A Novel Approach to Detecting Covert DNS Tunnels Using Throughput Estimation

by

Michael Himbeault

Context:
Motivation

- DNS tunnels uses:
  - Malware and botnet command-and-control channels
  - In/ex-filtration channels for transporting data through corporate security layers

- The ability to monitor the existence of these channels is important when securing a network.
  - Currently quite difficult

Context: Goals and Objectives

- Be able to identify DNS tunnels that do not violate DNS RFCs or specifications in near real time with high accuracy on commodity hardware.
  - Low false-positive rate

Background: Domain Name System (DNS)

- Translates text references to other forms of records. For example:
  - IP or IPv6 address (A or AAAA)
  - Another domain name (CNAME)
  - IP address to name (PTR)
  - Name to bulk data (TXT)

- Provides the human-interaction layer for the Internet
- Offers a great deal of flexibility for deploying automated services over existing infrastructure
  - For example: spam, malware, and address blacklists.

Background:
Covert Channels

- Utilize standard means of transportation in non-standard ways
  - Often transporting unintended data types over existing protocols
  - Occasionally involves new custom protocols built on existing ones
- Intention is rarely benign, often circumventing existing security layers

Background:
Covert Channels

- May or may not modify the standard protocols in ways that are conforming to specifications.
- May sacrifice ‘common’ features such as bidirectionality

- Examples:
  - IP timing channels
  - May use third party services such as Twitter, Facebook, or image hosting providers
    - Encoding information in JPEG headers
  - DNS tunnels

Background: Entropy

- Rough measure of the amount of information contained in a collection, *C:*

$$C = \{c_1, \ldots, c_n\}, P(c_i) = p_i$$

$$H(C) = -\sum_{i=1}^{n} p_i \log p_i$$

Background:
DNS Tunnels

- Existing implementations are commonplace
  - Iodine
  - OzymanDNS
  - Dns2tcp
  - DNScat
  - DeNiSe
  - PSUDP
- A custom implementation was built to simulate a next-generation tunnel

Background: DNS Tunnels

- Raw DNS tunnels
  - Utilize UDP/TCP port 53 for transmitting arbitrary data without respect for DNS protocol specifications
  - Not difficult to block
- Conforming DNS tunnels
  - Makes use of DNS packets that do not violate the protocol specifications to transmit arbitrary data
  - Can be very difficult to identify and block
  - The focus of this work

Custom DNS Tunnel Application

- Prototype proof-of-concept implementation
- Implements encoding to match character frequencies to circumvent detections approaches that tune to that statistic.
- Limited to client-to-server transfer only

State of the Art: DNS Tunnel Detection

- Fall into several categories
  - Signature based
  - Domain hash/blacklist
  - Flow data based
  - Character frequency analysis on queries
  - Behaviour of DNS queries on a per-domain basis
- The proposed approach falls into the last category

Proposed Approach: Assumptions

- DNS tunnels move more data than benign traffic

- Attempt to detect this increase in data transmission volume

Proposed Approach: Theory

- Collect DNS queries into temporal buckets
  - Ten-second windows were used in the analysis
- Further group queries by top-level domain (TLD)
  - google.com
  - cbc.ca
  - Etc…
- For each TLD (in the current window), compute a measure of how much data was transmitted

Proposed Approach: Measuring Data Volume

- Since common domains may appear more than uncommon, simple character count is insufficient
  - Modulo caching effects as described in section 5.1.3
- Average character count is similarly uninformative
  - Tunnels can use any length queries, including modelling a length distribution of legitimate traffic

Proposed Approach: Measuring Data Volume

- DNS tunnels can be expected to have very few queries that appear more than once
  - Since they are transmitting arbitrary data
- Benign domains can have many queries that appear a great number of times
  - Such as www.google.com

Proposed Approach: Domain Length-Weighted Entropy (DLWE)

- Consider the collection of queries to a TLD in an interval
- Treat each query as a symbol, and compute the entropy of the collection.
- Multiply the result by the average query length for the TLD in the interval.

- Expect large values for tunnel domains, small values for benign domains.

Evaluation:
Literature Candidates and Test Data

- Proposed approach was tested against candidates from the literature
  - *N*-gram detection proposed by Born
  - *Gzip* compression detection proposed by Paxson
  - Naïve counting of characters

- All approaches were implemented on a common Python framework
  - Analysis was done on approximately one billion UDP port 53 packets from a live ISP network as well as intentionally generated tunnel traffic

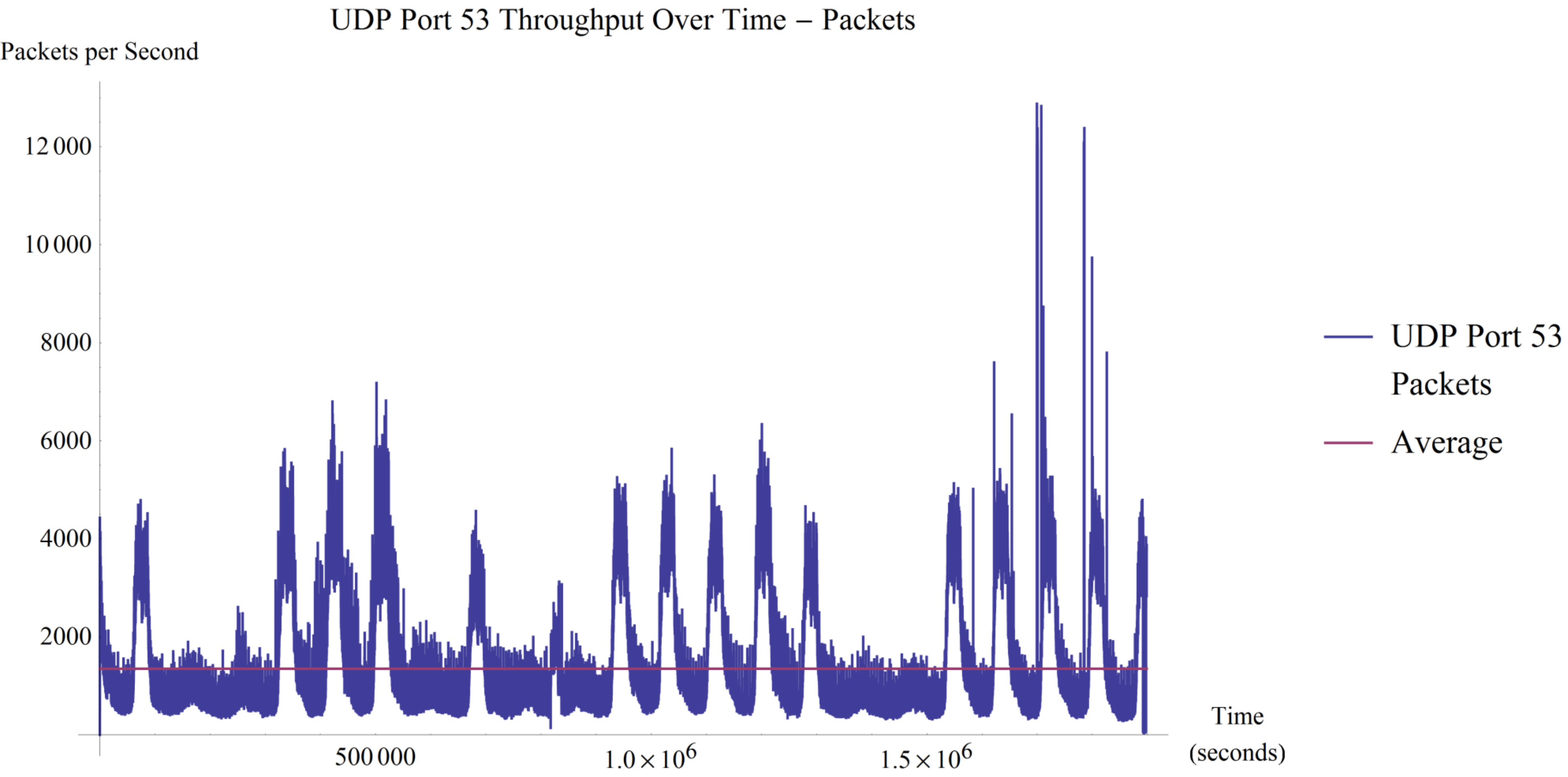
Evaluation Criteria

- Packet processing performance
  - A relative comparison, due to lack of optimized implementations no absolute target is chosen
- False positive rate
  - A relative ranking is employed
  - Intuitively, a false-positive rate of 1% will result in up to fifty alerts per second during average daytime traffic of the captured sample.

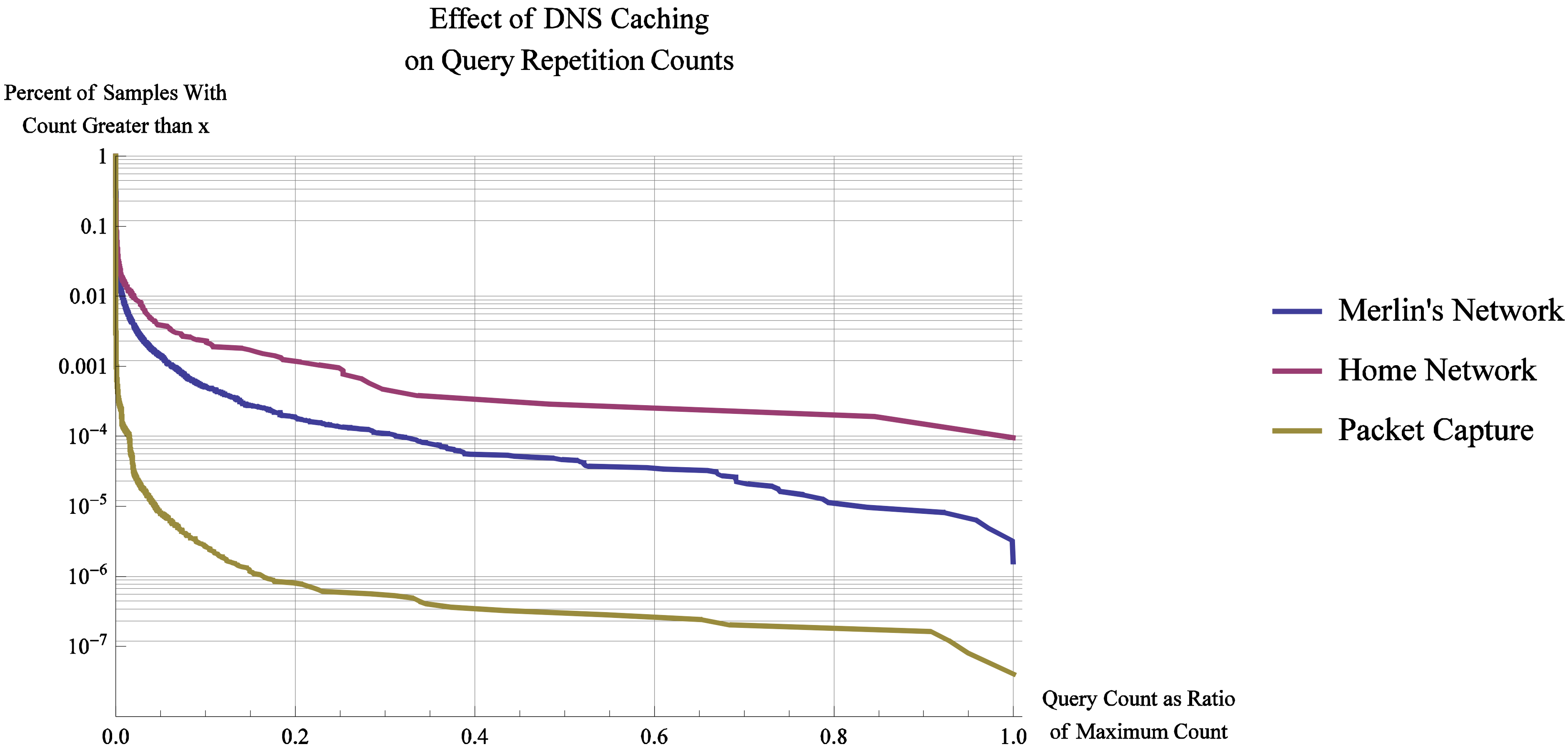
Sample Data

- May contain malicious traffic
  - In particular, DNS tunnels which will affect the false-positive rates of the detection methods.

