Proof of Backhaul: Trustfree Measurement of Broadband Bandwidth

Abstract

Recent years have seen the emergence of decentralized wireless networks consisting of nodes hosted by many individuals and small enterprises, reawakening the decades-old dream of open networking. These networks have been deployed in an organic, distributed manner and are driven by new economic models resting on *tokenized* incentives. A critical requirement for the incentives to scale is the ability to prove network performance in a decentralized "trustfree" manner, i.e., a Byzantine fault tolerant network telemetry system.

In this paper, we present a Proof of Backhaul (PoB) protocol which measures the bandwidth of the (broadband) backhaul link of a wireless access point, termed prover, in a decentralized and trustfree manner. In particular, our proposed protocol is the first one to satisfy the following two properties: (1) *Trustfree.* Bandwidth measurement is secure against Byzantine attacks by collaborations of challenge servers and the prover. (2) *Open.* The barrier-to-entry for being a challenge server is low; there is no requirement of having a low latency and high throughput path to the measured link. At a high-level, our protocol aggregates the challenge traffic from multiple challenge servers and uses cryptographic primitives to ensure that a subset of challengers or, even challengers and provers, cannot maliciously modify results in their favor. A formal security model allows us to establish guarantees of accurate bandwidth measurement as a function of the maximum fraction of malicious actors.

We implement our protocol with challengers spread across geographical locations. Our evaluation shows that our PoB protocol can verify backhaul bandwidth of up to 1000 Mbps with less than 8% error using measurements lasting only 100 ms. The measurement accuracy is not affected in the presence of corrupted challengers. Importantly, the basic verification protocol lends itself to a minor modification that can measure available bandwidth even in the presence of cross-traffic.

Finally, the security guarantees of our PoB protocol output are naturally composable with "commitments" on blockchain ledgers, which are commonly used for decentralized networks.

1 Introduction

Decentralized networks have been in the making for decades. Starting with Software Defined Networking [30, 31] to simplify the hardware and open software [40] to facilitate application development, finally real-world deployments of decentralized Internet Service Providers (ISPs) [3] and decentralized Mobile Network Operators (MNOs) [20] have emerged. These decentralized networks have been made possible by the convergence of several engineering, business, and policy developments: the availability of cheap hardware for WiFi access points, and now even cellular base stations; the availability of cloud-native orchestration and AAA software [34]; and the availability of lightly licensed spectrum for cellular communication [18]. However, the real breakthrough in deployment comes with the emergence of a token-driven incentive ecosystem to bootstrap network growth and make individual hosts provide good network service. The leading exponent of such growth is the Helium network [20], which is a multi-RAT (radio access technologies) network supported by hundreds of thousands of "hotspots" hosted by individuals.

But a new engineering challenge has emerged – we need to design *secure and decentralized* network telemetry. In centrally managed networks, network telemetry is used for performance measurement and subsequent optimization. In contrast, network telemetry plays a more pivotal role in decentralized networks. It is now needed to ensure that the network nodes provide the service that they are being paid for. For this purpose, there are two new requirements for decentralized network telemetry:

- **Trustfree.** The protocol is secure against Byzantine attacks by the parties involved.
- **Open.** The barrier-to-entry for servers participating in decentralized telemetry is low. In particular, any node with a "reasonably good" internet connection should be able to participate.

The measurements that we get as the output of such trustfree and open network telemetry protocols can be viewed as a cryptographically secure proof of appropriate network performance.

In this paper, we focus on measuring a specific network performance parameter which is of central importance in decentralized wireless network deployments. In such deployments, users are required to get a broadband connection with appropriate bandwidth, as a backhaul for the wireless access point. But how do we know that the user has indeed set up a good backhaul connection? Can we simply use any of the existing techniques from the large body of literature, spanning over decades, on bottleneck link-throughput measurement? It turns out none of the existing tools is applicable for our setting; below we point out shortcomings of prominent techniques and clarify our contributions.

Comparison with speedtest. Speedtest (speedtest.net) is a state-of-the-art bandwidth testing tool widely used globally. Whenever a user (the "prover") sends a measurement request, a nearby server is selected from a centralized challenger server pool. The selected server generates traffic continuously until the target link is saturated. This requires the challenger server to have a high bandwidth, low latency, low packet loss link to the prover; this represents a high barrier to becoming a challenger. Furthermore, the measurements rely on the rates of sending packets from challengers and

CCS '23, Anonymous Submission, 2023
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484566

acknowledgements from the prover – an untrustworthy prover or challenger can adversely impact the measurement. Speedtest and similar architectures are unsuited for trustfree network telemetry.
Traffic aggregation. One way to allow more challengers to participate in the telemetry (and thus being more open) is to aggregate traffic from multiple challengers. Such aggregation removes the requirement of high capacity for a single server to measure high-bandwidth links, by uniting a group of servers to generate sufficient traffic in *parallel*. While this technique can improve the accuracy (e.g., recent works [1, 9, 51, 52]), the method is not trustfree: a Byzantine prover can readily manipulate the measurement results with no check or balance.
Interactive Measurement. To eliminate the need of trust on the prover, challengers should interact with other parties in the network to generate measurements. Popular interactive telemetry tools such as traceroute [23] and pathchar [24] use the timing information obtained by combining the Internet control message protocol's (ICMP) time-to-live (TTL) and packet dropped messages to estimate link performance over the Internet. In particular, challengers estimate the round-trip time (RTT) to the two end-points of the link to be measured, the throughput is derived by dividing the packet size by the difference of RTT. To further increase accuracy, packets of different sizes can be transmitted and the measurements can be aggregated through linear regression [17, 24]. Secure measurement, resistant to collusion between the prover and challengers, is not guaranteed in these protocols.
Our contributions. We present the first multichallenger PoB protocol for measuring backhaul bandwidth that satisfies the aforementioned trustfree and open properties, c.f., §3. Broadly, the protocol is built by implementing the following ideas:

(1) *Traffic aggregation.* We simultaneously send challenge traffic from multiple challengers to the prover. The duration of the challenge is chosen to be sufficiently high to ensure that traffic from all the challengers queues at the prover's backhaul link.
(2) *Unforgeable probe.* The challengers are selected randomly from a larger pool and each sends digital signatures as traffic, so that no other party can forge the measurement probe. Furthermore, to limit the influence of any one challenger, we limit the amount of challenge traffic that can come from a single challenger.
(3) *Short witness.* The prover can send a short message to the challengers to prove that it has received appropriate amount of data. As will be seen below, our security considerations require us to use a partially verifiable hash. For this purpose, we use a Merkle tree [38].
(4) *Robust timing measurement.* We estimate the RTT for the overall challenge by taking the median of the RTT measured by different challengers.

We implement these steps and experimentally validate the design choices to identify the best performing configuration; see Figure 1 for a depiction. The main contribution of this work is the trustfree property of the proposed protocol – it is secure under a rigorous threat model that we outline in §4. Our proposed protocol is the first one that can measure bandwidth of hundreds of Mbps without requiring any specialized server with high throughput and low latency for challengers in the trustfree setting. We further extend this protocol to measure available bandwidth in the presence of cross-traffic, making it a truly distributed "speedtest."

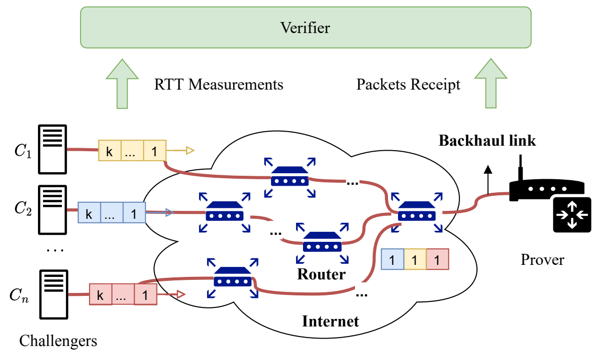


Figure 1: The multichallenger PoB Protocol.

We analyze the security of our multichallenger PoB protocol under a formal threat model which allows any subset of parties (up to 1/3 challengers collaborating with the prover) to maliciously deviate from the protocol. Since our probe is unforgeable, a corrupted prover still must get probe packets from the challengers. However, corrupted challengers, too, can modify the packet flow using two attacks: (i) the *withholding* attack where a corrupted challenger does not send probe packets; and (ii) the *rushing* attack where a corrupted challenger coordinates with the corrupted prover to send the packets or their information quickly without using the challenged link. To compensate for the withholding attack, we must send more packets than the link bandwidth to have sufficient traffic even after withholding attack. To compensate for the rushing attack, we multiply the actual measured bandwidth with a correction factor derived from the bound of bandwidth inflation to arrive at the *guaranteed bandwidth*. In addition, corrupted challengers can modify their outputs needed for verification. Specifically, they may report wrong RTT or they may claim modified packet data. We circumvent the former attack by taking a median of the measurements. To circumvent the latter attack, we use a Merkle tree which allows us to verify the consistency of the hash response from the prover with the data of uncorrupted challengers, without requiring correct data from the corrupted ones.

Overall, denoting the maximum fraction of corrupted challengers by β, we show that for $\beta < 1/3$[1], our protocol does not allow any prover to inflate the bandwidth and allows an honest prover to establish at least a fraction $(1-2\beta)/(1-\beta)$ of the true bandwidth.
Design variations using different primitives. Our main protocol assumes availability of digital signatures and nothing more. Different variants of this protocol are possible when different resources are available; we explore these variants in Appendix A. Specifically, we observe that (1) the protocol can achieve a higher accuracy of $(1-\beta)$ using TCP packets if a fairness property holds, as this prevents corrupted challengers from reporting deviating bandwidths higher than the majority; (2) the accuracy is reduced

[1]The adversarial threshold can be 1/2 if the verifier has access to a timer. See the discussion in §4.2.

