only: D.network
Chat: Join the conversation!

As most of modern traffic is now encrypted, deep packet inspection is becoming a key component for providing visibility in network traffic. nDPI is an open source toolkit able to detect application protocols both in plain text and encrypted traffic, extract metadata information, and detect relevant cybersecurity information. This talk shows how nDPI can be used in real life to monitor network traffic, report key information metrics and detect malicious communications.

The pervasive use of encrypted protocols and new communication paradigms based on mobile and home IoT devices has obsoleted traffic analysis techniques that relied on clear text analysis. DPI (Deep Packet Inspection) is a key component to provide network visibility on network traffic. nDPI is an open source toolkit designed to detect application protocols on both plain and encrypted traffic. it is also able to extract relevant metadata information including metrics on encrypted traffic for easy classification and accounting. This talk introduces nDPI, demonstrate how to use it in real life examples, and it presents how it can be effectively used not only for traffic monitoring but also in cybersecurity being it able to detect unusual traffic behaviour and security issues.

nDPI at FOSDEM ’22: Motivation

- For years most developers focused on efficient traffic/event capture and processing (DPDK, PF_RING, Netmap, eBPF…).
- Unfortunately, traffic analysis is often still limited to simple top/bottom X (elephants/mice) statistics.

- Goal of this presentation is to:
  - Introduce to various algorithms that can be used in real life to analyse traffic.
  - Show how nDPI has implemented them in order to be useful for live traffic processing rather than for offline analysis (post-processing) as most R/Python tools do due to their slow performance and inefficient implementation.
  - Use nDPI a a foundation layer for cybersecurity and traffic analysis applications.

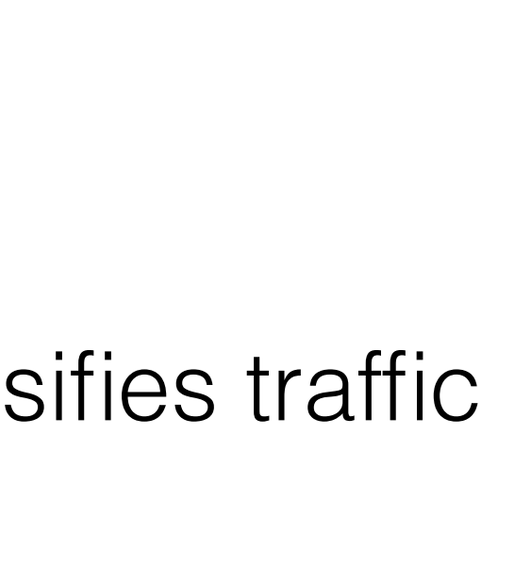
nDPI: A Recap

- nDPI is an open source toolkit that classifies traffic using DPI, deep packet inspection.

Layer 4 Protocol

Good or Bad?

TCP / HTTP 👍



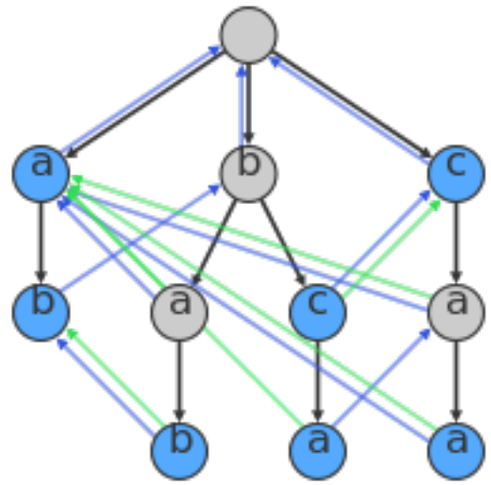
Layer 7 Protocol

String Searching: Aho-Corasick [1/4]

- Problem statement: substring matching (string that starts with or ends with) on a dictionary of strings (several thousand if not more) without doing a one-to-one comparison.

- Typical use cases:
  - List of domain/host names to match for exclusion (e.g. blacklist, spamming, advertisement etc) or traffic classification (e.g. all DNS queries for microsoft.com/windows.[net,com] -> DNS.Microsoft)

String Searching: Aho-Corasick [2/4]



- Aho-Corasick is a string searching algorithm that searches strings on a dictionary or words whose complexity is O (N + L) where N = length searched string, L =total length of the dictionary strings.
- It is based on automa that is built at runtime using the dictionary strings, i.e. if you need to add/remove a word a new automaton needs to be built (just do a hot swap to reload your data without stopping the application).
- The data structure has one node for every prefix of every string in the dictionary. So if (bca) is in the dictionary, then there will be nodes for (bca), (bc), (b), and (). If a node is in the dictionary then it is a blue node. Otherwise it is a grey node. [wikipedia]

String Searching: Aho-Corasick [3/4]

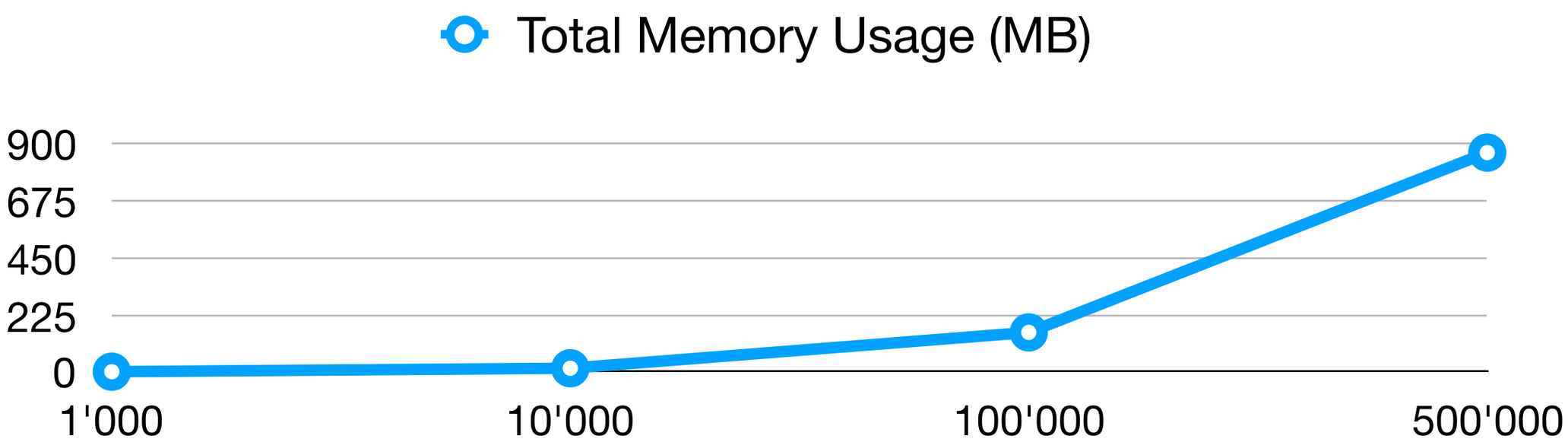
- nDPI implements a simple API for implementing string substring searching.

