# NETPEEK: Early Speed Test Termination and Download Speed Prediction

https://github.com/yuheho7749/netPeek

Carson Kopper  Joseph Ng  Krishna Dhulipala

## Abstract

Speed tests are crucial for gathering information on the quality of service of computer networks, especially when networks do not or cannot employ passive measurement. These active measurements can serve as ground-truth information on which to challenge provider-reported coverage data by estimating last-mile connectivity. However, speed tests, like NDT, run for a fixed amount of time, and clients saturate the bottleneck link as much as they can, leading to congestion. Generally, the estimated throughput tends to stabilize early in the test, allowing for early termination. In this paper, we present NETPEEK, a machine learning solution to early termination system for NDT tests. We use periodic statistics from the speed test to predict the final throughput, then evaluate the model's confidence in the prediction to determine whether to terminate.

## 1 Introduction

High-quality networks are critical for end users to enjoy services on the Internet, such as streaming, video conferencing, and web browsing. These networks must be measured and characterized to ensure that providers guarantee their subscribers a good experience. For instance, the US relies on coverage maps generated with radio propagation models to characterize the quality of mobile broadband coverage. However, these maps are prone to overstate coverage [4,5]. One way to challenge coverage overstatement is to collect ground truth data via active measurement such as speed tests [2,7,9].

Traditional broadband speed test applications, such as Ookla, work by initially sending a request for data (for download tests) or sends some data directly (for upload tests) to a nearby test server, then gradually increasing the rate at which packets are sent and the size of the chunks until the limit of a link at a particular point in time is reached [9]. Typically, the link becomes saturated relatively quickly, so the recorded download and upload speeds only change significantly at the beginning of the test, then reach a steady state for the remainder. Most speed tests run for a preconfigured amount of time, so if the limit of an intermediate link is reached early in the test, the link will continue to be fully saturated for the remainder of the test. Thus, the speed test takes up link capacity that could be used for other forms of Internet traffic. This problem is exacerbated on networks with limited bandwidth, as a larger proportion of the network is devoted to the tests.

In this paper, we show that network conditions at the beginning of a speed test generally inform the final recorded download speed result. We present NETPEEK, a set of machine learning models allowing for a speed test to be terminated before the fixed end of the test, allowing for the resources of the bottleneck link to be freed early. We use two models: a "predictor", which learns how the network conditions throughout the test influence the resultant download speed, and a "terminator", which learns what types of network conditions contribute to high prediction accuracy.

Previous work has examined the use of machine learning for throughput prediction, but most research has focused on throughput prediction over longer timescales [8] or focused on domain-specific applications such as video streaming and adaptive bitrate algorithms [6,10]. These applications differ from speed tests, where recorded throughput tends to stabilize over the 10-20 second window of the test, and none focus on early termination.

The remainder of the paper is organized as to illuminate the purpose of NETPEEK, its design, and its internals/results. We first situate NETPEEK within the broader landscape of active measurement, discuss the open challenges that motivate early termination, and distill the guiding design objectives. Building on that foundation, we proceed to lay out the modeling pipeline and termination strategy, describe the PyTorch-based implementation and dataset curation, and quantify accuracy and bandwidth savings under diverse network conditions. This paper then reflects on key findings, limitations, and avenues for extending netPeek.

## 2 Background and Motivation

In recent years, internet performance monitoring has relied heavily on active measurements such as broadband speed-tests. Applications such as Ookla or Measurement-

Lab's NDT inject probing traffic, the aim of which is to saturate the path between a client and a nearby measurement server. This methodology yields ground-truth data that effectively serves to expose over-optimistic coverage claims on regulatory maps [4, 5].

This being said, this type of framework is inherently wasteful, since tests keep their links fully occupied once their bottleneck capacities have been determined. Especially on congested and/or low-capacity networks, this results in a sustained load, ultimately degrading the very same user experience the measurement ecosystem is meant to protect.