Technique	Secure	Challenger BW < Backhaul BW	Accuracy
Pathchar [17, 24, 35]	✗	✓	Low
Packet dispersion based [12, 13, 33, 45]	✗	✗	–
Secure BW estimation [28, 48, 54]	✓	✗	–
Multichallenger PoB	✓	✓	High

(a)

Backhaul BW (Mbps)	Challenger BW (Mbps)	Challenge Data (MB)	Attack	Measured BW (Error %)	Guaranteed BW (Mbps)
250	25	3.44	–	246 (1.6%)	184
500	20	6.86	–	475 (5%)	356
750	75	10.31	–	705 (6%)	529
1000	100	13.75	–	921 (8%)	691
250	32	3.44	Rushing	331 (0.6%)	249
250	32	3.44	Withholding	241 (3.6%)	181

(b)

Figure 2: (a) Comparison of our multichallenger PoB protocol with prior-art techniques. (b) Summary of our performance results with 10 challengers. We perform attacks with 2 corrupted challengers.

to $(1-3\beta)/(1-\beta)$ when digital signatures are not used, due to the deniability of the packets delivery; (3) by limiting the access of corrupted challengers to additional links with higher bandwidth, the feasibility of rushing attacks is eliminated. Additionally, by implementing a shuffled packet creation phase, the effects of other attacks can be minimized and optimal accuracy can be achieved.

Implementation and evaluation. To convert the idealized protocol into a practical tool, we implement a variant of our protocol designed to address real-world issues (§5) and thoroughly evaluate its performance (§6). For instance, measuring links with 100 Mbps and higher bandwidth (commonplace in broadband services) requires latency measurements with an accuracy that is hard to achieve due to jitter in the Internet; we elaborate on overcoming this challenge in §6.1.

In our evaluation, we focus on the loss of measurement accuracy when using multiple challengers and traffic aggregation; in particular, we consider the loss of accuracy due to: (i) time synchronization errors and network jitter; (ii) computation time delays due to the use of digital signatures, hash computation, verification, and Merkle trees; and (iii) geographically spread challengers with heterogeneous capabilities. We also implement rushing and withholding attacks to illustrate that the security guarantees of the theory hold in practice. Our main experimental results are summarized in Figure 2. We report both the actual measured bandwidth and the output of our protocol – the guaranteed bandwidth – which is obtained by applying the correction factor $(1-2\beta)/(1-\beta)$.

2 Background and Related Work

Bandwidth estimation. The term bandwidth in the context of data networks quantifies the amount of data a network path can transfer per unit of time. Two metrics related to bandwidth are extensively investigated in the literature, the maximum possible data rate called *capacity* and the maximum available data rate called *available bandwidth* [45]. Packet dispersion techniques [12, 13, 29, 33, 47] are widely used to measure the capacity of the bottleneck link in a network path. Some of the techniques to measure available bandwidth are outlined in [1, 4, 11, 22, 25, 26, 36, 37, 46, 49, 51, 52]. Of these, tools such as Pathload [25, 26] and Pathchirp [46] create a short traffic load with different stream rates and observe the differences of one-way delay to adjust estimations. The state-of-the-art commercial tool Speedtest[4] employs a pool of servers with high bandwidth around the world to generate TCP traffic enough to saturate available bandwidth of the target link for a fixed duration. To improve accuracy, Speedtest and recent work (e.g., FastBTS[52]) leverage concurrent connections to generate TCP traffic in parallel. Swiftest[51] explores UDP to address limitations incurred by TCP-based methods such as slow start. Since bandwidth measurements play a critical role in optimizing centralized system performance and incentivizing decentralized services, other works shed light on the security of bandwidth measurements such as addressing inflation attacks in packet dispersion [28, 54]. Secure bandwidth estimation tolerating malicious parties in peer-to-peer networks has been discussed in [48], where every participant in the network evaluates the bandwidth of others and the results from all parties are combined into one consensus vector using principal component analysis. This scheme only obtains opportunistic observations during normal operations, and any node with high bandwidth cannot get fully appraised since all the other nodes are constrained by their own bandwidth. In another direction, [19] proposes a proof system for network telemetry for remunerating the relays in Tor network in proportion to the amount of data they transmit. The PoB proposed in this paper is aimed at measuring the backhaul bandwidth of end nodes in the Internet (e.g. WiFi access points and base stations). Further, we place no requirement on the bandwidth of the nodes measuring the backhaul; it can be much less than the backhaul bandwidth.

Per-hop capacity estimation. Of all the bandwidth estimation techniques in literature, [17, 21, 23, 24, 32, 35, 42] are closest to our work. These techniques can measure capacity for any link in an end-to-end path and so can be used to measure the prover backhaul, which is our goal. Traceroute [23] and pathchar [17, 24] make use of time-to-live (TTL) information in ICMP packets to control the packet drop at different intermediate hops to measure capacity of any link. [21, 32, 42] improve the approach used by pathchar [17, 24] with variable packet sizes. However these techniques require precise timing measurements of the order of packet transmission times. For bandwidth in 100s of Mbps, the packet transmission times are of the order of tens of microseconds. Given the jitter in latency over the Internet, our experiments in §6.1 reveal that such precise timing measurements are difficult. Indeed, [21] reports errors over 20% for measuring bandwidths of 500Mbps or more.

Decentralized networks. A common feature in every decentralized network deployment proposal is a proof system that can be used to verify a particular network performance parameter. The participants are incentivized to help in this proof system and also stand to gain when they can establish their contribution to this parameter. Helium [20] intends to unlock the potential of blockchains to establish a decentralized data network based on a tokenized incentive mechanism called proof-of-coverage. Hotspots are compensated for providing reliable coverage, to prove which challenge requests are issued regularly to random hotspots, who in turn are required to send beacons to other hotspots in the vicinity. Althea [50] aims to operate as a distributed ISP providing last-mile connectivity by creating a competitive platform and involving individual service providers into the market. Nodes maintain a route meter and accuracy score to assess the quality of neighbors to reach destinations and filter out inaccurate connections. To jointly address contractual and routing difficulties in inter-domain routing, Route Bazaar [14] constructs a system to establish end-to-end connectivity agreements among mutually untrusted parties automatically. The performance of the path is guaranteed by periodically generated forwarding proofs recorded on blockchains, which contain information like encrypted path tags, traffic samples and timing and throughput measurements.

3 The Multichallenger PoB Protocol

In this section, we formulate the PoB problem (§3.1), introduce main techniques (§3.2) and describe our multichallenger PoB protocol in details (§3.3) .

3.1 Problem Statement

We consider a system consisting of a group of end nodes such as base stations, WiFi access points and remote servers over the Internet willing to assist with backhaul bandwidth measurement. All nodes are connected to the network core through one backhaul link, simply referred to as backhaul from hereon, with an internal state θ representing the bandwidth of the link. We model the network core as a single point since fiber cables usually provide extremely high bandwidth, e.g., 100 Gbps. A PoB protocol allows a trusted verifier to use a subset of available nodes for securely measuring the backhaul bandwidth for a specific node called a *prover*, denoted P. The verifier can not observe the internal state θ_{P} of the prover directly. Instead, it needs to interact with the system by issuing a challenge request to the rest of the parties. We assume that n participants serve as *challengers*, denoted as $\{\mathsf{C}_1, \cdots, \mathsf{C}_n\}$, among which up to $f = \beta n$ challengers are corrupted, where β represents the fraction of adversarial challengers.

These challengers are responsible for generating and sending probes to the prover and output the measurements to the verifier. The output of PoB protocol is an estimation of the backhaul bandwidth of the prover. It guarantees the following two security properties:

- **Approximate completeness:** When the prover is uncorrupted, if the protocol outputs θ'_{P}, the actual bandwidth of the prover θ_{P} satisfies $\theta'_{\mathsf{P}} \geq \alpha\theta_{\mathsf{P}}$ for a constant accuracy ratio $\alpha \in [0, 1]$.
- **Soundness:** The protocol will not output a bandwidth higher than θ_{P}, even when the prover is corrupted.

Other assumptions for theoretical analysis. Our protocol makes use of digital signatures and collision resistant cryptographic hash functions. These primitives are assumed to be perfectly secure. Parties verify every received signature by default and ignore those signed invalidly. We assume the network is synchronous and every challenger has access to a synchronized clock. Each node knows the public address and public key of others. We suppose there exists a trusted verifier such as a blockchain to broadcast information to the system. It is crucial to note that the assumptions made in our theoretical analysis are for the purpose of simplification and easy understanding. However, when evaluating our implementation, we take into account the possible deviations from these assumptions that may occur in real-world scenarios.

3.2 Protocol Overview

Heuristically, the protocol proceeds by randomly selecting a set of n challengers from all the participants to send a train of probes to the prover (Figure 1). The protocol enforces packets from different challengers to arrive at the link to be measured around the same time. This traffic aggregation strategy effectively combines the group of challengers to an equivalent challenger with larger bandwidth and thereby renders the prover's backhaul the bottleneck link.