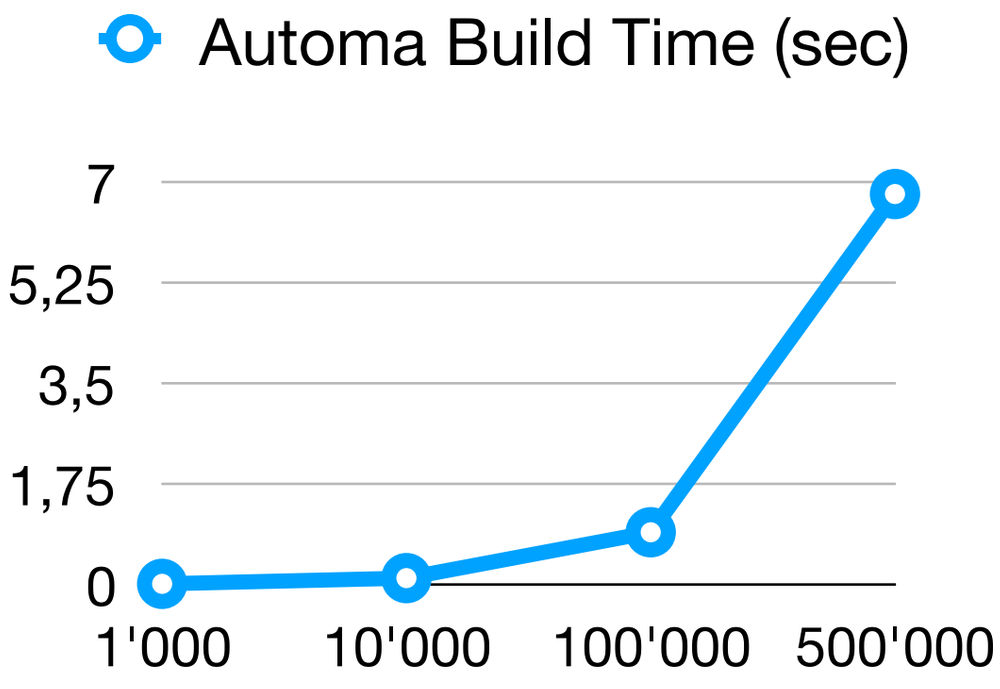
```
void automataUnitTest() {
  void *automa = ndpi_init_automa();

  assert(automa);
  assert(ndpi_add_string_to_automa(automa, strdup("hello")) == 0);
  assert(ndpi_add_string_to_automa(automa, strdup("world")) == 0);
  ndpi_finalize_automa(automa);

  assert(ndpi_match_string(automa, "This is the wonderful world of nDPI") == 1);
  ndpi_free_automa(automa);
}
```

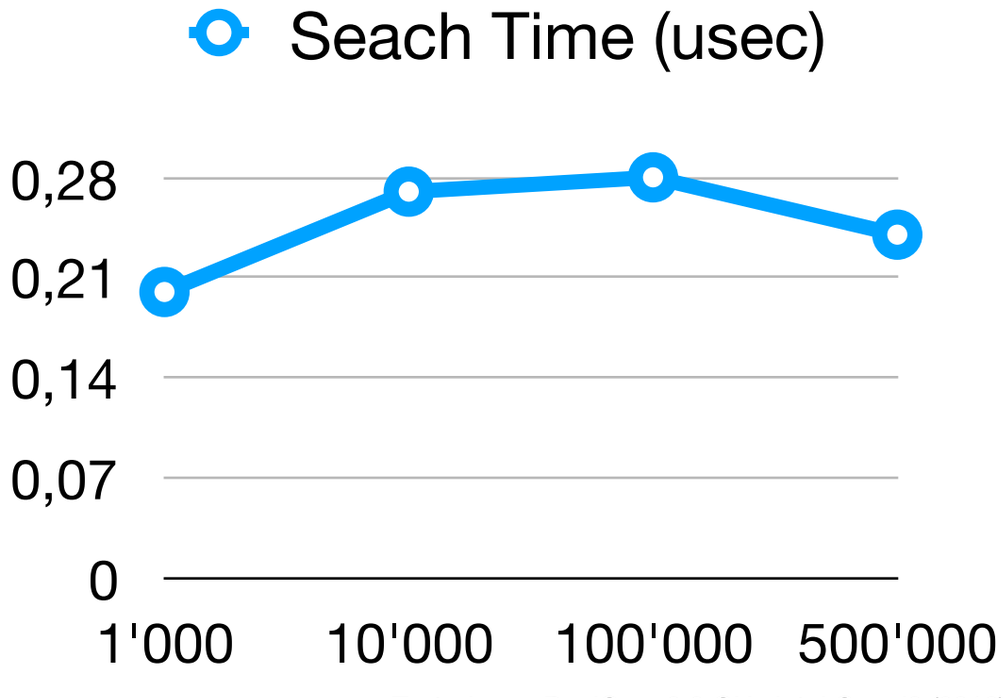
Check if the string to match contains any of the dictionary strings

- nDPI's implementation (based on an existing open source implementation significantly modified) supports "end with" (e.g. "hello$") that is useful when matching domain names that need to end with a prefix and avoid unwanted middle-string matches.

String Searching: Aho-Corasick [4/4]