Considering the demerits and intrinsic flaws of tests like these, it is natural to ask whether a speed test be terminated as soon as its final download-rate result is predictable with high confidence, without sacrificing accuracy. A framework achieving these goals would be valuable not only in terms of reduced request latency, but also in terms of freeing scarce bandwidth for other traffic while still giving users trustworthy throughput readings.

### 2.1 Unaddressed Research Gap

Early-termination systems (which do not rely on a fixed-length test runtime window, but rather on termination upon completion) have received surprisingly little rigorous attention in this area. Prior work in this sector has employed either heuristic stopping rules (e.g. terminate when the instantaneous rate stabilizes) or re-purposing models developed for adjacent domains such as adaptive-bitrate (ABR) video streaming. These approaches ultimately fall short, however, because they:

(a) neglect sub-second TCP startup dynamics that dominate the first several round-trip times,
(b) provide no statistical confidence bound on the predicted final rate, and
(c) require client-side instrumentation incompatible with thin-client deployments exemplified by NDT.

Consequently, the measurement community still lacks a *server-only*, confidence-aware solution that decides both *(i)* what the eventual throughput will be and *(ii)* when that estimate is already accurate enough to abort the test safely.

### 2.2 Design Objectives

Motivated by these findings, NETPEEK aims to:

(I) **Preserve Accuracy**—match the full-length NDT result within a user-configurable error budget;
(II) **Minimise Overhead**—operate exclusively on server-side telemetry, introducing *zero* additional probe traffic or client modifications;
(III) **Quantify Confidence**—expose a calibrated probability that the predicted rate lies inside the error budget, enabling principled early termination; and
(IV) **Enable Drop-In Deployment**—integrate seamlessly with existing measurement infrastructure, leveraging the standard NDT trace format and logging pipeline.

### 2.3 Approach Preview

To realise these objectives, NETPEEK treats early-phase TCP time-series as a regression task whose output feeds a secondary classifier that decides when to halt the test. The modelling pipeline, implementation details, and evaluation are elaborated in Sections 3–5.

## 3 Design Approach

NETPEEK involves two components: a throughput estimator, which attempts to extrapolate from network features at the beginning of a test and predicts the final download speed, and a termination module, which uses network features to determine whether the test should terminate early or not. Both models utilize supervised learning with using time series information from TCP_INFO snapshots recorded during an NDT speed test. A key design consideration of NDT was to make the client as simple as possible and to implement all complexity on the server [1], so our design uses information available on the server at the time of the test. Therefore, a production version of NETPEEK would also be deployed on this server and simply notify the client if it reaches a termination decision.

To predict final speeds, we group the server TCP_INFO snapshots from the test into windows of equal duration and define $k$ to be the window number of this snapshot. For most of the minimum features provided in NDT's TCP_INFO logs, we take aggregates of these values over the window. In particular, we use the following features:

- How long the test has been running.
- How long during the test has been spent actively sending data from a nonempty TCP socket queue.
- The amount of time the test has stalled because there is not enough buffer space at the client.
- The amount of time the test has stalled because there is not enough buffer space at the server.
- The number of bytes that the server has received acknowledgement for. This is roughly the amount of data the client has successfully received during the download.
- The number of bytes that have been sent, both for the first time and retransmitted.
- The number of bytes that have been retransmitted.
- The average round-trip time during the window.
- The variance of the round-trip time during the window.

For each window, we take time series of the previous $n$ windows to include in our feature vector, capturing changes in each feature over time. In addition, we also take the minimum observed round-trip time seen by the kernel up to that point. To label each feature vector with ground truth information, we calculate the final download speed from the last snapshot of each speed test as follows:

$$FinalSpeed = BytesAcked/ElapsedTime * 8$$

where $ElapsedTime$ is measured in microseconds, thus obtaining the final speed in Mbps.

Once the predictor has been trained, we append the predicted download speeds to the feature vectors in our training dataset. This serves as the feature vector for the terminator, which we train completely separately from the predictor. We include the predicted speed directly so that this model can focus on learning the confidence of the predictions rather than spending training time re-learning the predictions. To be able to train an early termination model, we first required a set of ground-truth labels that would denote whether or not to terminate.