Formally, suppose that the protocol starts at time t_0, and each challenger C_i starts to send a sequence of k packets of size b each at time t_{i1}, $1 \leq i \leq n$. We have the following two requirements:

(1) *Aggregation condition.* There is a $\theta_0 \leq \min(\theta_1, \cdots, \theta_n)$ such that the bandwidths θ_i of C_i satisfy
$$t_0 + \frac{b}{\theta_0} = t_{11} + \frac{b}{\theta_1} = \cdots = t_{n1} + \frac{b}{\theta_n}. \quad (1)$$
(2) *Bandwidth condition.* The quantity θ_0 satisfies
$$(n - f) \cdot \theta_0 \geq \theta_{\mathsf{P}}. \quad (2)$$

The “aggregation condition" coordinates the arrival time of packets sent from various challengers, allowing multiple traffic flows to be effectively aggregated and merged into one stream at an appropriate rate. In this way, at least $(n-f)b$ bits of data are transmitted within the transmission time of one packet for a single challenger (b/θ_0). Therefore, the equivalent bandwidth of the challenger group is enlarged by at least a factor of $(n - f)$. The “bandwidth condition" ensures that the prover's backhaul becomes the bottleneck link. We assume there always exist enough potential challengers to meet both conditions.

While honest participants are supposed to correctly report their own bandwidth and send packets on time, corrupted parties can violate the conditions in arbitrary ways. For instance, a corrupted challenger can *rush* the packets through extra links or refuse to send any packets. We therefore require the prover to send back a response to all challengers on receiving $(n - f)k$ packets as a transmission receipt, since we can not expect more packets in the case of a *withholding* attack (detailed in §4.1). Then challengers measure the time it takes to transmit all these packets. Since corrupted challengers can claim arbitrary values, the median of all reported time is used to avoid manipulations and provide robust timing measurement.

Cryptographic primitives. To save the bandwidth used for verification, the prover only sends back a short witness consisting of the hash of received packets to terminate the measurements. We define a hash function *Hash* that takes any string as input and outputs a deterministic fixed-length random string. When the input is a set of messages, we assume the set will be serialized to a string to compute the hash. For verification, we ask each challenger to verify only packets sent by itself and employ the Merkle tree construction to enable inclusion check with only partial data. A sequence of hashes can be aggregated using the function *MerkleRoot* to a single cumulative hash. This technique reduces the verification overhead per challenger to $O(\log n)$, which can greatly decrease the response traffic when n is large. In addition, our protocol uses digital signatures to generate unforgeable probes and ensure traceability of bad behavior, for which the following functions are provided: a key generation function *keyGen* which outputs a pair of secret and public keys, a signing function $sign(sk, msg)$ that allows anyone to sign an arbitrary message with a secret key sk, and a verification function $verify(pk, msg, \sigma)$ that checks whether the signature σ is derived by signing given message msg using the secret key paired with the public key pk.

Blockchain as a verifier. Our PoB protocol is triggered by a challenge request issued from a verifier, who is also responsible for the broadcast of public parameters such as protocol start time t_0 and bandwidth requirement θ_0. Generally, any trusted entity can play the role of a verifier. In tokenized decentralized settings, smart contracts supported by blockchains are a good fit to transparently generate, broadcast protocol parameters and coordinate measurement reports from multiple challengers. Implementing a version of our protocol with blockchain as a verifier (deploying appropriate smart contracts) is beyond the scope of this paper.

3.3 Full Protocol

The full protocol contains two phases, a *measurement phase* described in Algorithm 1, where challengers generate and send packets, and a *verification phase* described in Algorithm 2, where the prover constructs proofs for the verifier. Finally, the verifier outputs the final results after verification.

Measurement phase. At the beginning of the measurement phase, the verifier produces three public protocol parameters (t_0, m_0, θ_0) and broadcasts it to all challengers, where t_0 is the start time of the protocol, m_0 is a random message, θ_0 is the global minimum bandwidth. To participate in the measurement process, challengers must first measure their own bandwidth θ_i and generate a key pair consisting of a public key pk_i and a private key sk_i. The public key is then sent to the verifier. The time to start sending the first packet t_{i1} is determined by Eq. (1). The challenger C_i generates a sequence of k packets by signing the public message m_0 together with a sequence number q and sends them one by one to the prover with a fixed duration b/θ_0. The process is depicted in Figure 3, where the q-th packet of C_i is sent at time t_{iq} (see line 6 of Algorithm 1).

Algorithm 1 The Measurement Phase of PoB Protocol

```
1:  as a challenger $\mathsf{C}_i$
2:      $t_0, m_0, \theta_0 \leftarrow$ generated and broadcast by verifier
3:      measure its own bandwidth $\theta_i$ (require $\theta_i \geq \theta_0$).
4:      generate $(pk_i, sk_i) \leftarrow keyGen$, send $pk_i$ to verifier.
5:      for sequence number $q = 1, \cdots, k$ do
6:          $t_{iq} \leftarrow t_0 + q \cdot b/\theta_0 - b/\theta_i$
7:          $\sigma_{iq} \leftarrow sign(sk_i, (q, m_0)), m_{iq} \leftarrow (i, q, \sigma_{iq})$
8:          send packet $m_{iq}$ to $P$ at $t_{iq}$
9:      upon receiving $(h_{1i}, h_2, \sigma_i)$ from $P$ do
10:         if $verify(pk_P, (h_{1i}, h_2), \sigma_i)$ outputs 1 then
11:             record round trip time $\Delta_i \leftarrow curTime_i - t_i$
12:
13: as a prover
        $\forall i \in [1, n], \mathcal{M}[i] \leftarrow \emptyset, (pk_p, sk_p) \leftarrow keyGen$
14:     upon receiving packet $M'$ from $\mathsf{C}_i$ do
15:         $(i, q, \sigma) \leftarrow M'$
16:         add $(q, \sigma)$ to $\mathcal{M}[i]$
17:         if $\sum_{j=1}^{n} |\mathcal{M}[j]| = (n - f)k$ then
18:             $\forall j \in [1, n], h_{1j} \leftarrow Hash(\mathcal{M}[j])$
19:             $h_2 \leftarrow MerkleRoot(\{h_{1j}\}_{j \in [1,n]})$
20:             $\forall j \in [1, n], \sigma_j \leftarrow sign(sk_p, (h_{1j}, h_2))$
21:             $\forall j \in [1, n]$, send $(h_{1j}, h_2, \sigma_j)$ to $\mathsf{C}_j$.
22:
```

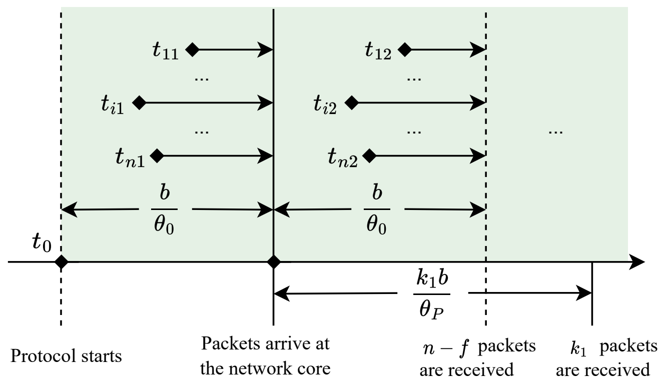


Figure 3: The measurement phase of PoB protocol. t_{iq} **is the time for challenger** C_i **to send the** q**-the packet.** k_1 **is the actual number of packets received with sequence number** 1**.**

On receiving the packets from the challengers, the prover separates the messages from different challengers and adds them to corresponding sets. When the total number of received packets reaches $(n - f)k$, the prover generates a response to broadcast to all challengers. This terminates the measurements phase. The response contains (1) a receipt h_{1i} to each challenger C_i, which is the hash of all packets received from the same sender; and (2) a Merkle root h_2 constructed from all receipts. All challengers record the round trip time Δ_i between the start time t_{i1} and the time $curTime_i$ at which a valid response is received.

Algorithm 2 The Verification Phase of PoB Protocol

```
1:  as a prover
2:      $B_i \leftarrow$ a bitmap of size $k$ where the $q$-th bit in the bitmap is set if some packet $(q, *) \in \mathcal{M}[i]$
3:      $P_i \leftarrow$ the Merkle proof of $h_{1i}$
4:      send $(B_i, P_i)$ to $C_i$, output (REPORT, $h_2$) to verifier.
5:
6:  as a challenger
7:      upon receiving $(B_i, P_i)$ from $P$ do
8:          Add all $(q, \sigma_{iq})$ to a set $\mathcal{M}$ if the $q$-th bit is set to 1 in $B_i$. Check whether $Hash(\mathcal{M}) = h_{1i}$.
9:          Reconstruct the Merkle root $h'$ using $P_i$ and $h_{1i}$. Check whether $h_2 = h'$.
10:         If both two checks are passed, output (REPORT, $\mathsf{P}, h_2, \Delta_i, |\mathcal{M}|$) to verifier.
11:
12: as a verifier
13:     $\mathcal{M} \leftarrow \emptyset, cnt \leftarrow 0$
14:     upon receiving (REPORT, $h$) from prover $\mathsf{P}$ do
15:         record $h$
16:     upon receiving (REPORT, $\mathsf{P}, h_2, \Delta_i, k_i$) from challenger $C_i$ do
17:         Check $h_2 = h$, add $\Delta_i$ to $\mathcal{M}$, $cnt \leftarrow cnt + k_i$
18:         if $cnt \geq (n - f)k$ and $|\mathcal{M}| \geq n - f$ then
19:             $\Delta' \leftarrow Median(\mathcal{M})$
20:             $\theta'_{\mathsf{P}} \leftarrow \frac{cnt \cdot b \cdot (n-2f)}{\Delta' \cdot (n-f)}$
21:             Output (PoB, $\mathsf{P}, \theta'_{\mathsf{P}}$)
22:
```