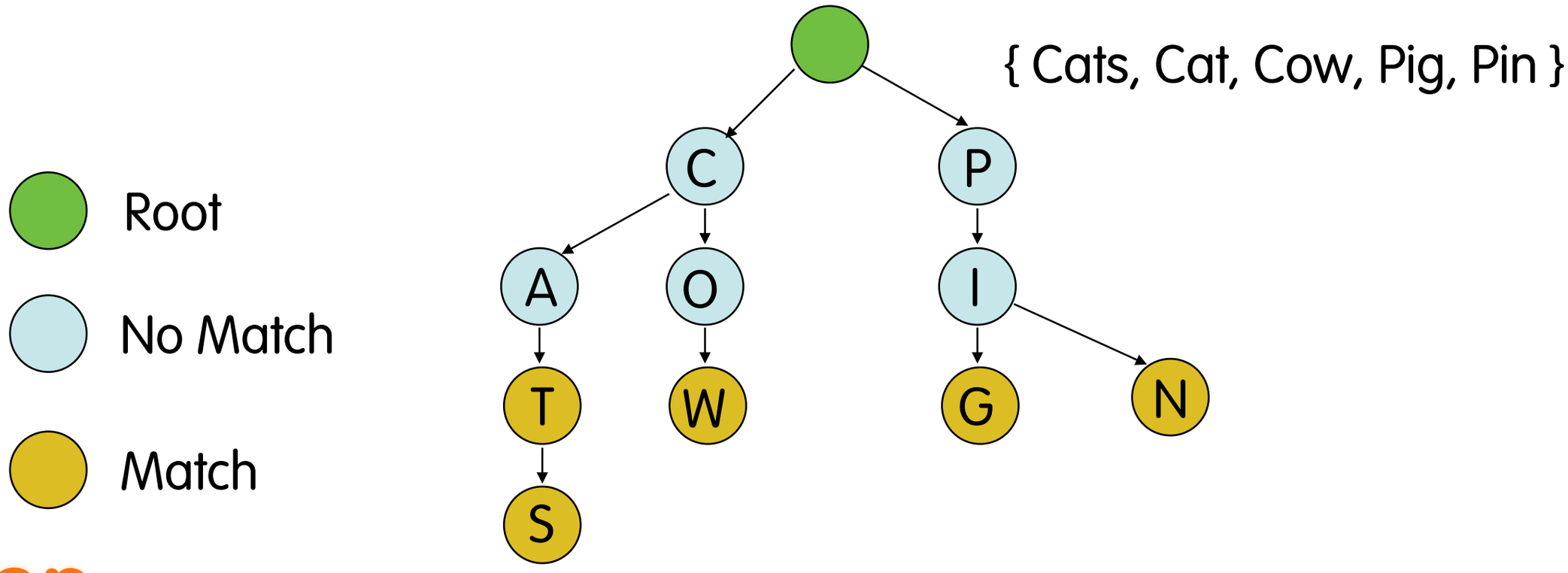




Tested on a DualCore 3,2 GHz Intel Core i3 (2010)

IP Matching: Radix Tree [1/5]

- A trie (pronounce as try, “pun on retrieval and tree”) is tree not based on comparisons (<,>)
  - Each node has a letter and a “marker”.
  - In case of multiple options per letter a list is used for each possible tree branch.



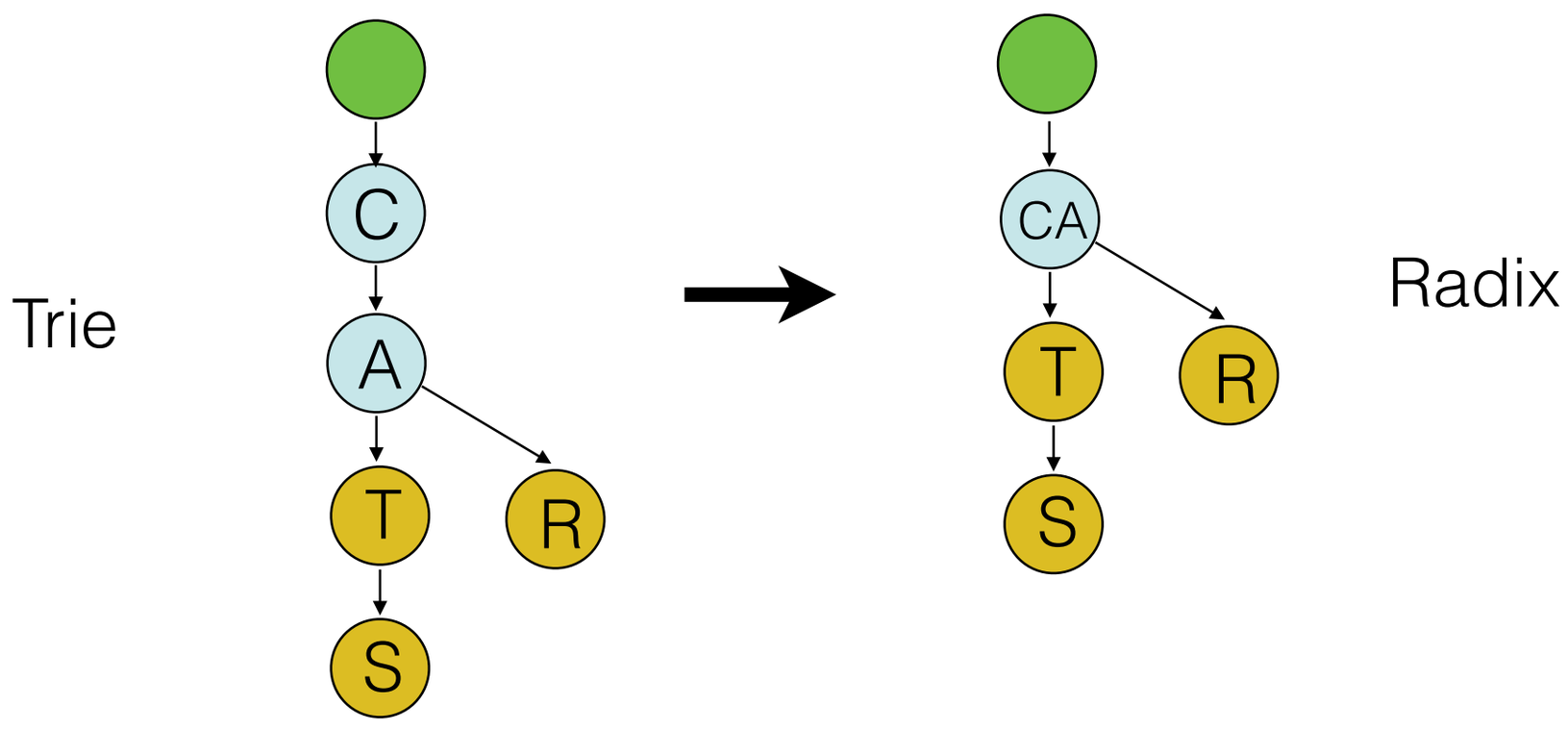
IP Matching: Radix Tree [2/5]

- In tries nodes can be added/removed/searched.
- Features
  - Ability to search strings starting with a given prefix.
  - Ability to generate string in dictionary order (if links in nodes are alphabetically sorted).
- Performance
  - insert O(w), where w is the length of the string to be inserted, regardless of the number of stored strings

Sounds familiar?

IP Matching: Radix Tree [3/5]

- From a trie to a radix tree: same as trie where nodes have a set of strings



IP Matching: Radix Tree [4/5]

- Patricia: Practical Algorithm to Retrieve Information Coded in Alphanumeric, D.R. Morrison (1968).
- Radix tree where numbers are used instead of strings.
- Use cases:
  - Efficient for subnet matching, IPv4/IPv6.
  - You can search partial matches (e.g. /24) and if you keep searching and found a match for finding a narrower match (e.g. /32).