We generate this set of labels using the percent error calculated from the predicted download speed and the final download speed. Our assumption is that a prediction with a low percent error is a strong indicator of high confidence, and hence is a suitable candidate for early termination.

The confidence level, which we define as analogous to the early termination label, should have an output domain $[0,1]$ and be calculated with an inverse relationship to the percent error. Several schemes such as the reciprocal relationship ($y \propto 1/x$) and the linear inverse relationship ($y \propto 1-x$) were considered. Linear inverse was ultimately chosen because it more evenly encodes the output domain and allows intuitive manual tuning for the confidence cut-off for early termination. Alternatively, a pure binary classification model was also considered, but we found a confidence level to be highly beneficial to the clarity of the decision making.

In our testing, we arbitrarily interpret a confidence level greater than 0.5 to indicate termination, but this cut-off can be freely adjusted as it is not intrinsically part of the terminator model. Additionally, to account for cases with unusually high percent errors (over 100% or 1) in the linear inverse relationship, we clamped the percent error to a maximum of 1. The equation for calculating early termination ($StopPredict$), is as follows:

$$StopPredict = 1 - min(PercentError, 1)$$

where $PercentError \in [0, \infty)$ is in decimal form and $StopPredict \in [0,1]$.

## 4 Implementation

Our models were written in Python using the PyTorch library. For both the predictor and terminator, TCP_INFO snapshots were grouped into windows of 100 milliseconds, yielding 100 data points for a standard NDT download speed test. This allows us to make decisions about termination at a high granularity while reducing noise from small timing variations.

To incorporate time series information, we use features from the past $n = 5$ windows. Initially, we included the first $n$ windows as part of our training set, zeroing out values in the time series that would not yet have been recorded. After training, however, we observed that these samples were affecting the prediction capabilities of the entire model. Thus, we removed them and only considered snapshots from 500ms onwards. This choice prevents us from terminating in this early period, but there is a high degree of uncertainty during this time that makes prediction unhelpful.

Both models are feedforward neural networks with two hidden layers of dimension 64. The input features are normalized using the mean and standard deviation of each feature in the training data. We attempt to optimize the mean squared error defined as

$$MSE = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

where $y_i$ is the ground-truth label and $\hat{y}_i$ is the predicted value of sample $i$ based on the weights of the model.

## 5 Evaluation

In evaluating the performance of NETPEEK, we seek to understand how prediction and termination accuracy vary over different samples and through the duration of a speed test. We also desire an explanation for the model's decisions, ensuring it is a robust, generalizable solution to early termination. To evaluate performance, we leverage NetReplica, a framework capable of running various network processes under different conditions. Besides limiting static network parameters, such as link capacity and restricting latency, NetReplica's key advantage over other network simulators is the ability to replay cross-traffic, collected from real-world data, during the execution of the primary network process to simulate different levels of congestion.

The addition of these dynamic network conditions allow us to collect a representative sample of speed tests, improving the generalizability of a model trained using these experiments. As a proof of concept, we utilize TCP_INFO snapshot from 250 NetReplica experiments under different levels of congestion, with bottleneck link speeds ranging from 10 to 3000 Mbps and latencies from 10 milliseconds to one second. We used a 70-30 train-test split over all snapshots in the dataset for both models.

Figure 1 shows the percent error of each prediction on our test dataset against the cumulative proportion of samples that achieved this prediction error or lower. Ap-