Verification phase. In the verification phase, the prover is responsible for proving to the challengers the content of received packets. To that end, it constructs another response revealing the indices B_i of packets received from C_i and showing the inclusion of each receipt in the Merkle tree with a Merkle proof P_i. It also sends the Merkle root to the verifier. On receiving the Merkle proof P_i from the prover, the challengers reconstruct the receipt hash and the Merkle root. The challenger C_i forwards Δ_i and the number of packets sent by it to the verifier, after making sure that both the hashes are consistent. At the end of the second phase, the verifier aggregates these measurements from all the challengers about how long the measurement phase takes and how many packets are indeed received by the prover. It also forwards the reports from challengers to the prover, who checks the consistency and submits the packets and the Merkle proof in case disputes exist. Once the verifier has received "sufficiently many" valid reports . Specifically it waits to receive a confirmation from at least $n - f$ challengers with at least $(n - f)k$ packets in total. We assume there is no packet drop and address the packet drop problem in §5. Then it calculates the final output bandwidth by dividing the total size of received packets by the median of the reported RTTs; see line 20 of Algorithm 2.

4 Security Model and Analysis

The primary challenge in trustfree networking is the inherent security vulnerability, since any party can depart from the protocol at will and even collude with other parties to manipulate the results. In this section, we formalize a broad threat model underlying measuring bandwidth, systematically examine the security issues to which the system is exposed (§4.1), and analyze the security guarantees for our protocol (§4.2).

4.1 Threat Model

We consider a static adversary allowed to corrupt at most f among n challengers before the protocol starts, the rest of uncorrupted challengers are referred to as honest. The prover can also be corrupted. In addition to the backhaul link indicated in the model in §3.1, we allow the adversary to access external communication channels. Specifically, the adversary has access to additional links with arbitrarily high bandwidth connecting to all the participants. The corrupted parties can act arbitrarily in order to either inflate or deflate the measured bandwidth; we discuss prominent attacks below.

Withholding attack. The measurement of bandwidth requires the challengers to send probes and measure the time it takes for the prover to receive the probes. The corrupted challengers who have been bribed by the consumers or the competitors of a prover might be motivated to deflate the bandwidth estimation to reduce service costs. They can delay the sending of the packets to increase the observed RTT or even withhold the packets for the entire protocol. During the verification phase, corrupted challengers can also refuse to report verification results. Moreover, the prover can also bribe the challengers to withhold packets during the measurement phase but report that the maximum number of packets have been sent in the verification phase.

Rushing attack. Since a reasonable incentive system will allow the participants to be compensated in proportion to their bandwidth, provers can collude with challengers to inflate bandwidth to get more rewards. During the measurement phase, instead of the backhaul link which is filled with the packets from uncorrupted challengers, corrupted challengers transmit packets through an extra link with an extremely high bandwidth to finish the measurements within a shorter time.

Information sharing attack. Besides the rushing attack, another way for the prover to get more information about the data from the challengers than that was transmitted through the backhaul link is to exploit the information structure. In the verification phase, to facilitate the verification of whether the packets received by the prover are indeed those sent by the challengers, the challengers are required to provide the information related to packet generation. If the information to generate packets is much smaller than the actual packet data and is shared to the prover directly, the prover can also terminate the measurements much earlier since it can generate a fraction of packets by itself. For instance, in our protocol, corrupted challengers can send their secret keys to the prover.

DoS and related attacks. Attacks are possible where a challenger sends traffic with invalid signatures, sends duplicate packets, or reports fewer packets in the verification phase. These attacks result in the same outcome: the prover is unable to report its actual bandwidth because unwanted extra traffic was present at the time of challenge. One can view all these attacks as variants of Denial-of-Service (DoS) attack. DoS or distributed Dos (DDoS) attacks are possible in our setting, especially if one implements the protocol by setting public IP address. Specifically, a challenger or a group of them with a high bandwidth link can flood the prover backhaul with invalid packets, preventing the valid packets sent by uncorrupted challengers from reaching the prover. Even a challenger who has not been selected for a particular challenge but knows the time of the challenge can disrupt the challenge similarly. It will be challenging to alleviate the problem using standard filtering techniques [43] without cooperation from the ISP. In fact, if ISP cooperates or the deployment is on a local network, a promising direction for countermeasure will be to adopt a network architecture with accountability[10] and incentivize ISPs to enforce challenge-specific security policies[41]. The detection and dis-incentivization of such attacks (for instance, using crypto-economic "slashing") is a topic of future work. Furthermore, we assume that there is no attack during the network synchronization phase; in future we can adopt Byzantine resistant time synchronization schemes such as those in [16] to further enhance security.

4.2 Security Properties

Theorem 1 (Soundness). *When $f < n/3$, the prover cannot inflate the bandwidth.*

Proof. According to the protocol, all packets with sequence number q sent by uncorrupted challengers will arrive at the network core at $t_0 + q \cdot b/\theta_0$ and be added to the backhaul link queue Q. Because it takes at least b/θ_0 to finish transmitting all packets with the same sequence number (according to bandwidth condition in Eq.2), the queue will never be empty during the measurement phase. Before sending the response, the prover waits for $K \geq (n-f)\cdot k$ packets, among which at most fk packets come from corrupted challengers. These packets can be sent through an external link (rushing attack) or generated by prover directly if the secret keys are shared in collusion (information sharing attack). In either case they will not actually consume the bandwidth of the prover's backhaul link. Even so, there are still at least $K - fk \geq (n-2f)\cdot k$ packets sent by uncorrupted challengers. Since packets from uncorrupted challengers are not forgeable by anyone else, the earliest time at which the prover can send response is the time at which $(K - fk)$ packets from Q get delivered, which is at least $t_R = t_0 + b/\theta_0 + (K-fk)b/\theta_{\mathsf{P}}$, whereby the uncorrupted challengers will receive the response and time $t_R^i > t_R$.

Since the verifier needs to collect at least $n-f$ time measurements, of which $n-2f$ must be reported by uncorrupted challengers, the median Δ' of the RTTs must be bounded by the minimum of honest measurements since $f < n/3$, in this way the estimated time will not get affected by individual misreports. Then, denoting the set of honest challengers as H, we have $\Delta' \geq \min\{t_R^i\}_{i\in H} - t_0 - b/\theta_0 > (K-fk)b/\theta_{\mathsf{P}}$ and

$$\theta'_{\mathsf{P}} = \frac{K\cdot b\cdot(n-2f)}{\Delta'\cdot(n-f)} \leq \frac{K\cdot(n-2f)\cdot\theta_{\mathsf{P}}}{(K-fk)\cdot(n-f)} \leq \theta_{\mathsf{P}}.$$

□

Theorem 2 (Approximate completeness). *When $f < n/3$ and the prover is uncorrupted, the protocol will always output bandwidth with accuracy $\alpha = \theta'_{\mathsf{P}}/\theta_{\mathsf{P}} \geq (n-2f)/(n-f)$.*

Proof. When the prover is uncorrupted, it waits for $(n-f)\cdot k$ packets to generate the response. Even under withholding attacks, where corrupted challengers never send their packets, $(n-f)\cdot k$ packets generated by honest challengers will arrive at the prover before $t_R = t_0 + b/\theta_0 + (n-f)kb/\theta_{\mathsf{P}}$. Then all uncorrupted challengers receive the response at the same time and output to the verifier. Assuming that the size and the latency of the response is negligible, we have the median RTT $\Delta' = (n-f)kb/\theta_{\mathsf{P}}$. If corrupted challengers try to misreport the number of packets received by the prover, claim the proof sent by the prover is incorrect, or even withhold the measurement results, the prover can send the genuine packets it has received from the challenger to the verifier together with the Merkle proof. The verifier will reconstruct the Merkle root from the submitted partial data and Merkle proof to solve disputes. Thus, even under attacks, the total number of packets are no less than $(n-f)k$. Consequently, the protocol will output

$$\theta'_{\mathsf{P}} = \frac{(n-f)kb\cdot(n-2f)}{\Delta'\cdot(n-f)} = \frac{n-2f}{n-f}\theta_{\mathsf{P}}.$$

□

Remark. (Adversarial threshold.) Our protocol can tolerate up to a fraction $1/3$ of Byzantine challengers. This threshold of $1/3$ arises from the requirement to ensure that a majority of $(n-f)$ RTT measurements are collected from uncorrupted challengers. This allows the verifier to terminate the collection responsively (or "lazily") when receiving enough reports without the requirement of a timer. However, if the verifier has access to a timer with desired accuracy (roughly 100ms for us), it can wait for a certain period (determined by maximal network delay and backhaul links transmission delays) to collect the measurements, by which all challengers who fail to send the report are recorded as 0. In this case at least $n-f$ reports of total n reports are from honest challengers, the majority requirement $n-f > f$ means that the protocol is able to tolerate a fraction $1/2$ of Byzantine challengers.

5 Protocol Implementation

In this section, we present the protocol implementation in a real system. Towards practicality, we discuss the factors that are not addressed in our theoretical modeling (§5.1) which leads to the modifications in implementation to the basic form of the protocol (§5.2).