IP Matching: Radix Tree [5/5]

```c
int main(int argc, char *argv[]) {
  ndpi_patricia_tree_t *p_v4;
  ndpi_prefix_t prefix;
  struct in_addr a;
  u_int16_t maxbits = 32; /* use 128 for IPv6 */
  ndpi_patricia_node_t *node;

  assert(p_v4 = ndpi_patricia_new(32));

  a.s_addr = inet_addr(line);
  ndpi_fill_prefix_v4(&prefix, &a, 32, maxbits);
  assert((node = ndpi_patricia_lookup(p_v4, &prefix)) != NULL /* node added */);

  a.s_addr = inet_addr("1.2.3.4");
  ndpi_fill_prefix_v4(&prefix, &a, 32, maxbits);
  node = ndpi_patricia_search_best(p_v4, &prefix));

  ndpi_patricia_destroy(p_v4, NULL);

  return(0);
}
```

You can add node “metadata”

```c
union ndpi_patricia_node_value_t {
  void *user_data;

  /* User-defined values */
  union {
    struct {
      u_int32_t user_value, additional_user_value;
    } uv32;

    u_int64_t uv64;
  } u;
};

typedef struct _ndpi_patricia_node_t {
  …
  union ndpi_patricia_node_value_t value;
} ndpi_patricia_node_t;
```

```
$ cd nDPI/tests/performance

$ ./patriciasearch
Patricia tree (IPv4) with 76378 IP prefixes built successfully in 0.05 sec [17.9 MB]
String searched in 0.10 usec
```

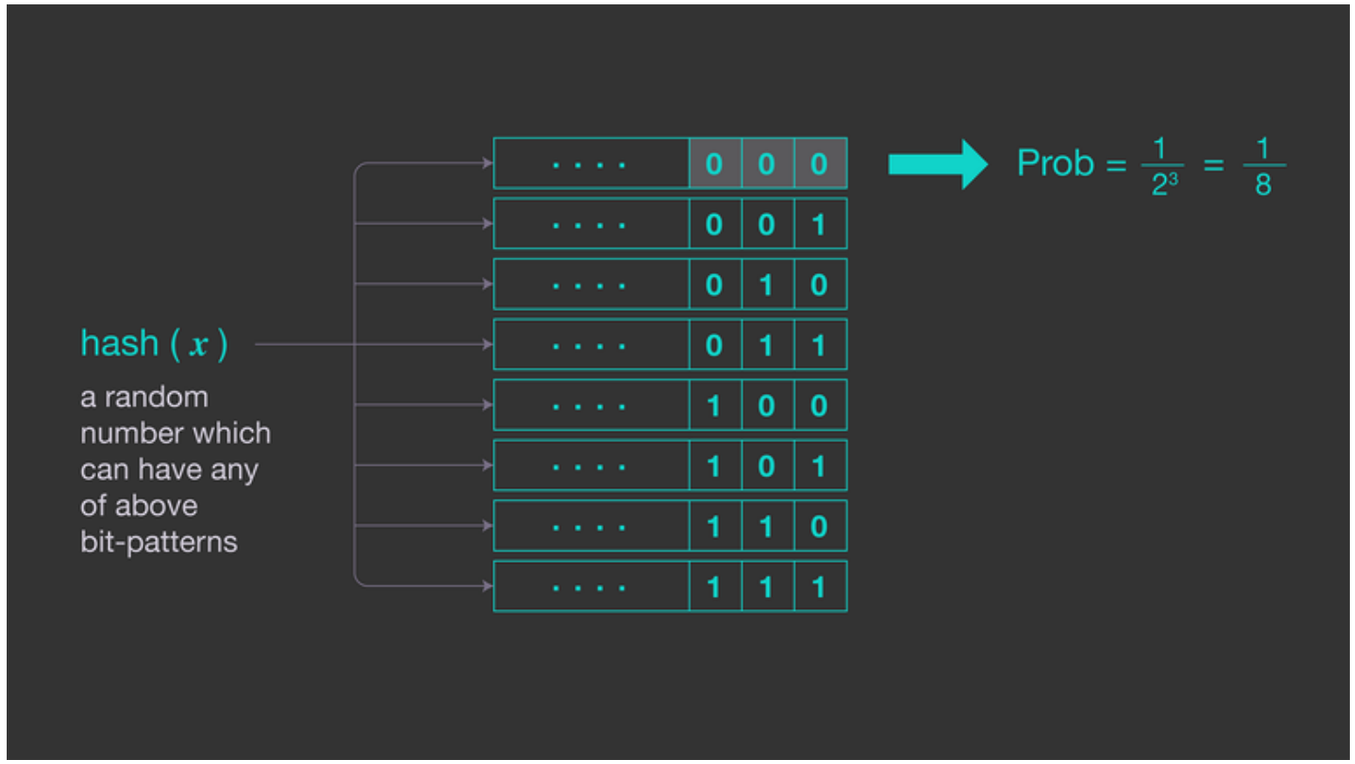
Tested on a DualCore 3,2 GHz Intel Core i3 (2010)

Probabilistic Counting: HyperLogLog [1/3]

- Problem Statement:
  - How can I get an estimate (i.e. approximate) of a number of unique set elements ? Of course you can do this in many ways (e.g. a hash table) but at a higher memory cost.
- Use Cases:
  - How many IP addresses has my host contacted in the past 5 minutes?
  - How many different IP countries has contacted host X ?
  - What is the host that has issues most different DNS host query names?

Probabilistic Counting: HyperLogLog [2/3]

- HyperLogLog is a probabilistic data structure used to estimate the cardinality of a set.
- It improves probabilistic counting by hashing every element, and counting the amount of 0s to the left of such hash.



HyperLogLog Paper: http://algo.inria.fr/flajolet/Publications/FlFuGaMe07.pdf

Probabilistic Counting: HyperLogLog [3/3]

- Usage

```
struct ndpi_hll hll_contacted_hosts, hll_contacted_countries;

asset(ndpi_hll_init(&hll_contacted_hosts, 8 /* i */) == 0);
asset(ndpi_hll_init(&hll_contacted_contries, 8 /* i */) == 0);

ndpi_hll_add(&hll_contacted_hosts, hostname, strlen(hostname));
ndpi_hll_add(&hll_contacted_countries, country, strlen(country));

num_contacted_hosts = ndpi_hll_count(&hll_contacted_hosts);
num_contacted_countries = ndpi_hll_count(&hll_countries_contacts);

ndpi_hll_destroy(&hll_contacted_hosts);
ndpi_hll_destroy(&hll_contacted_countries);
```