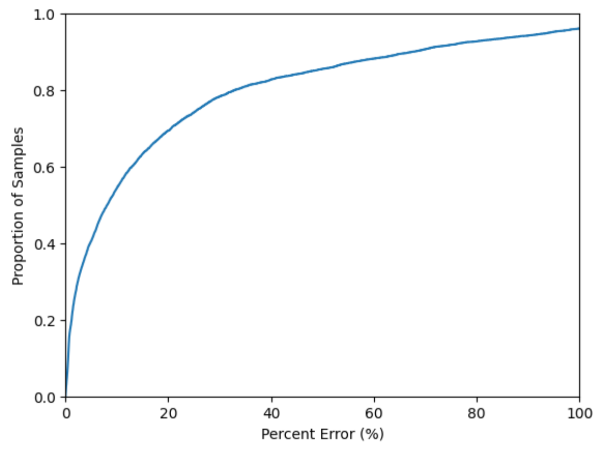

**Figure 1:** *Cumulative distribution function of percent error for each predictor snapshot.*

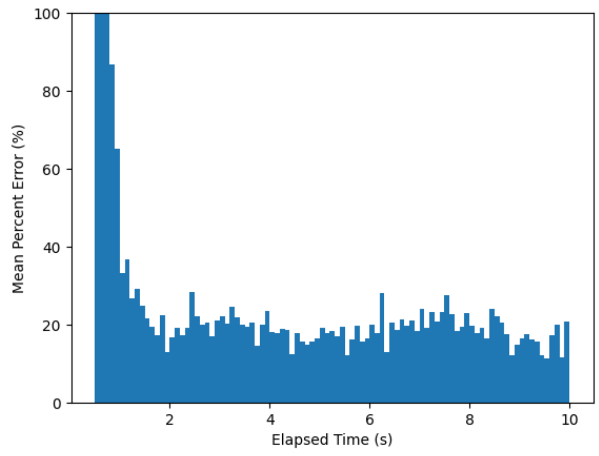

**Figure 2:** *Average percent error of snapshots, grouped by the duration of the speedtest.*

proximately half of all samples in the test set achieve a prediction error of 10% or lower. At the tail end, predictions are more inaccurate, reaching and exceeding 100% in extreme cases.

Figure 2 offers insights into the evolution of prediction accuracy throughout a speed test. Each TCP_INFO sample is grouped based on its window number. We take the average percent error of all samples that occurred in this window. As expected, the model performs worse earlier in the test when the state of the network can diverge; these errors around 100% are the same samples that are shown at the tail end of Figure 1. Past approximately two seconds, the average prediction error stabilizes at around 20%, showing the predictor's ability to estimate the final speed five times faster than the duration of a real speed test under different network conditions. We acknowledge that our training dataset was limited in scale and suspect that using more training samples covering a greater distribution of network conditions and resultant speeds has the potential to yield a higher prediction accuracy.

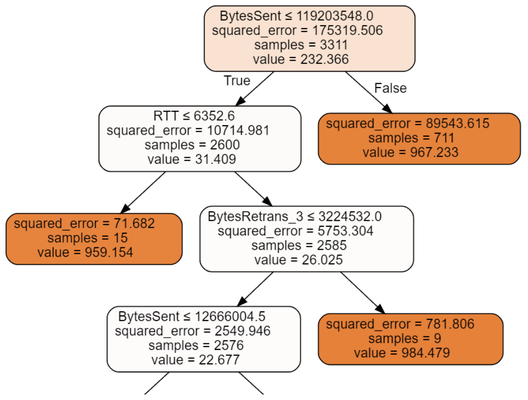

**Figure 3:** *Top nodes of the Trustee decision tree for the throughput predictor NN.*

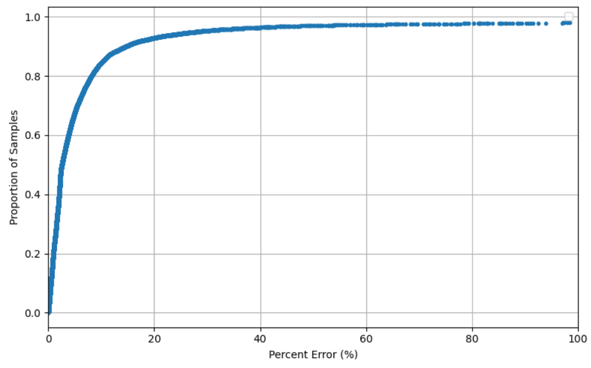

**Figure 4:** *Cumulative distribution of percent errors for terminator predictions.*

In an attempt to explain NETPEEK's decision-making, we apply Trustee to the predictor model, an AI explainability framework that produces a high-fidelity white-box decision tree of regression and classification models using imitation learning on the target model and training dataset [3]. While the extracted decision tree is large, even with Trustee's top-$k$ pruning capability, we present the highest-level decisions in Figure 3. In general, the throughput predictor appears to rely most heavily on the total number of bytes sent, the number of bytes retransmitted, and round trip time to make predictions. Notably, however, the elapsed time of the test does not appear as high in the decision tree, suggesting that the model does not perform drastically better as the test nears its completion.