5.1 Practical considerations

Challenger bandwidth. In §3.3, we assume each challenger can measure its spare bandwidth θ_i precisely. However, this bandwidth may be time-varying and it will be difficult for the challenger to measure every time. We relax this requirement by allowing every challenger simply ensure that it has at least θ_0 bandwidth available for the challenge. Here $\theta_0 = \theta_{\mathsf{P}}/(n-f)$ is the smallest value that satisfies the bandwidth condition in Eq. (2). Each challenger will now send the challenge traffic at rate θ_0.

Latency. The key requirement of our technique is that the packets from each challenger reach the prover backhaul at the same time. The aggregation condition Eq. (1) ensures this when there is no synchronization error or latency. However, in practice, a packet from the challenger C_i will take time l_i to reach the prover, where l_i is the one-way latency from challenger C_i to the prover. The value of l_i can indeed vary for different challengers and to account for such varying latencies, we modify Eq. (1) as

$$t_0 + \frac{b}{\theta_0} + l_0 = t_{11} + \frac{b}{\theta_0} + l_1 = \cdots = t_{n1} + \frac{b}{\theta_0} + l_n$$

where t_{i1} is the start time of challenger C_i to send the first packet. Note that θ_i is replaced by θ_0 as in our implementation; challengers release packets at rate θ_0.

Likewise, the response packet from the prover will take time l_i to reach challenger C_i. Accordingly, Δ_i in Algorithm 1 now changes to $\Delta_i = curTime_i - t_i - 2 \cdot l_i$, where $curTime_i$ is the time when challenger C_i receives the response from the prover. For measuring l_i, before the challenge starts, each challenger sends 20 ICMP ping packets to the prover and takes the average across these 20 packets as RTT. We set the value l_i as $RTT/2$. Note that using ping packets to measure l_i creates a vulnerability: prover can delay the ping response, inflating l_i and thereby θ'_{P}. To circumvent this, one possible solution is to restrict the challengers to be within certain geographical limit of the prover. This will ensure that the maximum l_i is bounded by a small value of say 10-15 ms and so the error in estimating θ'_{P} due to incorrect l_i is also small, provided the challenge duration is chosen to be sufficiently long; see error analysis of implemented protocol in § 5.3.

Packet drops. We have assumed that all the k packets from a challenger will reach the prover. However, since all the challengers send the packets simultaneously to the prover, there will be buffer overflow at the last link of the prover and some packets will be dropped. We use the UDP protocol for the challenge packets, so dropped packets will not be retransmitted. Since we use packet count as the termination condition, packet drops will prevent the challenge from being terminated. In our experiments, we find that we can compensate for the packet drops by asking challengers to send $1.1k$ packets, i.e., assuming a packet drop rate bounded by 10%, this guarantees that the prover receives $(n - f)k$ packets and terminates.

Time synchronization. We require that all the challengers are synchronized via Network Time Protocol (NTP) [8]. Note that NTP does not ensure perfect time synchronization, there can still be residual synchronization errors of the order of tens of milliseconds over the Internet [39].

Computation overhead. The use of cryptographic primitives like $Hash$ and $MerkleRoot$ (Algorithm 1) inevitably incurs computation overhead, which will delay the prover from sending responses to challengers and thereby add to an error in measurements. We detail empirical computation times of these primitives in §6.1 as a function of the number of challengers and challenge duration for completeness.

5.2 Implementation

We implement challengers and the prover as UDP socket applications in C++ and each challenger conducts measurements by sending UDP packets to the prover. Details are described below.

Digital signatures. As outlined in Algorithm 1, a challenger needs to sign each packet. We leverage the Edwards-curve digital signature scheme, Ed25519 [27] for signature generation and verification as its computation overhead is low. Our measurement results indicate that if we use challenge packets of 64 Bytes (the size of Ed25519 signature), measurement accuracy is affected especially if the challenger is connected over WiFi. For efficiency, modern generation WiFi uses packet aggregation where multiple packets from the network layer are combined into a single medium access control (MAC) layer packet of a larger size of up to 1 MB. [44]. If we use smaller-sized 64 bytes UDP challenge packets, WiFi MAC aggregation is affected reducing the throughput i.e., θ_0 for the challenger. To address this, we aggregate multiple signatures and send it as a single large packet. We use 1514 bytes challenge packets i.e., b in Eq. (1) is 1514 bytes (1472 byte UDP payload with a 42-byte header) which contain 23 different 64 byte signatures. Payload consists of all signatures since if any content were in plain text, it may be omitted by malicious challenger thereby reducing the packet size. We use the OpenSSL based implementation of Ed25519 [2].

Hashing and verification. As described in Algorithm 1, upon receiving the required number of total packets, the prover generates a hash for each challenger C_i, i.e., $h_{1i} \leftarrow Hash(M[i])$ where $M[i]$ is the set of all the signatures received from challenger C_i. The prover then generates a $MerkleRoot$ of all the hashes from all the challengers. For generating the hash we use $sha256$ hash function via the implementation [6] and for generating the Merkle root, we use a C++ open source implementation [7]. The prover sends h_{1i} and $MerkleRoot$ as response to the challenger C_i. The response packet is a UDP packet with a payload of 64 bytes as it contains two 256-bit hashes. In the verification phase (Algorithm 2), the prover sends bitmap B_i and Merkle proof P_i to challenger C_i, who then verifies the Merkle proof and sends RTT Δ_i and number of its packets received by the prover, to the verifier.

Precomputing the signatures. Signature generation incurs computation time too and our benchmarking of OpenSSL implementation [2] of Ed25519 indicates that generating one signature of 64 bytes takes about 50-60 microseconds (μs) on a resource-constrained Linux system consisting of 1 GB of RAM and 1 CPU core. For each packet, a challenger has to generate 23 signatures which will incur a maximum time of $23 * 60 \approx 1.4$ ms. As the signature generation time is more than the packet transmission time of about 1.2 ms even at $\theta_0 = 10$ Mbps, in our implementation challengers precompute all the signatures before the challenge begins. This can be done after the challenger receives the challenge request and while measuring the ping latency l_i.

Benchmarking the technique. Making use of multiple challengers with the additional requirement of security introduces more sources of errors. Particularly, l_i is not a constant and has some jitter. NTP synchronization can result in error of tens of milliseconds over the Internet. Computation overhead of hash and Merkle tree generation adds delay. Given these sources of error, we evaluate the accuracy as a function of challenger duration and the number of challengers.

5.3 Security Analysis of Implemented Protocol

Taking practical factors such as packet drop rate bound r and an overall latency bound L (including time synchronization error, transmission latency and computation overhead) into consideration, we analyze the security properties of our implemented protocol below.

Theorem 3 (Soundness). *In implemented protocol, by adjusting the correction factor to $(n - 2f - rf)/(n - f) - L\theta_0/(kb)$, the prover cannot inflate the bandwidth when $f < n/3$.*

Proof. When at least $n - f$ honest challengers send packets at rate θ_0, according to the bandwidth condition in Eq.2, the backhaul link will never be empty. To account for packet drops, challengers are required to send $(1+r)k$ packets, which means at most $(1+r)fk$ packets can be sent by corrupted challengers without actually consuming the bandwidth (in rushing attack or information sharing attack). Thus, the number of packets sent by uncorrupted challengers is at least $(n - 2f - rf) \cdot k$.

For handling latency, note that the median Δ' of the RTTs is at least $(n - 2fk)b/\theta_\mathsf{P} - L$, whereby

$$\begin{aligned}\theta'_\mathsf{P} &\le \frac{(n-f)kb}{(n-2f-rf)kb/\theta_\mathsf{P} - L} \cdot \frac{(n-2f-rf)kb - L(n-f)\theta_0}{(n-f)kb} \\ &\le \theta_\mathsf{P}.\end{aligned}$$

□

Theorem 4 (Approximate completeness). *When $f < n/3$ and the prover is uncorrupted, the implemented protocol will always output bandwidth with accuracy $\alpha' = \theta'_\mathsf{P}/\theta_\mathsf{P} \ge \frac{(n-2f-rf)kb/(n-f)-L\theta_0}{(1+r)kb+L\theta_0}$.*

Proof. In presence of latency and packet drops, the maximum median RTT is $\Delta' = (n-f)(1+r)kb/\theta_\mathsf{P} + L$. But the total number of packets are still no less than $(n-f)k$. Consequently, the bandwidth of honest challenger output by the protocol is at least

$$\begin{aligned}\theta'_\mathsf{P} &= \frac{(n-f)kb}{(n-f)(1+r)kb/\theta_\mathsf{P} + L} \cdot \frac{(n-2f-rf)kb - L(n-f)\theta_0}{(n-f)kb} \\ &\ge \frac{(n-2f-rf)kb - L(n-f)\theta_0}{(n-f)(1+r)kb + L(n-f)\theta_0}\theta_\mathsf{P} \\ &= \frac{(n-2f-rf)kb/(n-f) - L\theta_0}{(1+r)kb + L\theta_0}\theta_\mathsf{P}.\end{aligned}$$

□

6 Experimental Evaluation

In this section, we first evaluate how the use of multiple challengers can accurately measure the available bandwidth at the backhaul. For this, we consider only the setting where all participants are honest. We also highlight how existing per-hop capacity estimation techniques fail to give accurate results for backhaul of 100 Mbps or more in §6.1. Also, we stress-test our experiments under Byzantine attacks to evaluate the security of the protocol in §6.2.