Estimate

- Memory and Cardinality Error

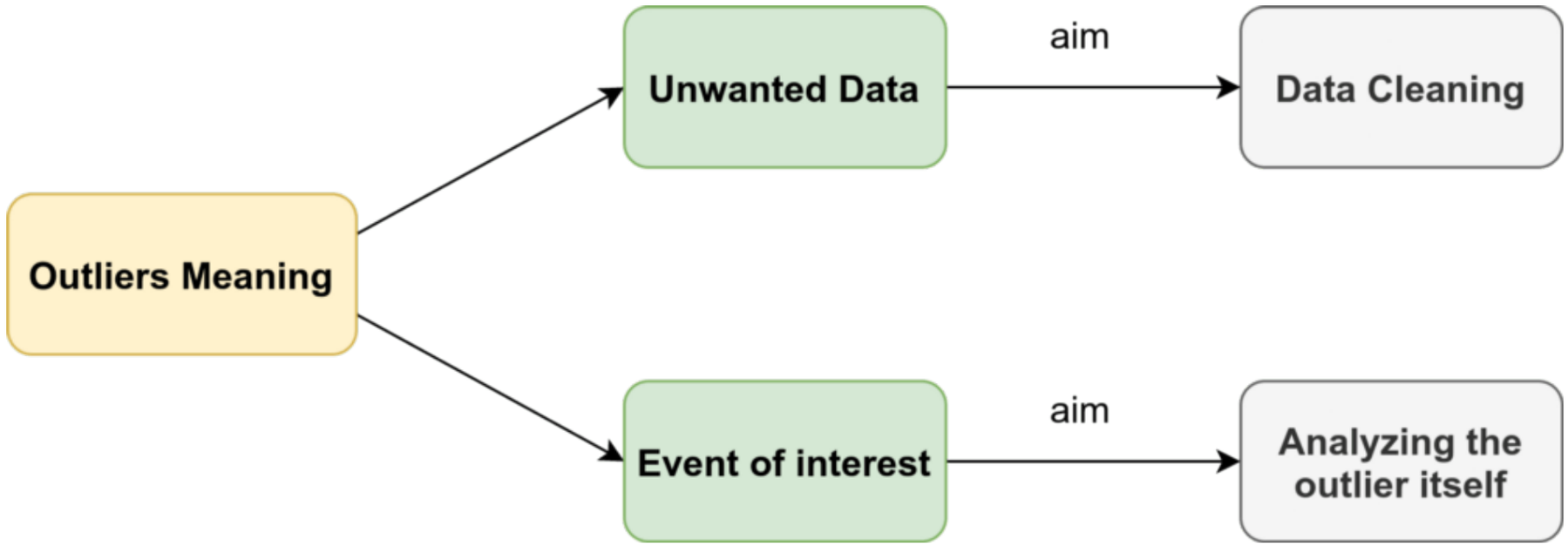
```
StdError = 1.04/sqrt(2^i)

[i: 4] 16 bytes      [StdError: 26%  ]
[i: 5] 32 bytes      [StdError: 18.4%]
[i: 6] 64 bytes      [StdError: 13%  ]
[i: 7] 128 bytes     [StdError: 9.2% ]
[i: 8] 256 bytes     [StdError: 6.5% ]
[i: 9] 512 bytes     [StdError: 4.6% ]
[i: 10] 1024 bytes   [StdError: 3.25%]
[i: 11] 2048 bytes   [StdError: 2.3% ]
[i: 12] 4096 bytes   [StdError: 1.6% ]
[i: 13] 8192 bytes   [StdError: 1.15%]
[i: 14] 16384 bytes  [StdError: 0.81%]
[i: 15] 32768 bytes  [StdError: 0.57%]
[i: 16] 65536 bytes  [StdError: 0.41%]
[i: 17] 131072 bytes [StdError: 0.29%]
[i: 18] 262144 bytes [StdError: 0.2% ]
[i: 19] 524288 bytes [StdError: 0.14%]
```

Anomaly Detection [1/10]

- Anomaly: an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism.

Anomaly Detection [2/10]

- Finding anomalies is manyfold:



Anomaly Detection [3/10]

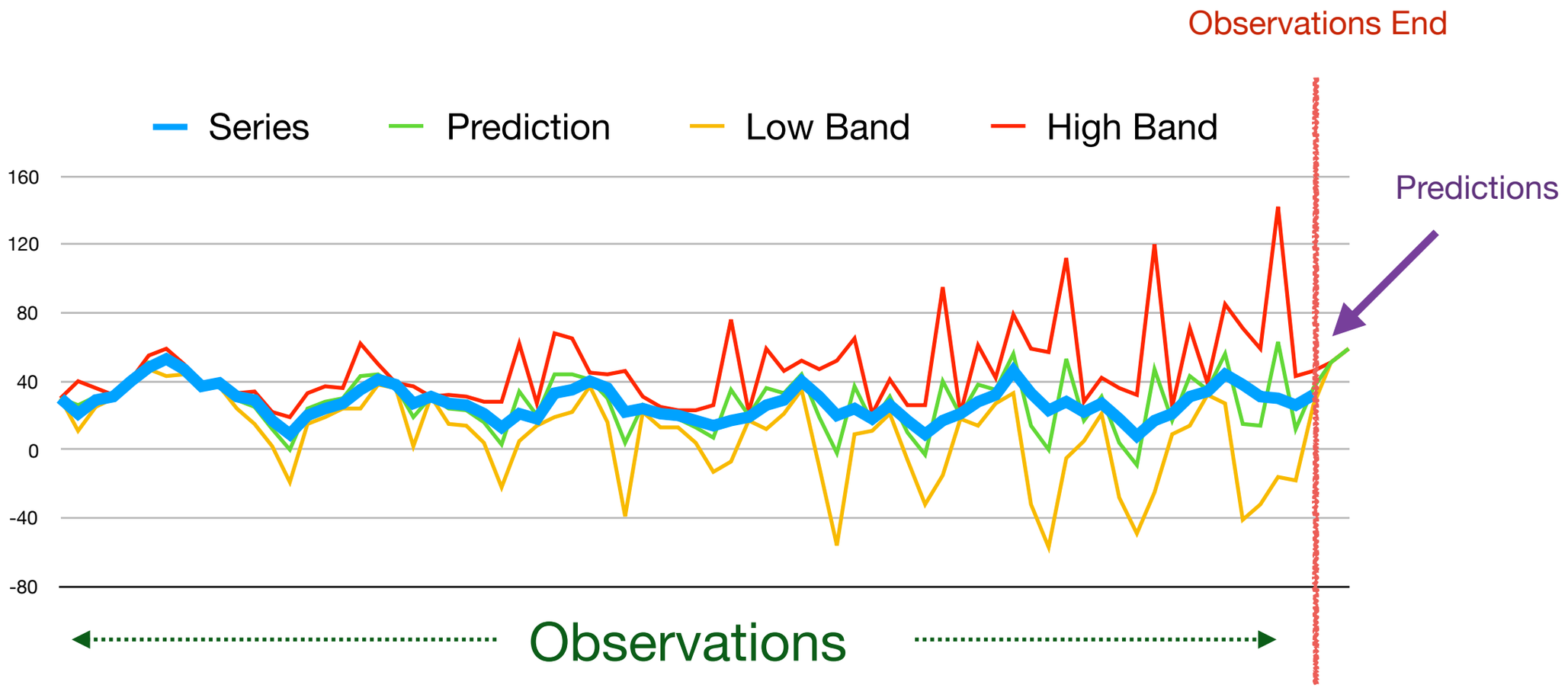
- Some definitions:
  - Series: an ordered sequence of numbers.
  - Order: the index of a number in the series.
  - Timeseries: a series of data points in time order.
  - Observation: the numeric value observed (in reality) at a specified time.
  - Forecast: estimation of an expected value (that we don't know yet) at a specific time.
  - Forecast Error: positive/negative difference of the observation with respect to the forecast. Usually the error is reported as square (SSE) i.e. the sum of squared errors of a series SUM((observation$_i$ - forecast$_i$) ^2)