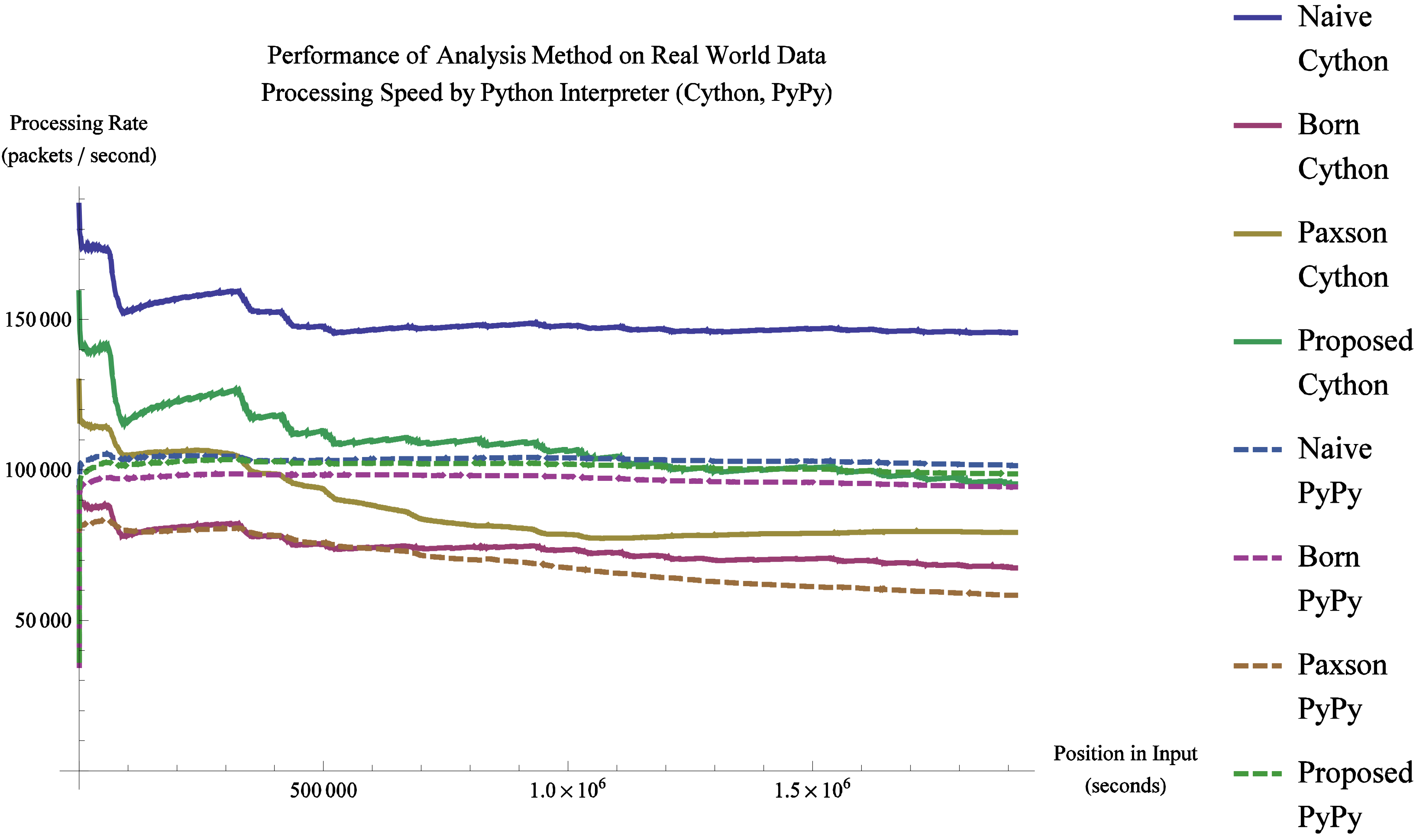
Sample Data: Packets per second



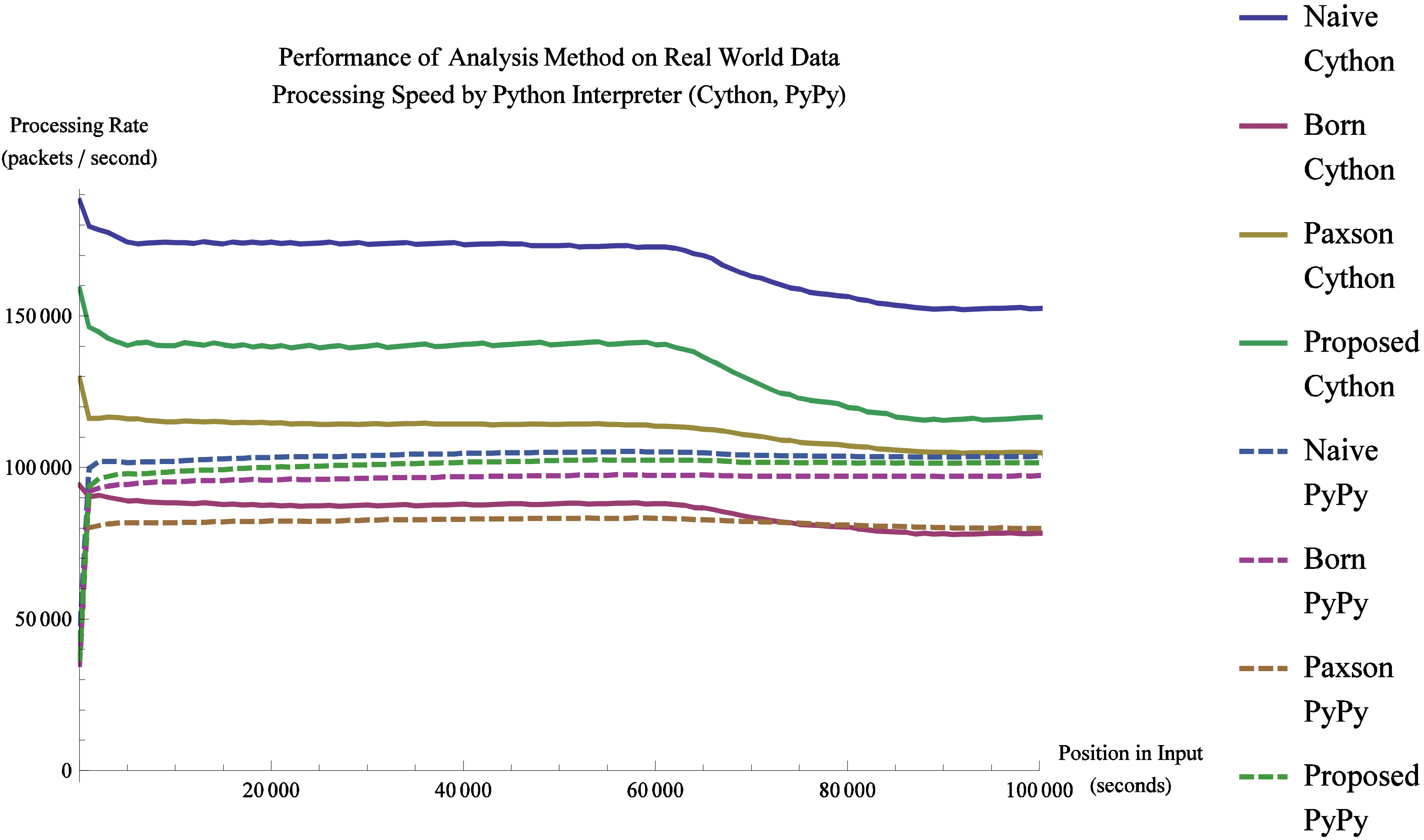
Confounding Factor: Effect of DNS Caching