Experimental setup. Our setup consists of a diverse set of challengers in terms of computation capability and geographical location. We carry out experiments with a maximum of ten challengers. The prover has backhaul bandwidth (θ_P) of 250 Mbps enforced by Linux rate limiter `tc`. The details of the prover and different challengers are listed in Table 1. Challengers 1-3 are connected to the Internet via WiFi links, while other challengers have a wired link.

6.1 Performance Evaluation

First, we benchmark the accuracy of our measurements by carrying out experiments without corrupted challengers. We evaluate the performance in the presence of corrupted challengers later in §6.2.

Measurement accuracy with all honest participants. To study how accuracy varies with challenge duration and the number of challengers, we conduct experiments by adjusting the number of selected challengers from 4 to 10 and challenge duration from 25 ms to 200 ms for the prover.

	Compute Parameters		Location	RTT
	RAM (GB)	CPU		(ms)
Prover	1	1	AWS X	
Ch. 1-3	12-16	4-8	Y	25
Ch. 4-5	1	1	AWS Z	198
Ch. 6-10	1	1	AWS X	1

Table 1: Experimental setup details for the prover and challengers. Location of the nodes are in different continents and are anonymized.

Challenge duration is the time required to transmit the required amount of packets i.e., $(n-f)k$ packets through the prover backhaul. Individual challengers will take longer to complete the challenge due to their latency, l_i, and the fact that they send $1.1k$ packets to account for packet drops. We rate-limit the prover backhaul to 250 Mbps using the Linux utility tc [5] and set the bandwidth of each challenger (θ_0) as θ_P/n, where n is the number of challengers.

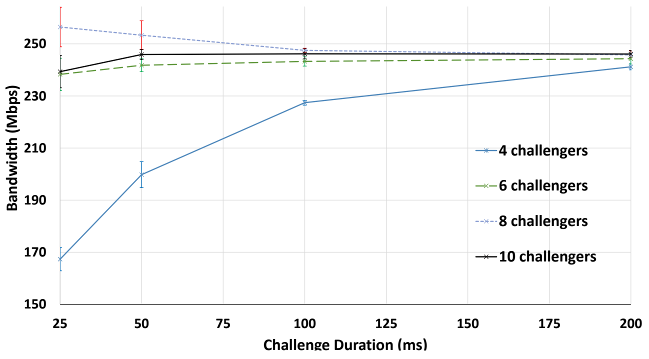


Figure 4: Backhaul measured by our technique for different challenge durations. Error bars ≈ std. deviation.

Fig. 4 shows the backhaul measured by our technique for varying number of challengers and challenge durations as 25 ms, 50 ms, 100 ms and 200 ms. For each challenge duration, ten experiments are carried out. We plot the average and standard deviation for ten experiments in Fig. 4.

As can be seen from Fig. 4, with the number of challengers set to 4, the measured backhaul is only about 167 Mbps for 25 ms challenge duration, but when the challenge duration is increased to 200 ms, the measured backhaul increases to about 241 Mbps with an error of about 4%. On the other hand, when the number of challengers is increased to 6 or more, the measured backhaul has an error of less than 5%, even for 25 ms challenge. However, the standard deviation for 25 ms and 50 ms experiments is higher. The measurement accuracy increases and the standard deviation decreases, if the challenge duration is increased to 100 ms or more. For 100 ms, we observe an error of less than 5% for six or more challengers.

We look at how measurement accuracy is affected as a function of the challenge duration for the case of $n = 8$. Fig. 5 shows the

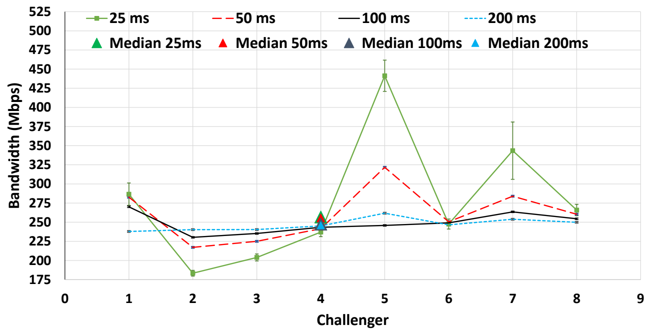


Figure 5: Backhaul measured by each challenger ($n = 8$) for different challenge durations. Error bars ≈ std. deviation.

backhaul measured by each of eight challengers, as an average across ten experiments for different challenge durations. As can be seen from Fig. 5, the backhaul measured by individual challenger shows higher error when challenge duration is 25 ms or 50 ms. For example, the backhaul measured by challenger 5 is about 440 Mbps and 320 Mbps for 25 ms and 50 ms duration. However, the error decreases when challenge duration is increased to 100 ms or more. Note that standard deviation across 10 experiments for each challenger also decreases as the challenger duration is increased.

Some ISPs use token bucket filter to rate limit the backhaul [53], whereby few packets can be released in a burst at much higher rate than the average backhaul speed. Since in our technique we use large number of packets (much more than the burst size of typical token bucket filter), we will still be able to measure the average backhaul which is our goal.

Sources of error. As shown in Fig. 5, it is interesting to note that some challengers measured the prover's backhaul as higher than the actual value of 250 Mbps. This is due to errors affecting measurement accuracy (see §5) such as time synchronization and jitters in latency. We observe that due to these errors, there is a time difference of 20-30 ms between the first packet from the first and the last challenger reaching the prover backhaul. To compensate for the maximum packet drop rate of 10% (see §5), each challenger sends 10% more data. So, the challengers that start late might receive the response from the prover before they finish sending their share of challenge packets, if sufficient challenge packets have been received by the prover from the challengers that start early. Such late starting challenger's backhaul estimate may be higher than the actual value. However, the median evaluation at the final step, which is primarily designed for security, also provides robustness against such outliers. Consequently, our measurement accuracy increases as we increase the number of challengers.

The computation overhead of hash and Merkle tree generation also adds to the measurement error. We observe that the computation overhead for the case of 4 challengers for 25 ms challenge duration is about 500 μs, while for 10 challengers for 200 ms challenge duration is about 3ms.

Amount of data. For the PoB protocol designed to handle the packet drop rate of 10%, the total amount of data required for different challenge duration for prover backhaul of 250 Mbps is given in Table 2.

Expt. Duration (ms)	25	50	100	200
Data (MB)	0.86	1.71	3.44	6.88

Table 2: Amount of challenge data required.

As seen from Fig. 5, for challenges with 100-ms duration we get a good accuracy for each challenger. Thus, our results show that our technique can measure 250 Mbps backhaul in 100 ms with about 3.5 MB of data and an error of less than 5%, when 6 or more uncorrupted challengers are involved.

Comparison with a single challenger. With a single challenger that has a bandwidth of 250 Mbps, we could measure prover backhaul of 250 Mbps with less than 2% error with challenge duration being only 10 ms and the amount of data required is about 345 KB. Multichallenger technique requires larger challenge duration due to the aforementioned errors. As the duration of the challenge is longer, the amount of data used correspondingly increases. But the primary benefit of multichallenger technique is that each challenger requires much smaller bandwidth. With ten challengers, each challenger requires a bandwidth of only 25 Mbps to measure prover backhaul of 250 Mbps.

Accuracy for larger prover backhauls. The experimental results of accuracy for larger prover backhauls (500 Mbps to 1000 Gbps) with 10 challengers are tabulated in Table 3. We observe that the measurement error grows as prover backhaul increases; however even for prover backhaul of 1000 Mbps, the measurement error is less than 8%.

Backhaul (Mbps)	500	750	1000
Measured BW (Mbps)	474.7	705.4	921.4
Error (%)	5.1	5.9	7.9

Table 3: Measured bandwidth for larger prover backhauls.

Backhaul (Mbps)	500	750	1000
Overhead (ms)	4.6	7.3	10.2

Table 4: Computation overhead

One reason for higher measurement error as prover backhaul increases is the increasing computation time for hash and Merkle tree construction. Table 4 shows the computation overhead for various prover backhauls. The computation overhead for a prover with 1000 Mbps is about 10 ms which is 10% of the challenge duration of 100 ms. These experiments suggest that as the prover backhaul increases, the computation overhead increases. So for even larger prover backhaul than 1000 Mbps, the challenge duration should be increased.

Effect of cross traffic. Our PoB protocol terminates when $(n - f)k$ packets are received by the prover. The number of packets k sent by each challenger is determined by the prover backhaul and challenge duration. However, if there is cross-traffic, the available bandwidth at the prover will be reduced and the challenge packets may experience a larger drop rate than 10% that we assume for our experiments. In this situation the experiment may not terminate.

We propose a modification to measure the available bandwidth in the presence of cross traffic, up to a fixed accuracy δ. The protocol repeats the basic PoB protocol, but instead of verifying θ_{P}, it verifies iteratively $\theta_{\mathsf{P}}^{(1)} = \delta, \theta_{\mathsf{P}}^{(2)} = 2\delta, ..., \theta_{\mathsf{P}}^{(\ell)} = \ell\delta$ and so on till θ_{P}. In more detail, we proceed as follows.