Anomaly Detection [4/10]

- A time series is stationary when its statistical properties (e.g. mean and variance) do not change overtime, i.e. if they have no trend or seasonality.
- Counters are not stationaries, gauges are. Solution: store counters as value difference (observation(t) - observation(t-1)) rather than absolute values.
- Goal: Given a timeseries, we want to find anomalies by detecting those observations that fall outside of the expected low/high value forecast.
- Note: this technique complements static low/high threshold that are still recommended to have.

Anomaly Detection [5/10]

- In summary we need to implement a system that forecasts the next value and produces alerts.



Anomaly Detection [6/10]

- nDPI implements three “smoothing” functions for data forecast:
  - Single exponential smoothing: value
  - Double exponential smoothing: value + trend
  - Triple exponential smoothing (Holt-Winters): value + trend + seasonality.
  - Notes:
    - When a series is repetitive at regular intervals, it is defined seasonal.
    - Season Length: the number of data points in a season.

Anomaly Detection [7/10]

- Definitions:
  - α : Smoothing factor
  - β: Trend factor
  - γ: Seasonal smoothing factor
- All values are in 0..1 range: close to 1 means that recent values are more important than past values, close to 0 means that past values are more important than more recent ones.
  - Single exponential smoothing: α
  - Double exponential smoothing: α, and β
  - Triple exponential smoothing: α, β, and γ

Anomaly Detection [8/10]

- Exponential Smoothing API

```
/* Single Exponential Smoothing [60 bytes] */

int ndpi_ses_init(struct ndpi_ses_struct *ses, double alpha, float significance);
int ndpi_ses_add_value(struct ndpi_ses_struct *ses, const u_int64_t _value, double *forecast, double *confidence_band);
void ndpi_ses_fitting(double *values, u_int32_t num_values, float *ret_alpha);

/* ****************************** */

/* Double Exponential Smoothing [80 bytes] */
int ndpi_des_init(struct ndpi_des_struct *des, double alpha, double beta, float significance);
int ndpi_des_add_value(struct ndpi_des_struct *des, const u_int64_t _value, double *forecast, double *confidence_band);
void ndpi_des_fitting(double *values, u_int32_t num_values, float *ret_alpha, float *ret_beta);
```

```
/* ****************************** */

 /* Triple Exponential Smoothing (Holt-Winters) [172 bytes] */
int  ndpi_hw_init(struct ndpi_hw_struct *hw, u_int16_t num_periods, u_int8_t additive_seeasonal,
                  double alpha, double beta, double gamma, float significance);
void ndpi_hw_free(struct ndpi_hw_struct *hw);
int  ndpi_hw_add_value(struct ndpi_hw_struct *hw, const u_int64_t value, double *forecast,  double *confidence_band);
```

- Note
  - The process of finding the best value for α and β is named fitting.

Anomaly Detection [9/10]

```
struct ndpi_des_struct des;
 u_int8_t trace = 0;
 double v[] = {
   31.908466339111,
   87.339714050293,
   173.47660827637,
   213.92568969727,
   223.32124328613,
   230.60134887695,
   238.09457397461,
   245.8137512207,
   251.09228515625,
   251.09228515625,
   259.21997070312,
   261.98754882812,
   264.78540039062,
   264.78540039062,
   270.47451782227,
   173.3671875,
   288.34222412109,
   288.34222412109,
   304.24795532227,
   304.24795532227,
   350.92227172852,
   384.54431152344,
   423.25942993164,
   439.43322753906,
   445.05981445312,
   445.05981445312,
   445.05981445312,
   445.05981445312
 };
```

```
u_int i, num = sizeof(v) / sizeof(double);
float alpha = 0.9, beta = 0.5;
FILE *fd = fopen("/tmp/des_result.csv", "w");

assert(ndpi_des_init(&des, alpha, beta, 0.05) == 0);

if(trace) {
  printf("\nDouble Exponential Smoothing [alpha: %.1f][beta: %.1f]\n", alpha, beta);

  if(fd)
    fprintf(fd, "index;value;prediction;lower;upper;anomaly\n");
}

for(i=0; i<num; i++) {
  double prediction, confidence_band;
  double lower, upper;
  int rc = ndpi_des_add_value(&des, v[i], &prediction, &confidence_band);

  lower = prediction - confidence_band, upper = prediction + confidence_band;

  if(trace) {
    printf("%2u)\t%12.3f\t%.3f\t%12.3f\t%12.3f\t %s [%.3f]\n", i, v[i], prediction, lower, upper,
           ((rc == 0) || ((v[i] >= lower) && (v[i] <= upper))) ? "OK" : "ANOMALY",
           confidence_band);

    if(fd)
      fprintf(fd, "%u;%.0f;%.0f;%.0f;%.0f;%s\n",
              i, v[i], prediction, lower, upper,
              ((rc == 0) || ((v[i] >= lower) && (v[i] <= upper))) ? "OK" : "ANOMALY");
  }
}

if(fd) fclose(fd);

ndpi_des_fitting(v, num, &alpha, &beta); /* Compute the best alpha/beta */
```

Return code

0	**Too early: we're still in the learning phase.**
1	**Normal processing: forecast and confidence_band are meaningful**