To evaluate the accuracy of the terminator module and provide insights into cut-off confidence levels for early termination, we analyzed the error rate of the predicted confidence levels. Figure 4 illustrates the cumulative distribution of prediction errors, indicating that approximately 80% of the samples exhibit a percent error of 10%

or less. The distribution rises steeply, with over 90% of the samples falling within a 20% error margin. Based on this analysis, a confidence level cut-off point for early termination should account for 10-20% prediction uncertainty. In order to prevent premature model termination due to margin of error, we recommend to skew the termination cut-off to be higher than 0.5, although a more detailed analysis should be further investigated.

## 6 Conclusion

Using snapshots of TCP statistics during NDT tests, NETPEEK is able to predict the final download speed accurately and determine whether to terminate early in many cases. This mechanism, if implemented in an NDT server, allows tests to avoid contributing to congestion in large-scale broadband measurement campaigns and reduces the space overhead for providers, such as M-Lab, that store speed test data publicly for research and policymaking purposes. Admittedly, the size of our training dataset may have hindered higher-accuracy results; while only a proof-of-concept, we expect this type of system to become more applicable when trained on a larger dataset, capturing more nuanced depictions of network conditions. Additionally, we did not analyze what types of conditions contributed to high-confidence termination in NETPEEK. Such an analysis would be a valuable future extension of this work.

## References

[1] BASSO, S. ndt7 protocol specification. https://github.com/m-lab/ndt-server/blob/main/spec/ndt7-protocol.md. (Cited on page 2.)

[2] CLOUDFLARE. About speed. https://speed.cloudflare.com/about. (Cited on page 1.)

[3] JACOBS, A. S., BELTIUKOV, R., WILLINGER, W., FERREIRA, R. A., GUPTA, A., AND GRANVILLE, L. Z. Ai/ml for network security: The emperor has no clothes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2022), CCS '22, Association for Computing Machinery, p. 1537–1551. (Cited on page 4.)

[4] MACCARTNEY, G. R., AND RAPPAPORT, T. S. Rural Macrocell Path Loss Models for Millimeter Wave Wireless Communications. *IEEE Journal on Selected Areas in Communications 35*, 7 (2017), 1663–1677. (Cited on pages 1 and 2.)

[5] MANGLA, T., SHOWALTER, E., ADARSH, V., JONES, K., VIGIL-HAYES, M., BELDING, E., AND ZEGURA, E. A Tale of Three Datasets: Towards Characterizing Mobile Broadband Access in the United States. *Communications of the ACM 65*, 3 (March 2022). (Cited on pages 1 and 2.)

[6] MAO, H., NETRAVALI, R., AND ALIZADEH, M. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2017), SIGCOMM '17, Association for Computing Machinery, p. 197–210. (Cited on page 1.)

[7] MEASUREMENT LAB. What is measurement lab? https://www.measurementlab.net/about. (Cited on page 1.)

[8] NA, H., SHIN, Y., LEE, D., AND LEE, J. LSTM-based throughput prediction for LTE networks. *ICT Express 9*, 2 (2023), 247–252. (Cited on page 1.)

[9] OOKLA. Ookla speedtest. https://www.speedtest.net/. (Cited on page 1.)

[10] YAN, F. Y., AYERS, H., ZHU, C., FOULADI, S., HONG, J., ZHANG, K., LEVIS, P., AND WINSTEIN, K. Learning in situ: a randomized experiment in video streaming. *arXiv [cs.NI]* (2019). (Cited on page 1.)