(1) At step i, execute multichallenger PoB protocol with $\theta_{\mathsf{P}}^{(i)} = i\delta$. Note that each challenger must release challenge traffic at rate $\theta_0 = \theta_{\mathsf{P}}^{(i)}/n$ at this step.
(2) Each challenger sets a timeout of 5× (challenge duration). If the response from the prover is not received during this period, the challenger declares `not terminate`. If majority of the challengers declare `not terminate`, we say that the protocol does not terminate.
(3) If the protocol for the ith step terminates, increment $i \leftarrow i+1$ and repeat the steps above.
(4) Else if the protocol for the ith step does not terminate, output the bandwidth obtained in the previous execution of the PoB protocol.

We assume that the amount of cross traffic does not vary throughout all the steps of the experiment. Using the approach above, we carried out experiments to measure available bandwidth in the presence of different amounts of cross-traffic. The backhaul of the prover is set to 250 Mbps and the number of challengers is 10. $\theta_{P,0}$ is set to 40 Mbps and δ to 20 Mbps.

Available BW (Mbps)	220	140	90
Measured BW (Mbps)	219.6	144.6	104.1

Table 5: Measured bandwidth in the presence of cross traffic.

Table 5 summarizes the results. The measured bandwidths are close to available bandwidths, except for 90 Mbps. The likely reason for this discrepancy is that the presence of challenge traffic reduces cross traffic, resulting in an overestimation of the available bandwidth.

Comparison with pathchar. As we outlined earlier, the performance of pathchar depends on how robust are minimum delay estimates over the Internet and how long will it take for us to get a robust estimate. Thus, to evaluate the performance of pathchar, we measure the RTT to the prover node using ping for 15 different packet sizes in multiples of 100 Bytes, starting from 100 Bytes and ending at 1500 Bytes. We ran the experiment five times and took 500 measurements for each packet size. Pathchar [17, 24] suggests taking the minimum RTT for each packet size and fitting the linear least squares line to the data.

Expt.	1	2	3	4	5
RTT	16.589	16.511	16.635	16.565	16.500

Table 6: Linear least squares line's intercept values for each experiment's dataset. RTT is in *ms*.

Figure 6 shows the minimum RTT (ms) versus packet size in bytes and the fitted line for the first experiment. Table 6 shows the y-intercepts for five experiments; note that the y-intercept represents

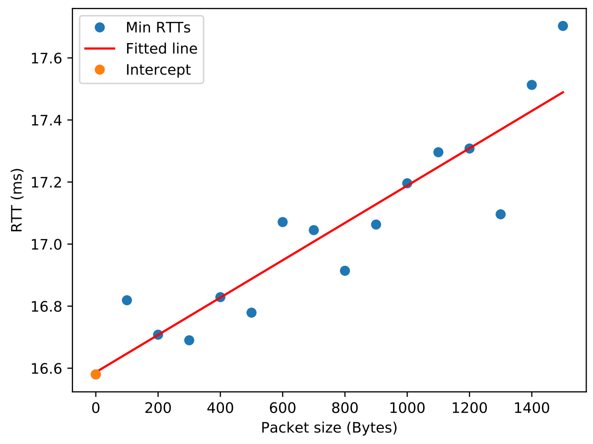


Figure 6: Minimum round trip time to the prover versus packet size in Bytes. The line shows the linear least squares fit.

latency. We can see that the y-intercepts have a difference of 50-100 ms. Thus, we can say that the jitters experienced over the internet is not negligible; in particular, we cannot estimate the minimum latency below accurancy of 50-100 microseconds. Consequently, it is not feasible to use pathchar to measure 100 Mbps or higher bandwidth.

6.2 Security Evaluation

We carry out experiments to study how measurement results are effected in the presence of malicious challengers. We choose total number of challengers as $n = 10$ and malicious challengers $f = 2$. We carry out measurements for two different prover backhauls of 100 Mbps and 250 Mbps, with a challenge duration of 100 ms.

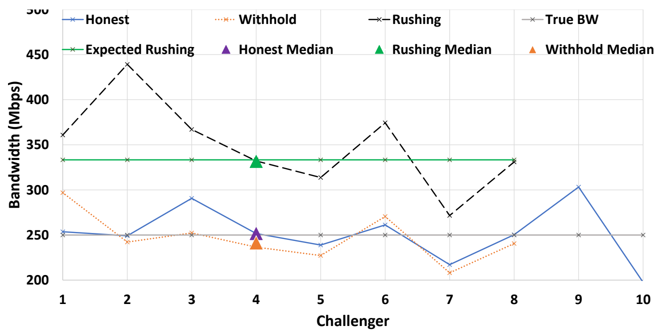


Figure 7: Backhaul measured by each challenger in case of withholding and rushing attack. Prover backhaul is 250 Mbps.

Fig. 7 shows the average bandwidth measured by each challenger across ten experiment runs for the case when prover backhaul is 250 Mbps. As can be seen from Fig. 7, the measured backhaul in the case when all challengers are honest (honest median in Fig. 7) is 251.5 Mbps, while the measured backhaul in the case of withholding attack (Withhold Median in Fig. 7) is 241.4 Mbps. The expected measured backhaul in case of withholding attack is 250 Mbps. So the measured accuracy in case of withholding attack is within 4%. Note that this is the accuracy of the measurement technique. Our PoB protocol will apply a correction factor $\alpha = (n - 2f)/(n - f)$ (Algorithm 2) and output the guaranteed bandwidth for the prover as $241.4\alpha \approx 181$ Mbps which is about 28% less than the prover backhaul of 250 Mbps.

In case of a rushing attack, the measured bandwidth is inflated to $250/\alpha \approx 333$ Mbps. The measured backhaul (rushing median in Fig. 7) 331.5 Mbps matches our theoretical prediction. The output of the protocol in this case will be $331.5\alpha \approx 249$ Mbps, which is only 1 Mbps less than the prover backhaul.

Note that our security guarantees require us to curtail bandwidth inflation. Indeed, we can observe that even under a rushing attack, the guaranteed bandwidth does not exceed the actual bandwidth. This is enabled by multiplying by a shrinkage factor to compensate for adversarial challengers trying to help the prover to claim an inflated bandwidth. However, this comes at the cost of lower guaranteed bandwidth even when all challengers are reporting honestly. After repeating the experiment for a backhaul of 100 Mbps, the results stay similar, validating our theoretical predictions.

7 Conclusion and Discussion

Summary. Trustfree telemetry is a central problem in decentralized networks. Our Proof of Backhaul protocol addresses a core requirement by providing a secure and accurate backhaul bandwidth measurement service for wireless access points while also allowing open participation. The protocol is operated by a group of challengers, whose latency and bandwidth can be ordinary, with the goal of measuring a prover hotspot who may have a high-bandwidth backhaul link. We have established a trust model for the PoB problem, designed precise specifications of the PoB protocol, and tested a high-performance, low-overhead implementation.

Improving accuracy. Our protocol guarantees soundness and completeness of backhaul measurements with a reasonable accuracy in the presence of Byzantine parties. The accuracy ratio $(1-2\beta)/(1-\beta)$ is determined by the Byzantine fraction due to a correction made for an unavoidable rushing attack – corrupted challengers can always rush their packets through an external high-bandwidth link to lower RTT and inflate backhaul bandwidth to be measured. However, such backdoor links may incur substantial costs in practice, necessitating a more relaxed threat model and a family of extended protocols. Without rushing links, we equip PoB protocols with a shuffle phase where a pair of challengers are asked to jointly sign packets. This mechanism improves accuracy by making information sharing attack harder in a probabilistic manner, with a cost of higher communication overhead for verification. Designing a secure and efficient proof structure for such a shuffle protocol is an active area of research.

Cross traffic. In our proposed method for handling cross-traffic in §6.1, we run experiments for increasing values of bandwidth below the claimed link capacity. This requires fresh data to be sent for each value and increases the amount of data needed. To reduce the data consumption, a naive approach could be that the prover replies to the challengers with the number of packets received in a fixed duration. An alternative approach is to send intermediate responses when appropriate amounts of data are received. Both approaches cannot guarantee a fixed accuracy for different available bandwidths. Designing a protocol which is more data efficient and has such guarantees is an open problem.