Anomaly Detection [10/10]

```
Double Exponential Smoothing [alpha: 0.9][beta: 0.5]

Index    Value   Prediction   Upper        Lower
 0)       31.908 31.000        31.000       31.000  LEARNING [0.000]
 1)       87.340 81.400        73.637       89.163  OK [7.763]
 2)      173.477 166.360      156.529      176.191  OK [9.831]
 3)      213.926 213.844      205.290      222.398  OK [8.554]
 4)      223.321 227.213      218.717      235.709  OK [8.496]
 5)      230.601 232.954      224.846      241.062  OK [8.108]
 6)      238.095 239.399      231.821      246.976  OK [7.578]
 7)      245.814 245.714      238.608      252.819  OK [7.106]
 8)      251.092 251.424      244.719      258.129  OK [6.705]
 9)      251.092 251.804      245.424      258.185  OK [6.380]
10)      259.220 258.680      252.594      264.767  OK [6.086]
11)      261.988 261.312      255.482      267.142  OK [5.830]
12)      264.785 264.135      258.533      269.736  OK [5.602]
13)      264.785 264.356      258.955      269.757  OK [5.401]
14)      270.475 269.618      264.397      274.840  OK [5.222]
15)      173.367 183.016      175.969      190.063  ANOMALY [7.047]
16)      288.342 273.349      263.588      283.109  ANOMALY [9.760]
17)      288.342 288.975      279.479      298.471  OK [9.496]
18)      304.248 304.499      295.253      313.744  OK [9.246]
19)      304.248 305.827      296.780      314.874  OK [9.047]
20)      350.922 346.538      337.585      355.490  OK [8.952]
21)      384.544 382.767      374.005      391.528  OK [8.762]
22)      423.259 422.045      413.467      430.623  OK [8.578]
23)      439.433 440.802      432.374      449.231  OK [8.428]
24)      445.060 447.267      438.961      455.573  OK [8.306]
25)      445.060 446.893      438.717      455.070  OK [8.177]
26)      445.060 446.004      437.971      454.037  OK [8.033]
27)      445.060 445.463      437.573      453.353  OK [7.890]
```

Data Comparison: Binning [1/5]

- Data binning is a technique that allows data to be classified in a small number of “bins”, that in essence is a vector of positive numbers where each bin value contains the number of observations.
- Bins allow data to be classified using a small set of intervals instead of individual values that can lead to observation errors.
- Data is classified by
  - defining the bin number
  - adding data to the individual bins
  - normalising the data so that bins with different number of elements can still be compared.

Data Comparison: Binning [2/5]

- Bins do not store the data order (i.e. how the individual events happened) but just the data.
- Example: if you want to compare two hosts if they use similar protocols you can create a set of bins (e.g. 256 bins as the number of protocols recognised by nDPI) and for each new flow increase the bin-id that corresponds to the protocol. Then you can compare bins for equality to see what hosts are similar.

Data Comparison: Binning [3/5]

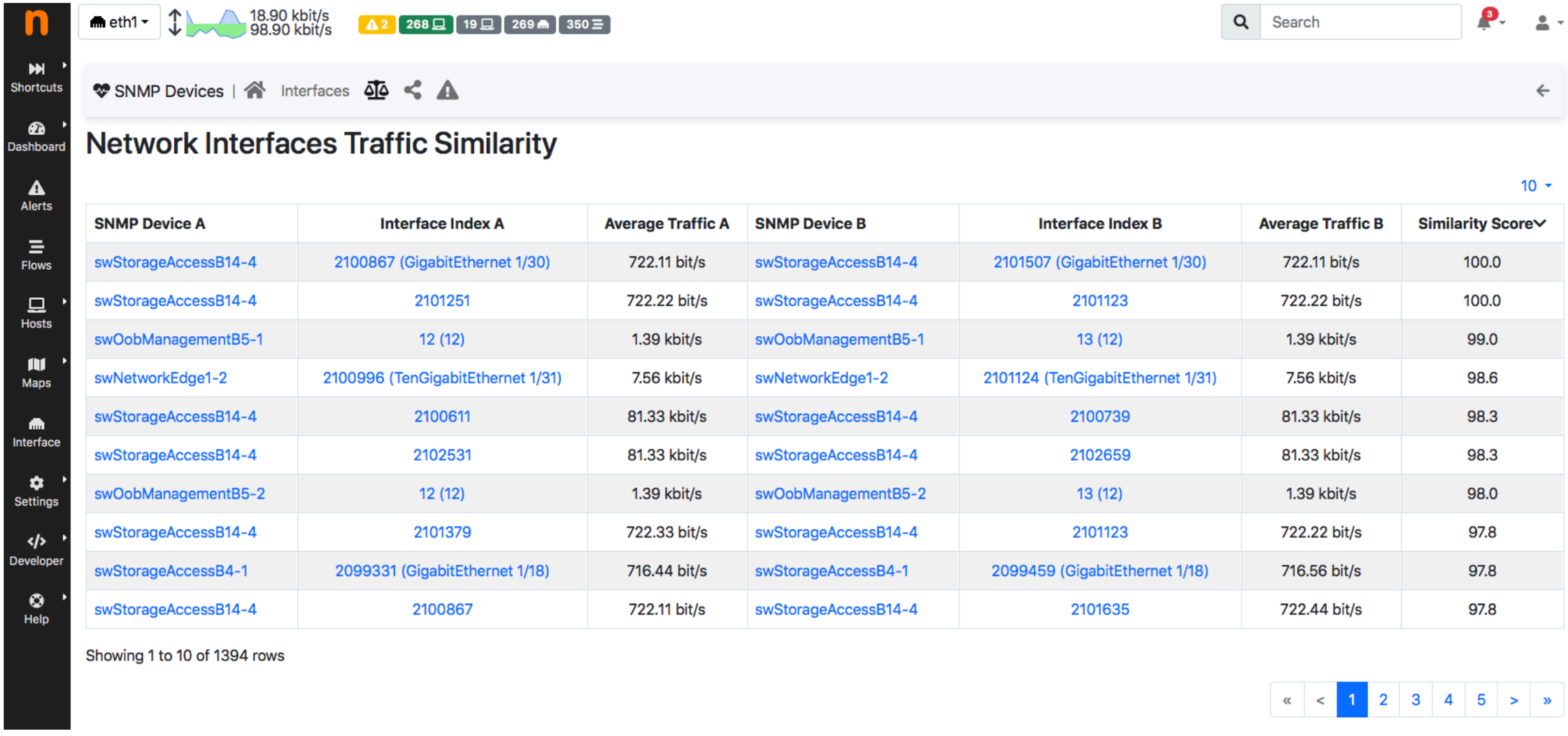
- Bins are an efficient way of storing observations but we need to find a way to compare them to find similarities (e.g. two hosts with the same behaviour).
- Use Cases:
  - Compare all hosts timeseries to find hosts that have a similar behaviour.
  - Compare two initial connection bytes sequence to see if they are similar.
  - Find hosts with the same packet size distribution. Note: packets lengths can be grouped in 6 bins of size <= 64 bytes, 65-128, 129-256, 257-512, 513-1024, 1025+.