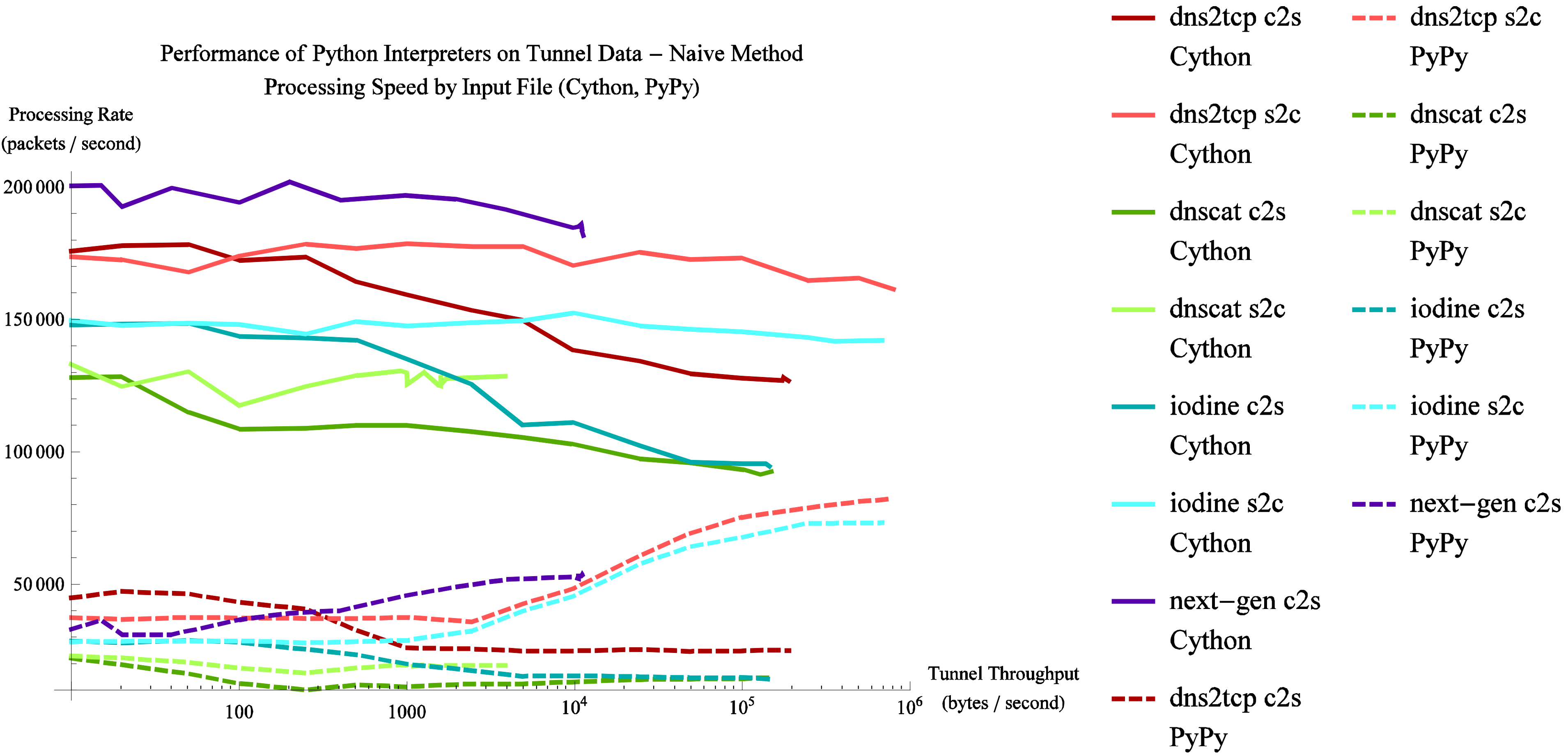
Processing Performance



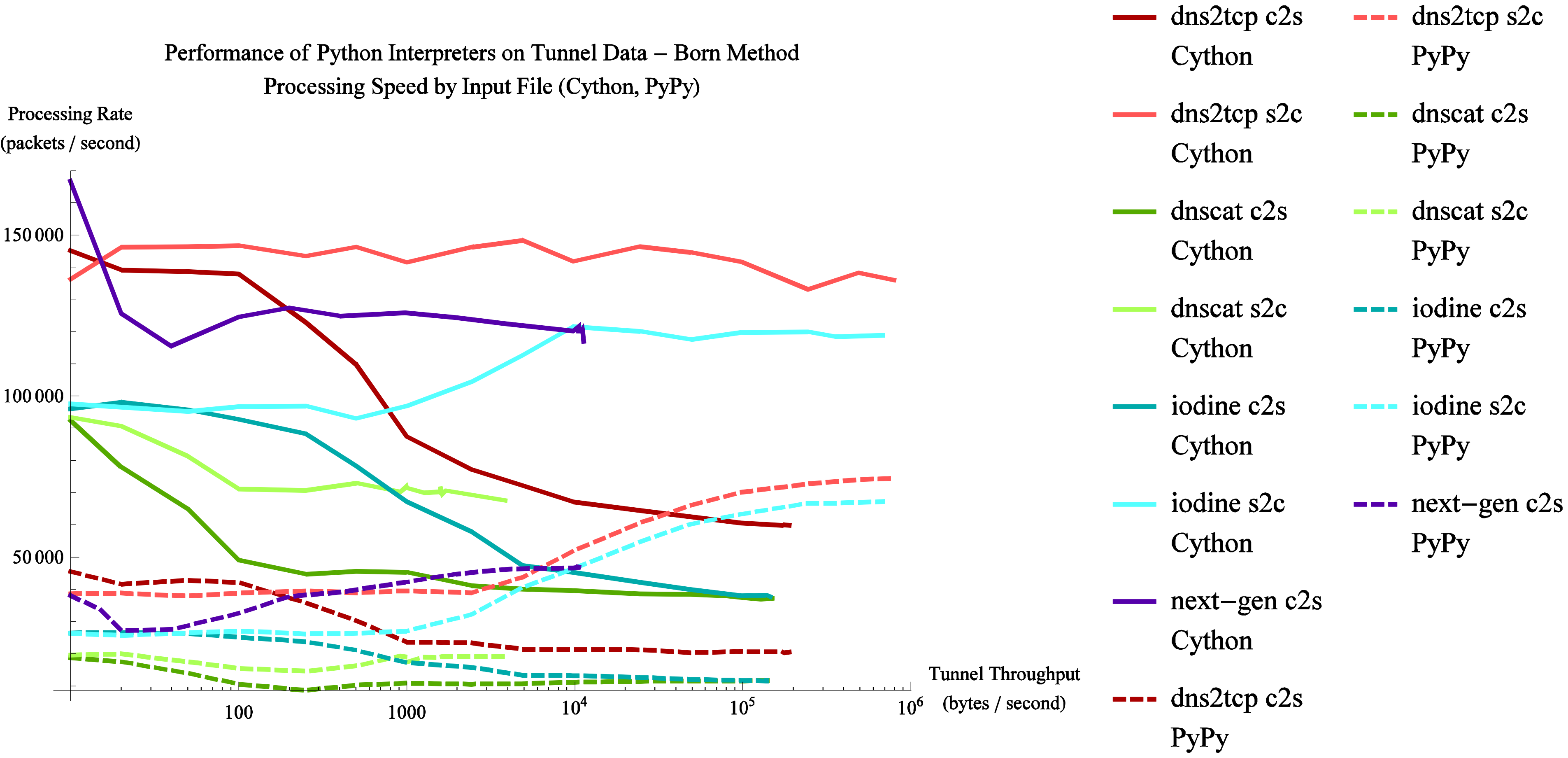
Processing Performance



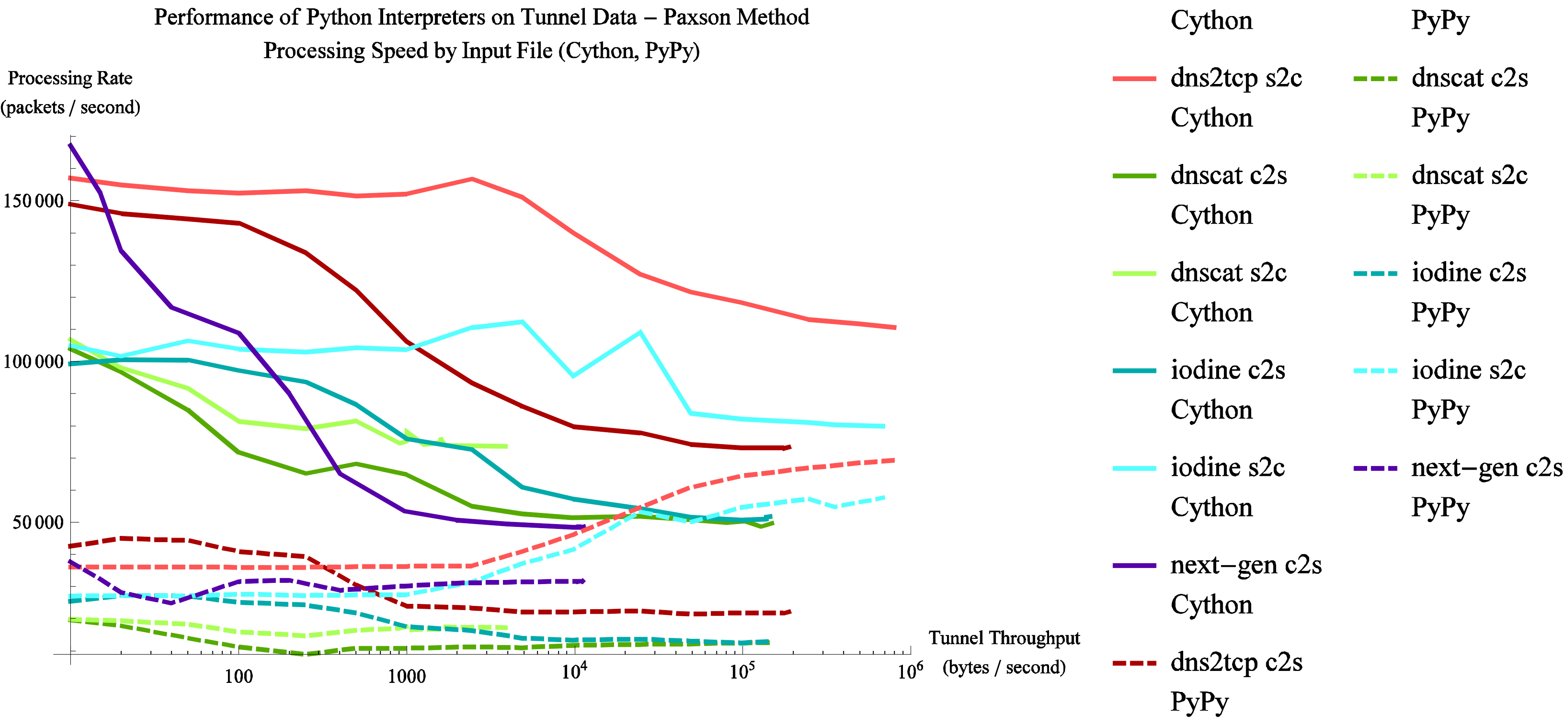
Processing Performance: Naïve Method



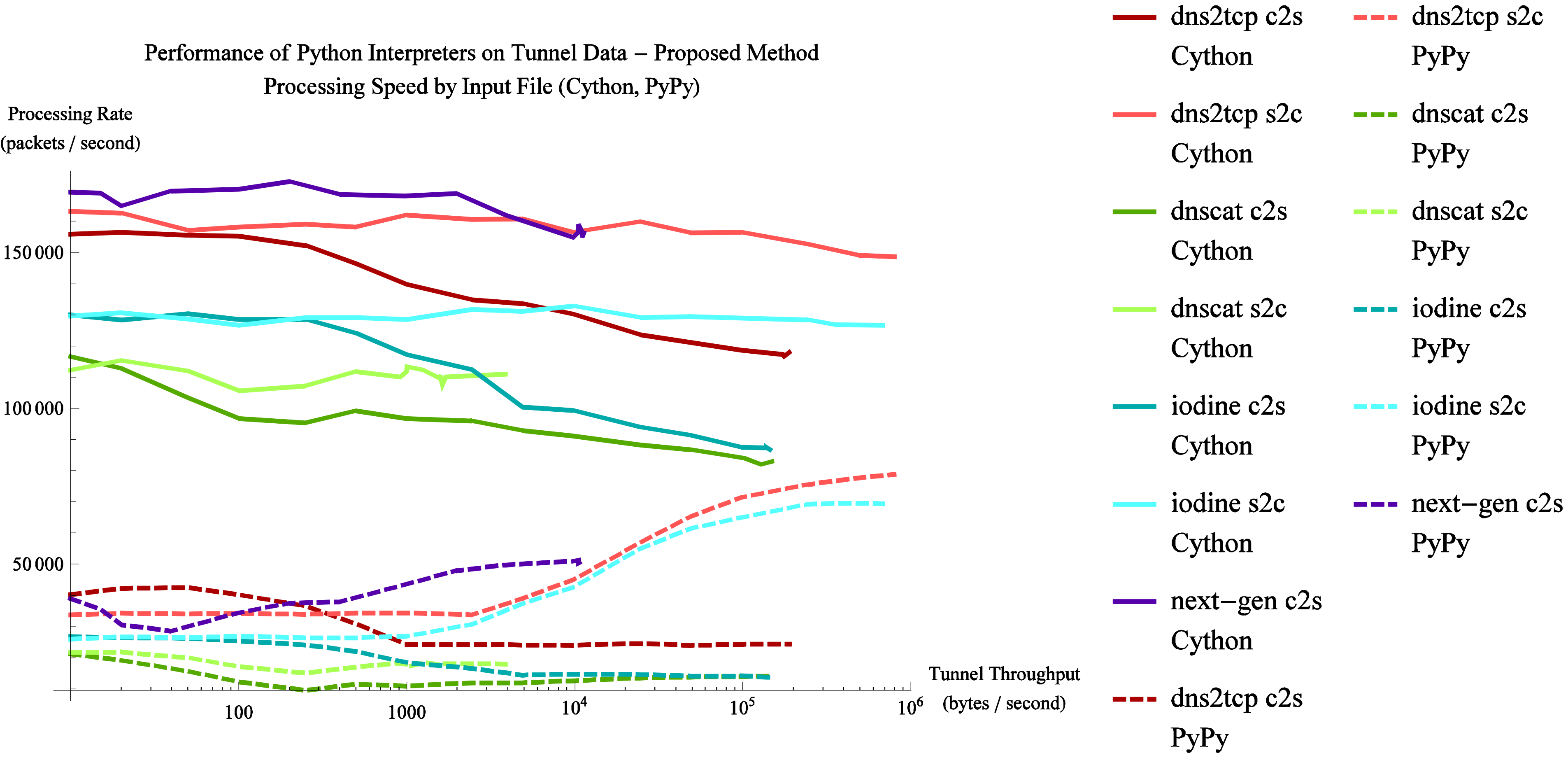
Processing Performance : Born Method



Processing Performance : Paxson Method