Data Comparison: Binning [4/5]

```
void find_rrd_similarities(rrd_file_stats *rrd, u_int num_rrds) {
  u_int i, j, num_similar_rrds = 0, num_potentially_zero_equal = 0;

  for(i=0; i<num_rrds; i++) {
    for(j=i+1; j<num_rrds; j++) {
      /*
        Average is the circle center, and stddev is the radius
        if circles touch each other then there is a chance that
        the two rrds are similar
      */

      if((rrd[i].average == 0) && (rrd[i].average == rrd[j].average)) {
        if(!skip_zero)
            printf("%s [%.1f/%.1f]  - %s [%.1f/%.1f] are alike\n",
                   rrd[i].path, rrd[i].average, rrd[i].stddev,
                   rrd[j].path, rrd[j].average, rrd[j].stddev);

        num_potentially_zero_equal++;
      } else if(circles_touch(rrd[i].average, rrd[i].stddev, rrd[j].average, rrd[j].stddev)
                ) {
        float similarity = ndpi_bin_similarity(&rrd[i].b, &rrd[j].b, 0, similarity_threshold);

        if((similarity >= 0) && (similarity < similarity_threshold)) {
          if(verbose)
            printf("%s [%.1f/%.1f]  - %s [%.1f/%.1f] are %s [%.1f]\n",
                   rrd[i].path, rrd[i].average, rrd[i].stddev,
                   rrd[j].path, rrd[j].average, rrd[j].stddev,
                   (similarity == 0) ? "alike" : "similar",
                   similarity
                   );

          num_similar_rrds++;
        }
      }
    }
  }

  printf("Found %u (%.3f %%) similar RRDs / %u zero alike RRDs [num_rrds: %u]\n",
         num_similar_rrds,
         (num_similar_rrds*100.)/(float)(num_rrds*num_rrds),
         num_potentially_zero_equal,
         num_rrds);
}
```

https://github.com/ntop/nDPI/blob/dev/rrdtool/rrd_similarity.c

Data Comparison: Binning [5/5]



Additional nDPI Features

- Streaming Data Analysis

```
struct ndpi_analyze_struct* ndpi_alloc_data_analysis(u_int16_t _max_series_len);
  void ndpi_init_data_analysis(struct ndpi_analyze_struct *s, u_int16_t _max_series_len);
  void ndpi_free_data_analysis(struct ndpi_analyze_struct *d, u_int8_t free_pointer);
  void ndpi_reset_data_analysis(struct ndpi_analyze_struct *d);
  void ndpi_data_add_value(struct ndpi_analyze_struct *s, const u_int32_t value);

  /* Sliding-window only */
  float ndpi_data_window_average(struct ndpi_analyze_struct *s);
  float ndpi_data_window_variance(struct ndpi_analyze_struct *s);
  float ndpi_data_window_stddev(struct ndpi_analyze_struct *s);

  /* All data */
  float ndpi_data_average(struct ndpi_analyze_struct *s);
  float ndpi_data_entropy(struct ndpi_analyze_struct *s);
  float ndpi_data_variance(struct ndpi_analyze_struct *s);
  float ndpi_data_stddev(struct ndpi_analyze_struct *s);
  u_int32_t ndpi_data_last(struct ndpi_analyze_struct *s);
  u_int32_t ndpi_data_min(struct ndpi_analyze_struct *s);
  u_int32_t ndpi_data_max(struct ndpi_analyze_struct *s);
  float ndpi_data_ratio(u_int32_t sent, u_int32_t rcvd);
```

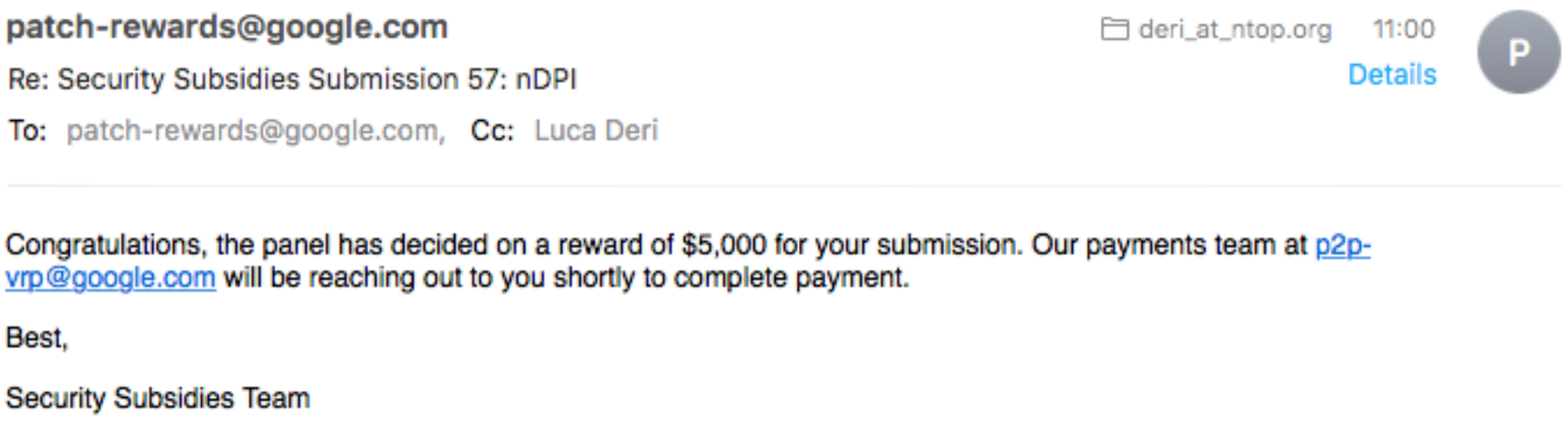
- Clustering (Unsupervised Machine Learning)

```
int ndpi_cluster_bins(struct ndpi_bin *bins, u_int16_t num_bins,
                      u_int8_t num_clusters, u_int16_t *cluster_ids,
                      struct ndpi_bin *centroids);
```

- Data Serialisation, Jitter/Entropy…..

Finally, Some Good News

- Recently Google awarded nDPI.
- (As soon as we receive the payment) We want to invest this money in nDPI development.
- Those interested to contribute to nDPI (being paid), are encouraged to contact us.



patch-rewards@google.com

deri_at_ntop.org 11:00

Details

P

Re: Security Subsidies Submission 57: nDPI

To: patch-rewards@google.com, Cc: Luca Deri

Congratulations, the panel has decided on a reward of $5,000 for your submission. Our payments team at p2p-vrp@google.com will be reaching out to you shortly to complete payment.

Best,

Security Subsidies Team

https://github.com/ntop/nDPI