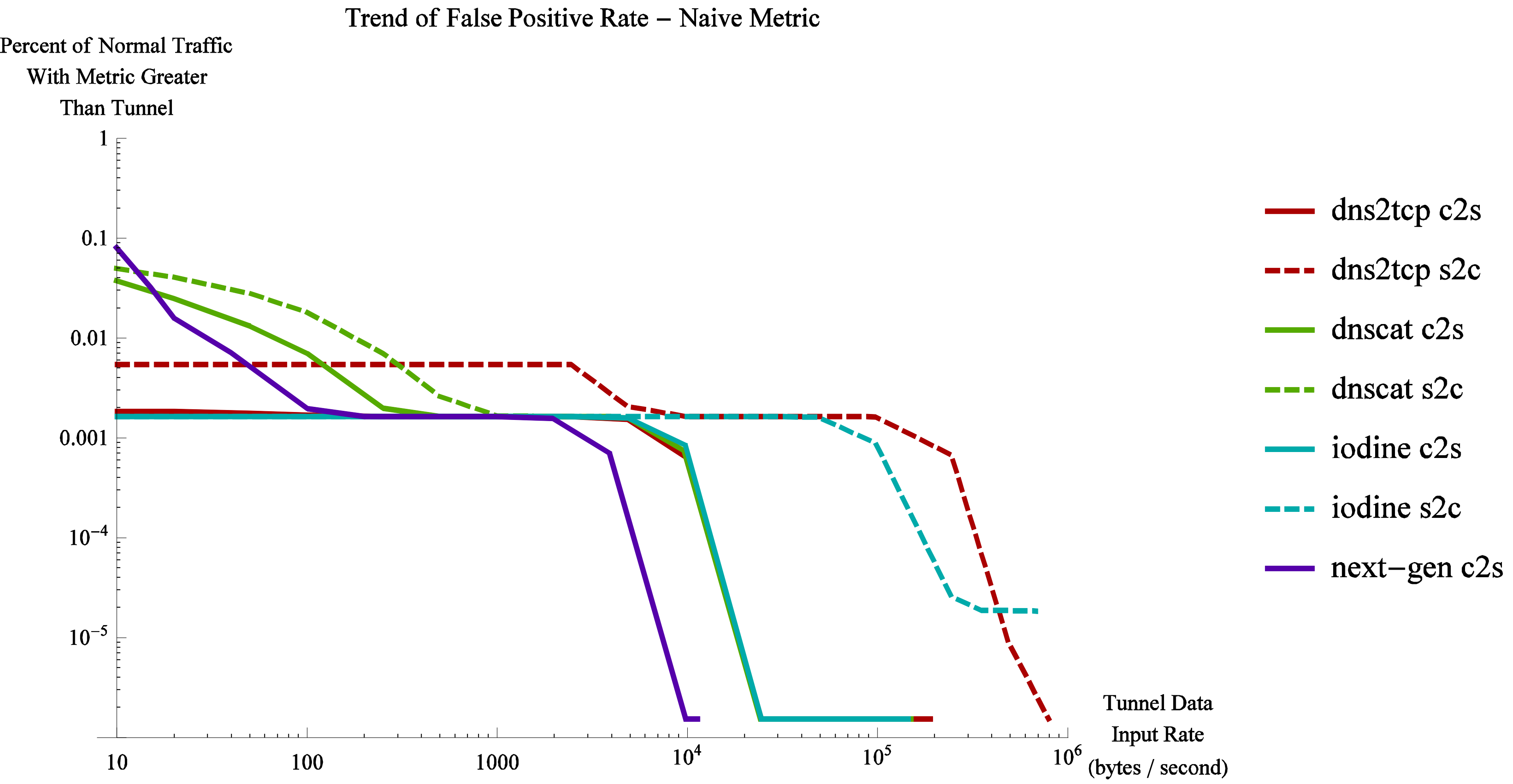
Processing Performance : Proposed Method



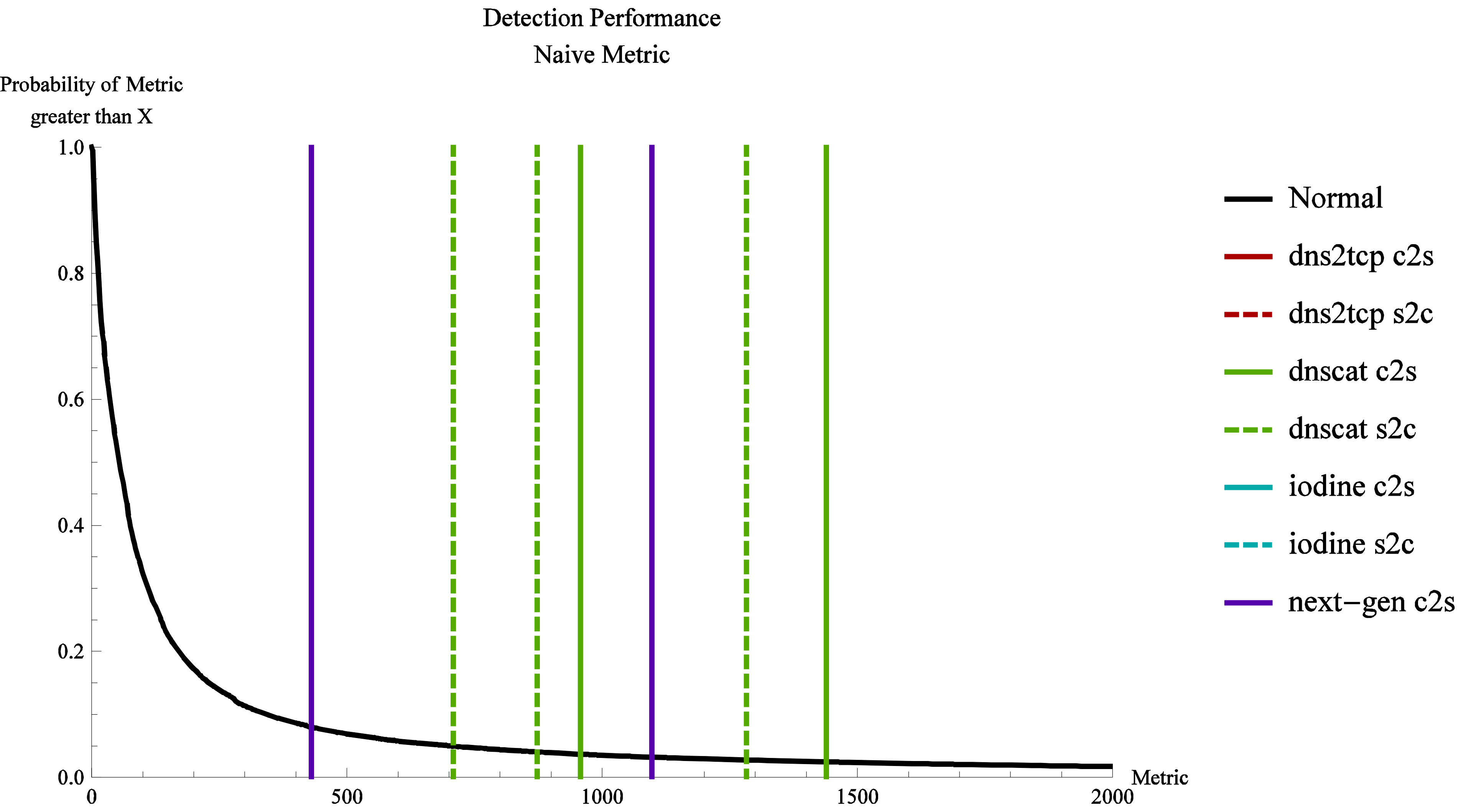
Processing Performance: Conclusions

- The naïve and proposed methods far outperform the other methods
- As throughput increases, both Paxson and Born approaches suffer severe degradation in performance.

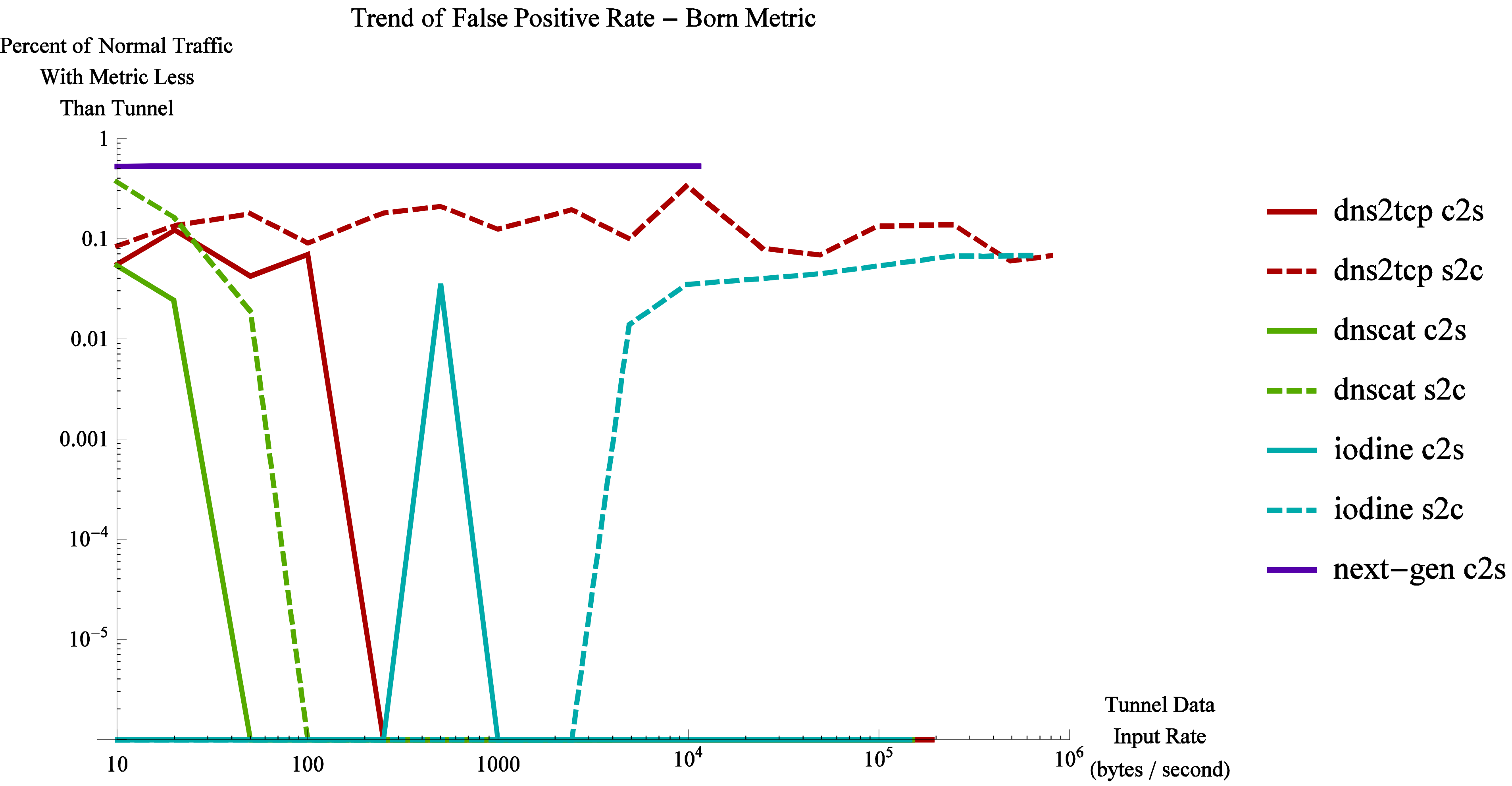
Detection Performance : Naïve Method



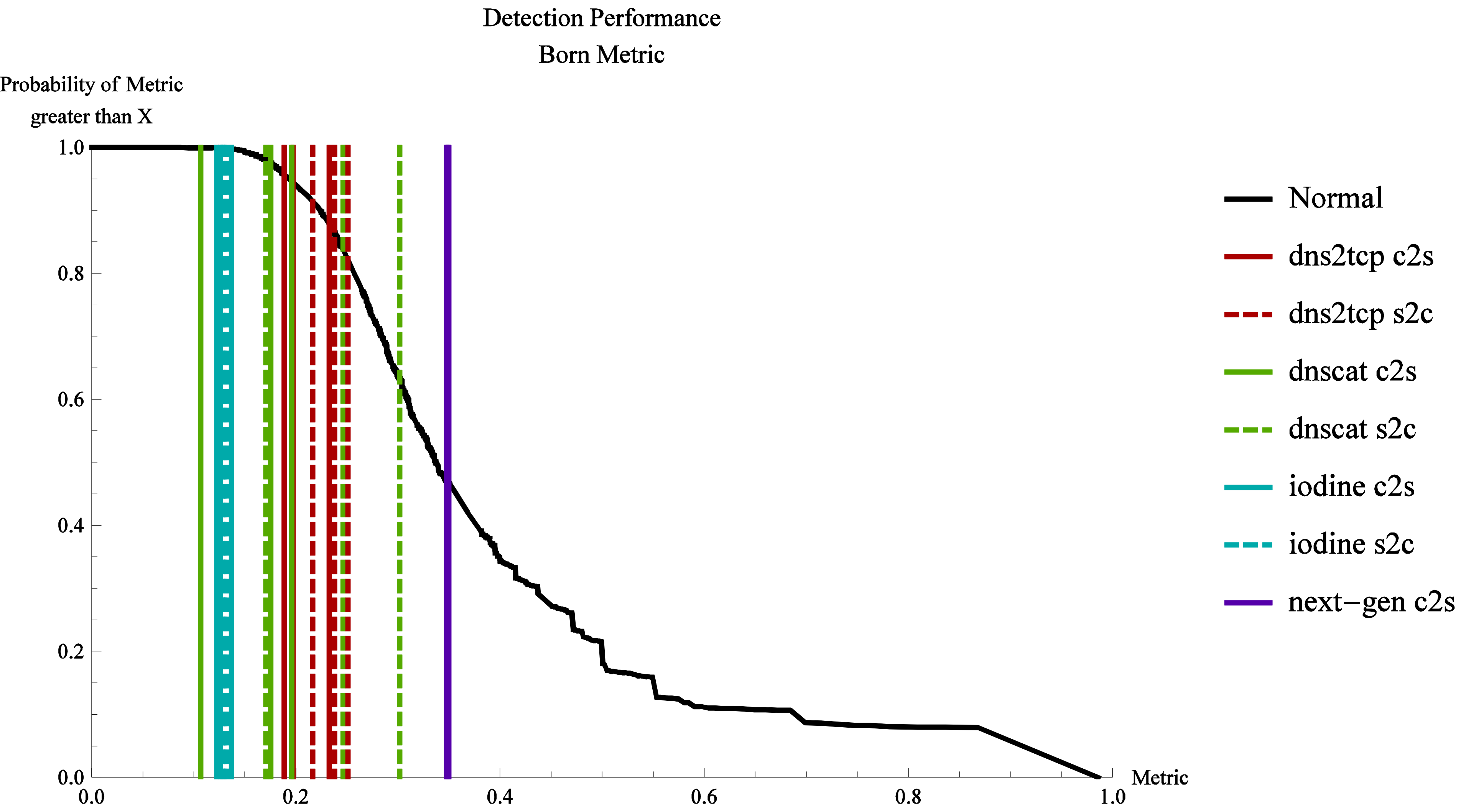
Detection Performance : Naïve Method



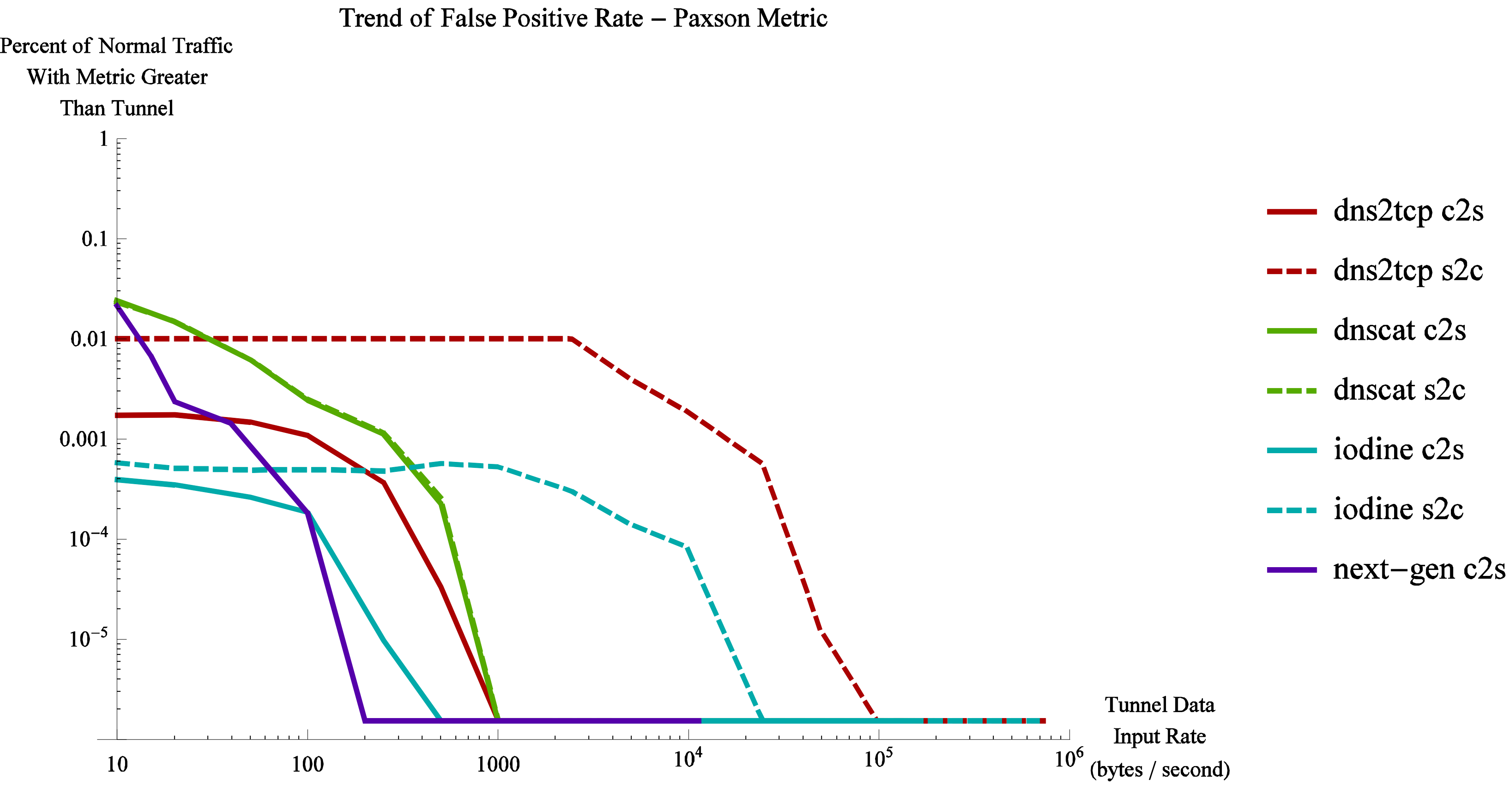
Detection Performance : Born Method



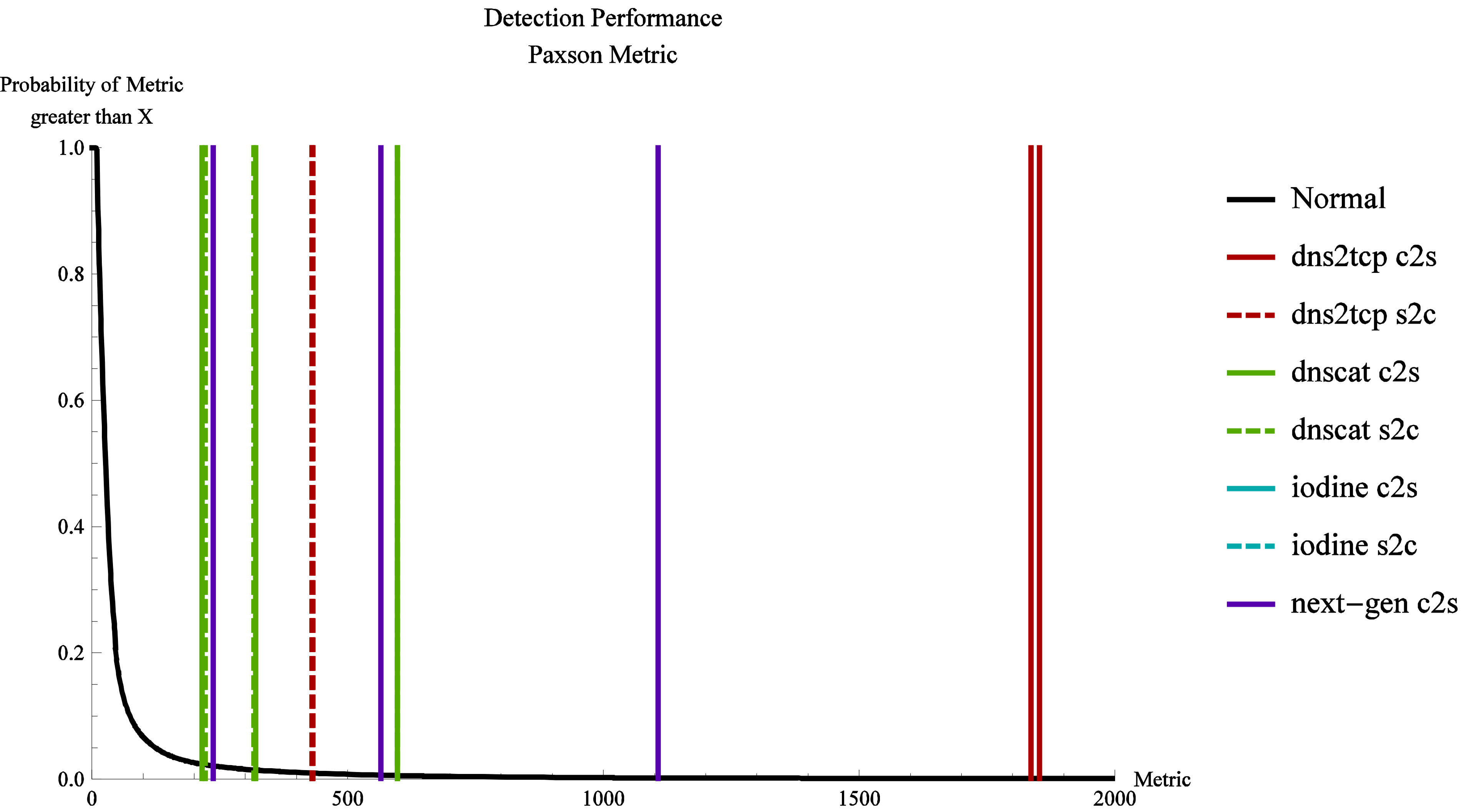
Detection Performance : Born Method



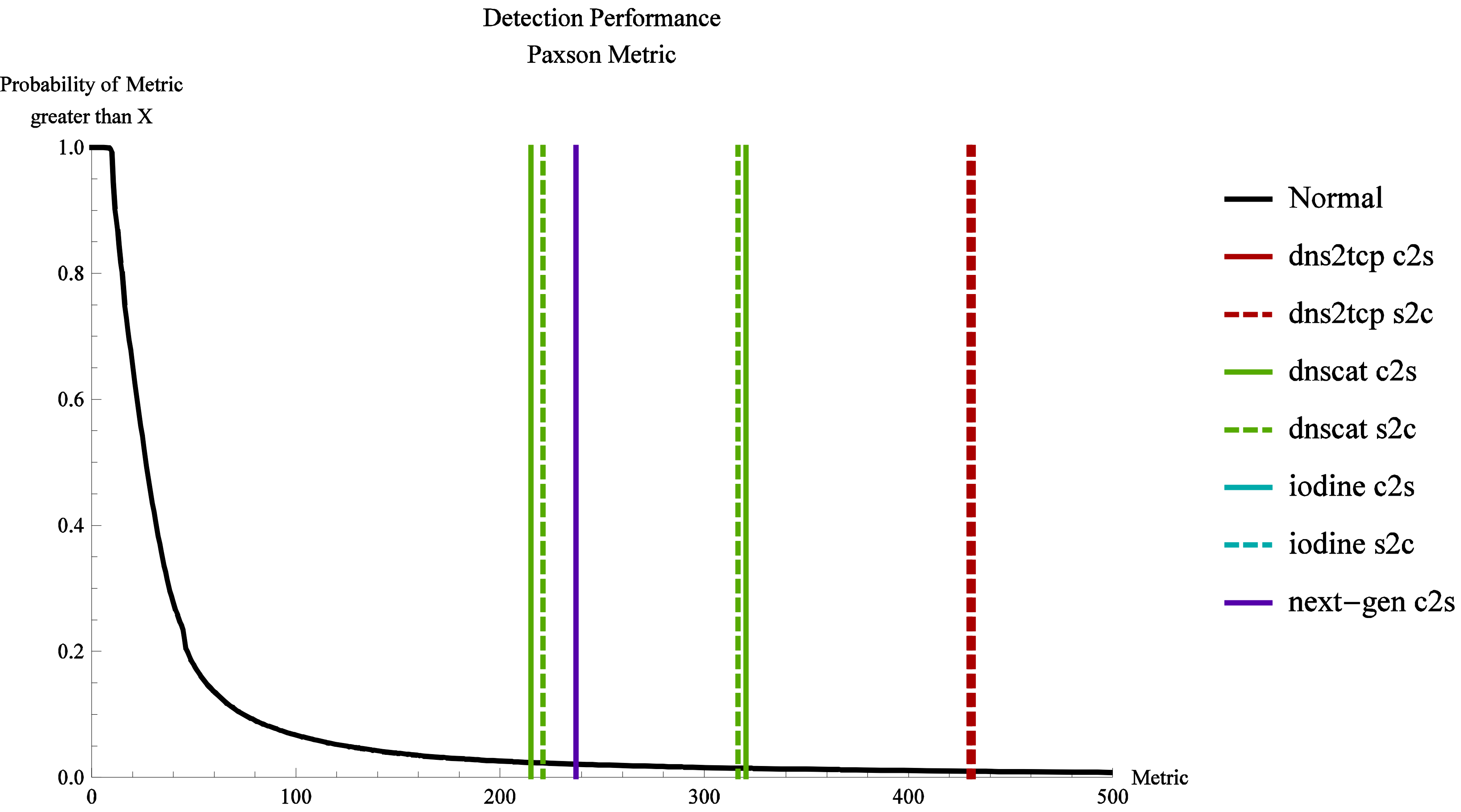
Detection Performance : Paxson Method



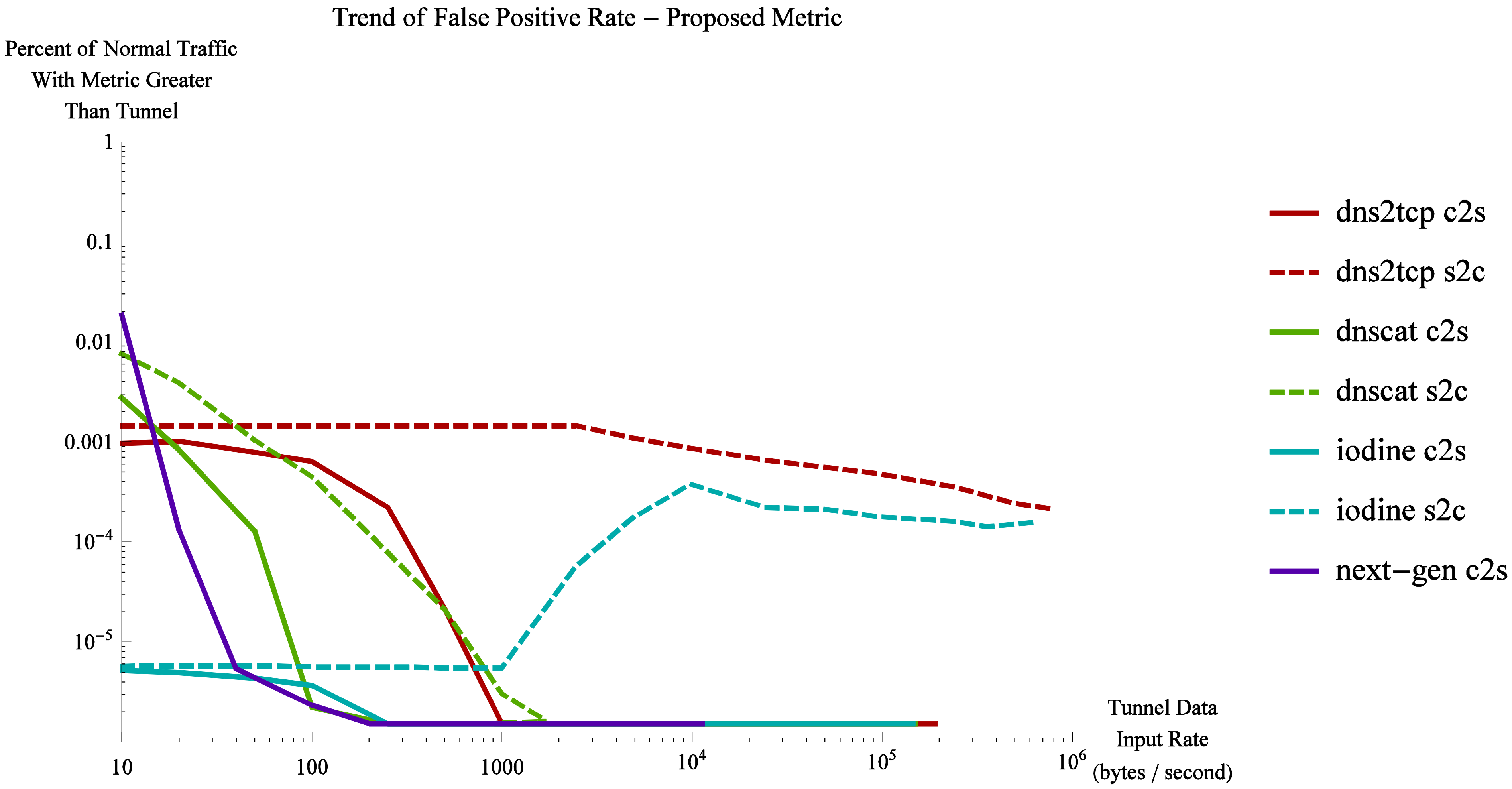
Detection Performance : Paxson Method



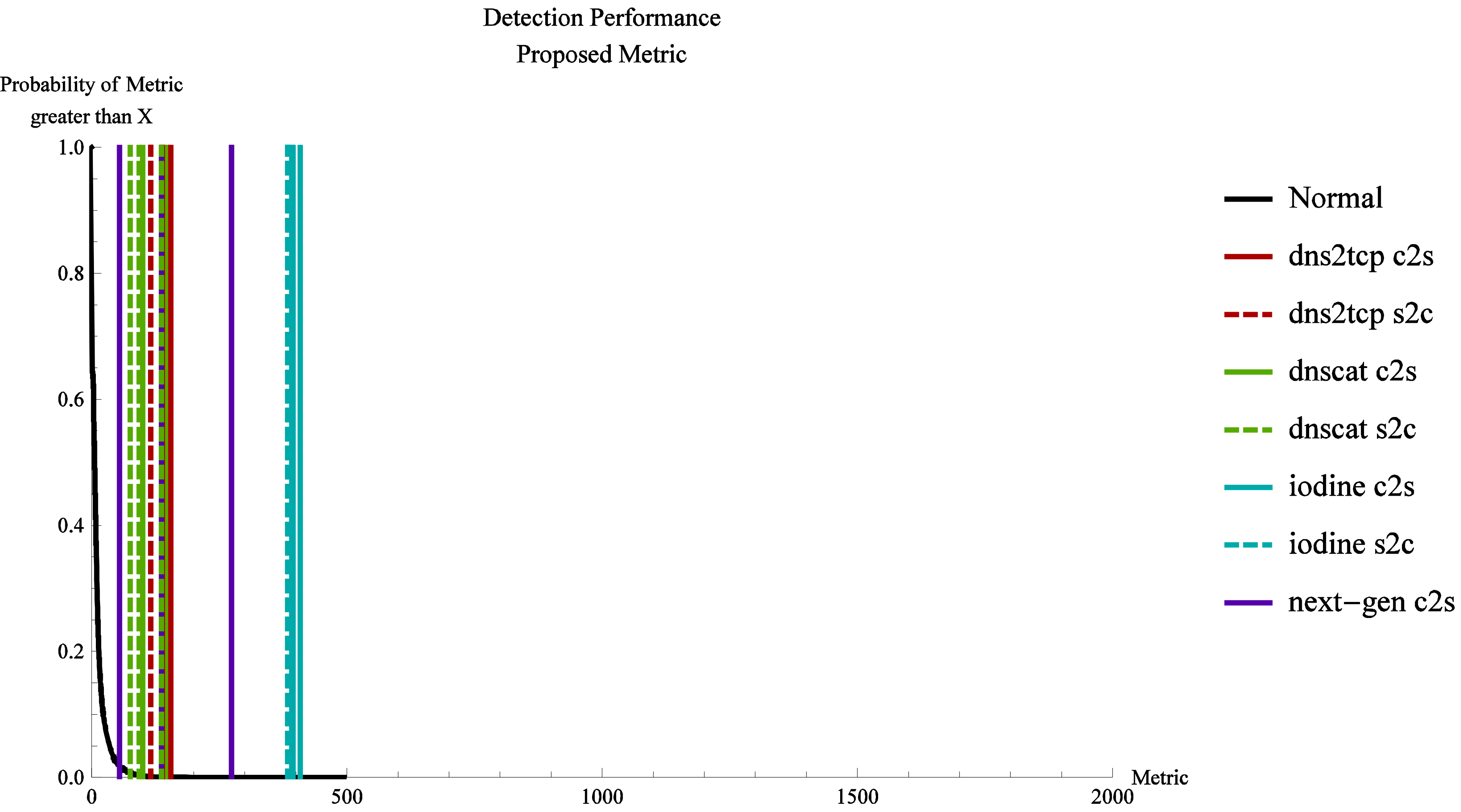
Detection Performance : Paxson Method



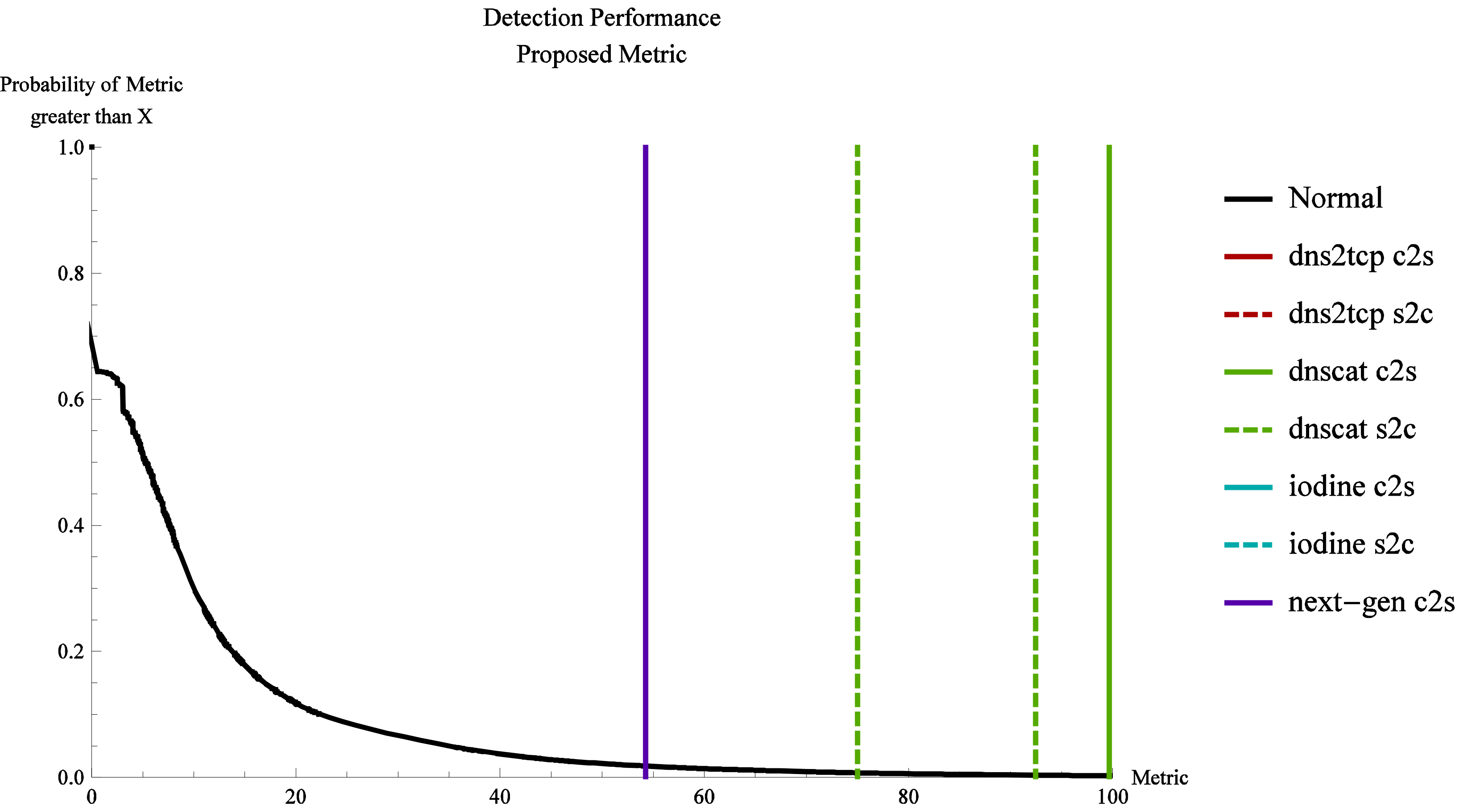
Detection Performance : Proposed Method



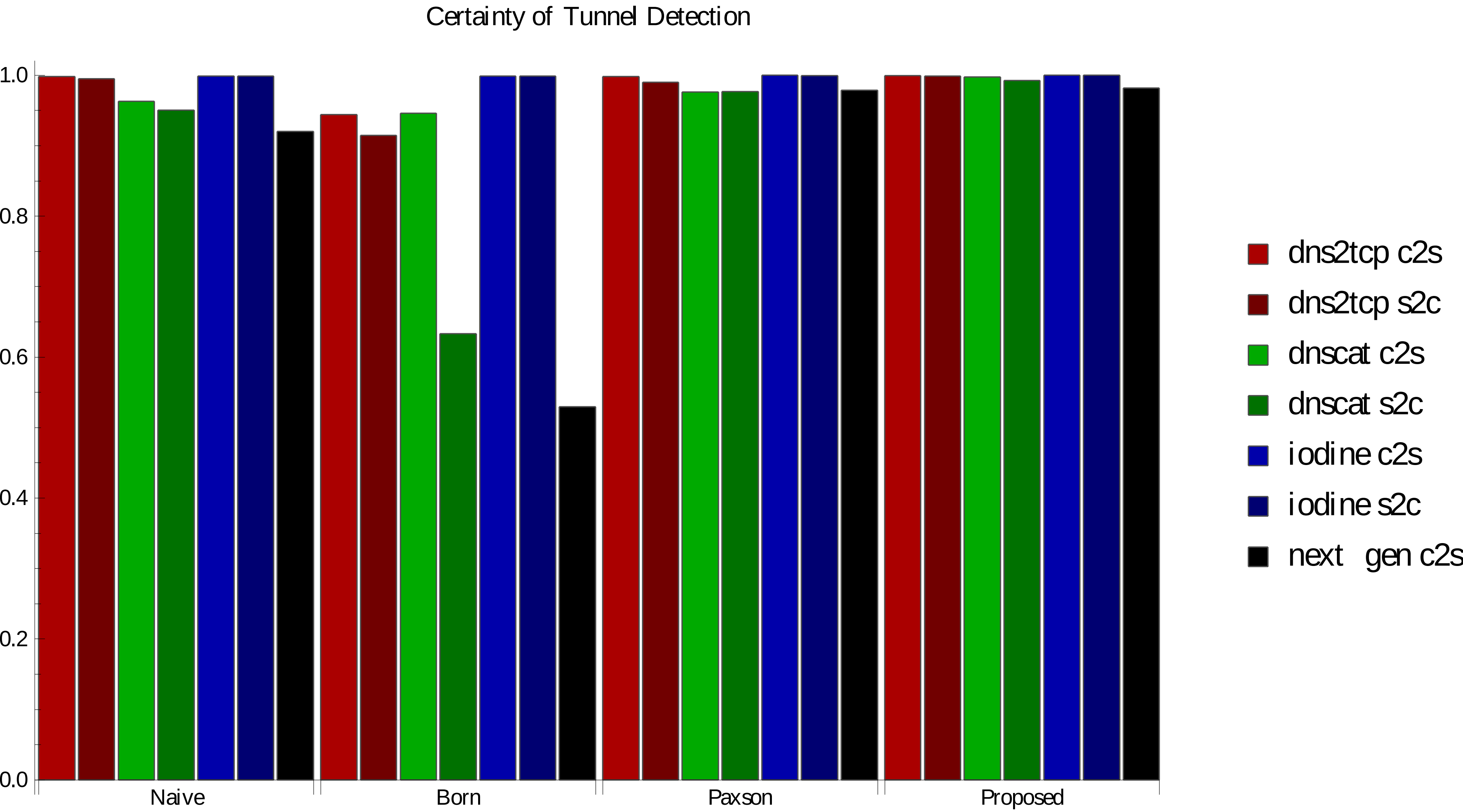
Detection Performance : ProposedMethod



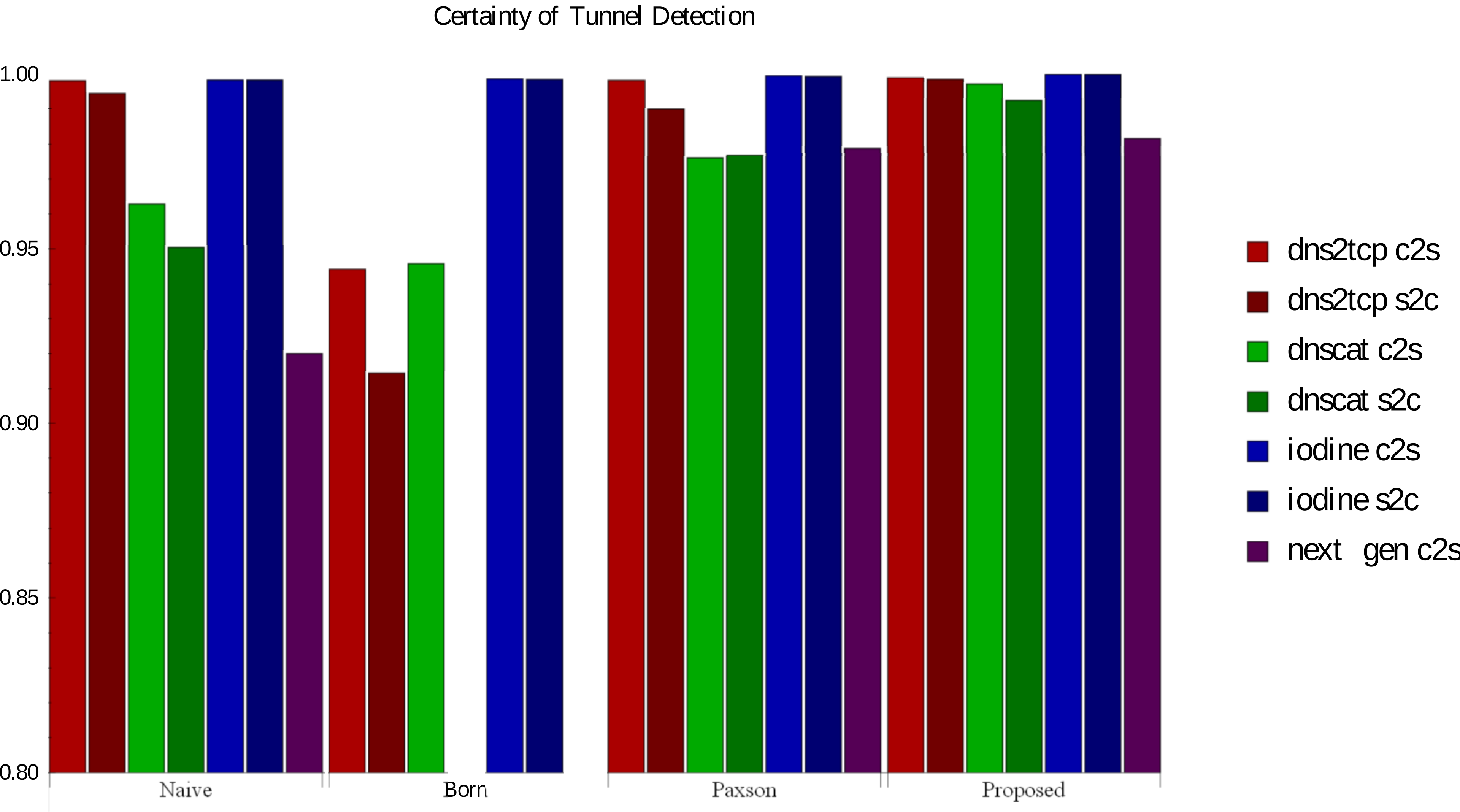
Detection Performance : Proposed Method



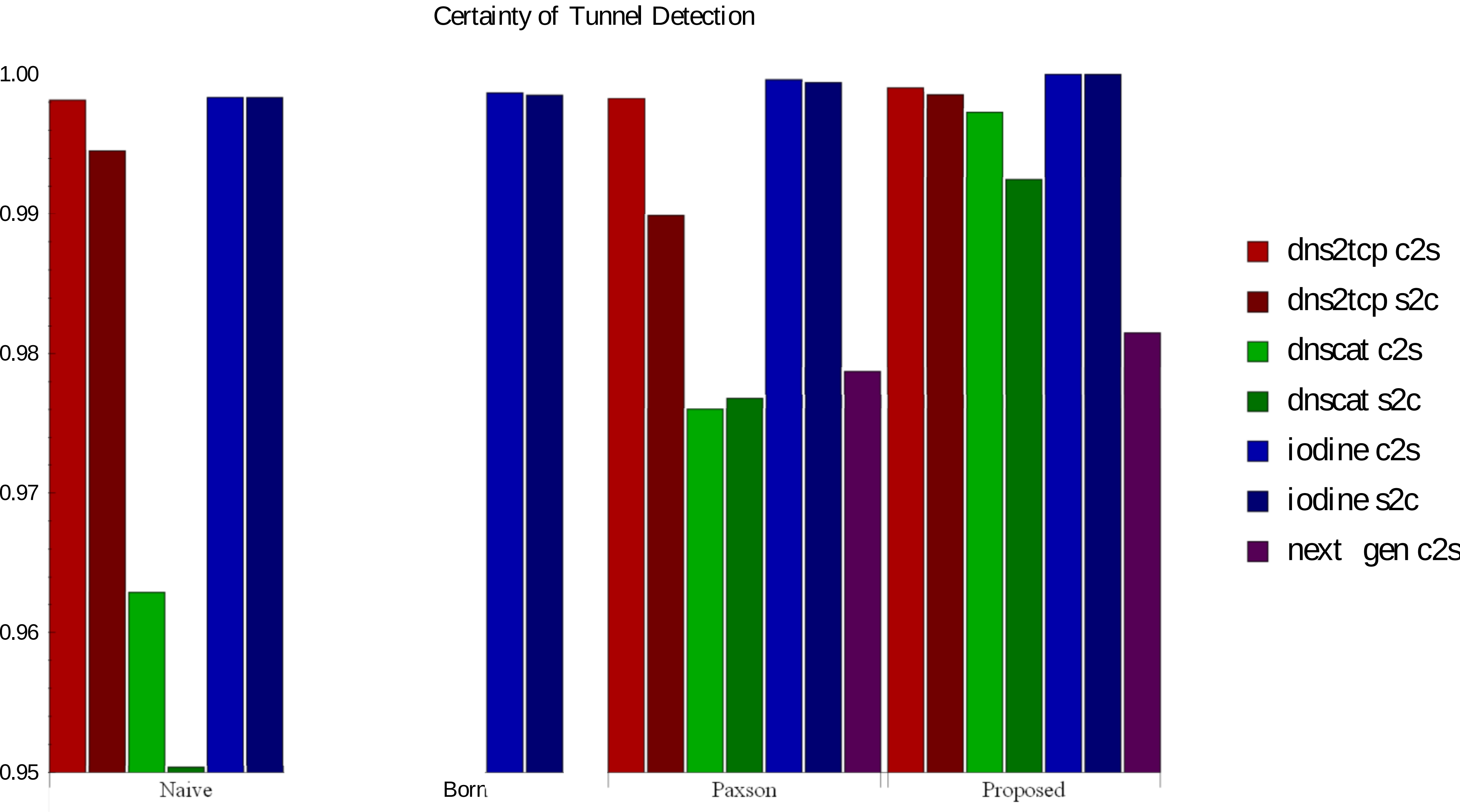
Detection Performance: Comparison of False Positive Rates



Detection Performance: Comparison of False Positive Rates



Detection Performance: Comparison of False Positive Rates



Detection Performance: Conclusions

- The proposed approach achieves categorically lower false positive rates than all other approaches.

- The prototype next-gen tunnel is the most difficult tunnel to detect by far.
  - Born’s approach has a false-positive rate little better than random chance.

Conclusions

- Proposed method:
  - Achieves the best detection performance, and nearly the best processing performance.
  - Represents a notable and novel contribution to the field.
  - *Is already implemented in high-performance C/C++, making deployment possible.*

Potential Future Work

- Test on more strictly curated data sets to remove confounding factors.
- Identify ways of improving false positive rate
  - Potentially with a more tailored metric
  - Potentially with more temporal knowledge and correlation

Thank you

- Questions