References

[1] [n. d.]. FAST Internet Speed Test. https://fast.com/. [Online; accessed 13-October-2022].
[2] [n. d.]. OpenSSL Ed25519 Implementation. https://www.openssl.org/docs/man1.1.1/man7/Ed25519.html. [Online; accessed 18-September-2022].
[3] [n. d.]. PM-WANI Central Registry. https://pmwani.gov.in/wani. [Online; accessed 18-September-2022].
[4] [n. d.]. Speedtest. https://www.speedtest.net. [Online; accessed 13-October-2022].
[5] 2001. TC - traffic control, Linux Manual. https://man7.org/linux/man-pages/man8/tc.8.html. [Online; accessed 18-September-2022].
[6] 2012. C++ SHA256 Implementation. http://www.zedwood.com/article/cpp-sha256-function. [Online; accessed 18-September-2022].
[7] 2015. https://github.com/IAIK/merkle-tree. [Online; accessed 18-September-2022].
[8] 2020. NTP: The Network Time Protocol. http://www.ntp.org/. [Online; accessed 18-September-2022].
[9] 2021. Multi-server testing. https://www.ookla.com/articles/how-ookla-ensures-accurate-reliable-data-2021. [Online; accessed 11-October-2022].
[10] David G Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. 2008. Accountable internet protocol (AIP). In *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*. 339–350.
[11] Suman Banerjee and Ashok K. Agrawala. 2000. Estimating available capacity of a network connection. In *Proceedings IEEE International Conference on Networks 2000 (ICON 2000). Networking Trends and Challenges in the New Millennium*. IEEE.
[12] Robert L. Carter and Mark E. Crovella. 1996. *Dynamic server selection using bandwidth probing in wide-area networks*. Technical Report. Boston University Computer Science Department.
[13] Robert L Carter and Mark E Crovella. 1996. Measuring bottleneck link speed in packet-switched networks. *Performance evaluation* 27 (1996), 297–318.
[14] Ignacio Castro, Aurojit Panda, Barath Raghavan, Scott Shenker, and Sergey Gorinsky. 2015. Route Bazaar: Automatic Interdomain Contract Negotiation. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. USENIX Association, Kartause Ittingen, Switzerland. https://www.usenix.org/conference/hotos15/workshop-program/presentation/castro
[15] Dah-Ming Chiu and Raj Jain. 1989. Analysis of the increase/decrease algorithms for congestion avoidance in computer networks. j-COMP-NET-ISDN, 17 (1): 1–14.
[16] John Douceur, Jon Howell, and John JD Douceur. 2003. Scalable Byzantine-fault-quantifying clock synchronization. (2003).
[17] Allen B Downey. 1999. Using pathchar to estimate Internet link characteristics. *ACM SIGCOMM Computer Communication Review* 29, 4 (1999), 241–250.
[18] FCC. 2020. *Title 47, Chapter I, Subchapter D, Part 96, Citizens Broadband Radio Service*. Regulatory Information. Federal Communications Commission. https://www.govinfo.gov/content/pkg/CFR-2020-title47-vol5/pdf/CFR-2020-title47-vol5-part96.pdf
[19] Mainak Ghosh, Miles Richardson, Bryan Ford, and Rob Jansen. 2014. *A TorPath to TorCoin: Proof-of-bandwidth altcoins for compensating relays*. Technical Report. NAVAL RESEARCH LAB WASHINGTON DC.
[20] Amir Haleem, Andrew Allen, Andrew Thompson, Marc Nijdam, and Rahul Garg. 2018. *Helium: A Decentralized Wireless Network*. White Paper. Helium Systems, Inc. http://whitepaper.helium.com
[21] Khaled Harfoush, Azer Bestavros, and John Byers. 2003. Measuring bottleneck bandwidth of targeted path segments. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies*.
[22] Ningning Hu and Peter Steenkiste. 2003. Evaluation and characterization of available bandwidth probing techniques. *Journal on Selected Areas in Communications* 21, 6 (2003), 879–894.
[23] Van Jacobson. [n. d.]. Traceroute. *https://linux.die.net/man/8/traceroute6* ([n. d.]).
[24] Van Jacobson. 1999. Pathchar. *ftp://ftp.ee.lbl.gov/pathchar/* (1999).
[25] Manish Jain and Constantinos Dovrolis. 2002. End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput. In *ACM SIGCOMM Computer Communication Review*.
[26] Manish Jain and Constantinos Dovrolis. 2002. Pathload: A measurement tool for end-to-end available bandwidth. In *In Proceedings of Passive and Active Measurements (PAM) Workshop*. Citeseer.
[27] Simon Josefsson and Ilari Liusvaara. 2017. *Edwards-curve digital signature algorithm (EdDSA)*. Technical Report.
[28] Ghassan O Karame, Boris Danev, Cyrill Bannwart, and Srdjan Capkun. 2012. On the security of end-to-end measurements based on packet-pair dispersions. *IEEE Transactions on Information Forensics and Security* 8, 1 (2012), 149–162.
[29] Srinivasan Keshav. 1991. A control-theoretic approach to flow control. In *Proceedings of the conference on Communications architecture and protocols*.

[30] Hyojoon Kim and Nick Feamster. 2013. Improving network management with software defined networking. *IEEE Communications Magazine* 51, 2 (2013), 114–119.
[31] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. 2014. Software-defined networking: A comprehensive survey. *Proc. IEEE* 103, 1 (2014), 14–76.
[32] Kevin Lai and Mary Baker. 2000. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of the conference on applications, technologies, architectures, and protocols for computer communication.* 283–294.
[33] Kevin Lai and Mary Baker. 2001. Nettimer: A tool for measuring bottleneck link bandwidth. In *3rd USENIX Symposium on Internet Technologies and Systems (USITS 01).*
[34] magma [n. d.]. Magma: A modern mobile core network solution. https://magmacore.org Magma Core Foundation.
[35] B. A. Mah. 2000. pchar: A tool for measuring internet path characteristics. *http://www. employees. org/ bmah/Software/pchar/* (2000).
[36] Bob Melander, Mats Bjorkman, and Per Gunningberg. 2000. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Globecom'00-IEEE. Global Telecommunications Conference.* IEEE.
[37] Bob Melander, Mats Bjorkman, and Per Gunningberg. 2002. Regression-based available bandwidth measurements. In *International Symposium on Performance Evaluation of Computer and Telecommunications Systems.*
[38] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques.* Springer, 369–378.
[39] David L Mills. 1989. On the accuracy and stablility of clocks synchronized by the network time protocol in the internet system. In *ACM SIGCOMM Computer Communication Review.*
[40] oran [n. d.]. ORAN: Transforming the Radio Access Networks Towards Open, Intelligent, Virtualized and Fully Interoperable RAN. https://www.o-ran.org O-RAN Alliance e.V..
[41] Christos Pappas, Raphael M Reischuk, and Adrian Perrig. 2015. FAIR: Forwarding accountability for Internet reputability. In *2015 IEEE 23rd International Conference on Network Protocols (ICNP).* IEEE, 189–200.
[42] Attila Pasztor and Darryl Veitch. 2002. Active probing using packet quartets. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment.* 293–305.
[43] Tao Peng, Christopher Leckie, and Kotagiri Ramamohanarao. 2003. Protection from distributed denial of service attacks using history-based IP filtering. In *IEEE International Conference on Communications, 2003. ICC'03.*, Vol. 1. IEEE, 482–486.
[44] Eldad Perahia and Robert Stacey. 2013. *Next generation wireless LANs: 802.11 n and 802.11 ac.* Cambridge university press.
[45] Ravi Prasad, Constantine Dovrolis, Margaret Murray, and KC Claffy. 2003. Bandwidth estimation: metrics, measurement techniques, and tools. *IEEE network* 17, 6 (2003), 27–35.
[46] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. 2003. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and active measurement workshop.*
[47] Khondaker M Salehin and Roberto Rojas-Cessa. 2013. Packet-pair sizing for controlling packet dispersion on wired heterogeneous networks. In *2013 International Conference on Computing, Networking and Communications (ICNC).* IEEE, 1031–1035.
[48] Robin Snader and Nikita Borisov. 2009. EigenSpeed: secure peer-to-peer bandwidth evaluation.. In *IPTPS.* 9.
[49] Jacob Strauss, Dina Katabi, and Frans Kaashoek. 2003. A measurement study of available bandwidth estimation tools. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement.* ACM.
[50] Jehan Tremback and Justin Kilpatrick. 2017. *Althea: An incentivized mesh network protocol.* White Paper. Althea Network, Inc. https://github.com/althea-net/althea-whitepaper/blob/master/whitepaper.pdf
[51] Xinlei Yang, Hao Lin, Zhenhua Li, Feng Qian, Xingyao Li, Zhiming He, Xudong Wu, Xianlong Wang, Yunhao Liu, Zhi Liao, et al. 2022. Mobile access bandwidth in practice: measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2022 Conference.* 114–128.
[52] Xinlei Yang, Xianlong Wang, Zhenhua Li, Yunhao Liu, Feng Qian, Liangyi Gong, Rui Miao, and Tianyin Xu. 2021. Fast and light bandwidth testing for internet users. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21).* 1011–1026.
[53] Ertong Zhang and Lisong Xu. 2015. Capacity and token rate estimation for networks with token bucket shapers. *Computer Networks* 88 (2015), 1–11.
[54] Peng Zhou, Rocky KC Chang, Xiaojing Gu, Minrui Fei, and Jianying Zhou. 2015. Magic train: design of measurement methods against bandwidth inflation attacks. *IEEE Transactions on Dependable and Secure Computing* 15, 1 (2015), 98–111.

A Extended Protocol

In this section, we investigate variants of our protocol and variations in security guarantees when different primitives such as throughput fairness (§ A.1), digital signatures (§ A.2) and extra side-links (§ A.3) are present or absent.

A.1 Stronger Primitive: Fairness

We first consider a stronger primitive when using TCP packets as our challenge flow, whose congestion control algorithm is known to provide fairness [15]. Fairness guarantees that when n challengers are sending traffic simultaneously with the same rate to a link with capacity B, each should have an average rate of B/n. Under fairness assumption, we modify the packet-based termination rule (prover generates the first response on receiving enough packets) of the protocol to a time-based rule that prover will wait for a fixed amount of time Δ before sending the first response. On receiving the first response, challengers stop timer and measure their average throughput during the measurement period. In this variant, the final output is the robust sum of throughput reports. In particular, we take the median throughput from n collected reports and multiply it by $(n-f)$ to get the estimated bandwidth. The report collection phase lasts long enough to make sure all honest reports get delivered, and the throughput of those who never submit reports is set to a default value 0. Since honest challengers send the challenge traffic at rate $\theta_0 = \theta_{\mathsf{P}}/(n-f)$ as is specified in the protocol, they are guaranteed to share the backhaul bandwidth in a same rate $\theta_0' \leq \theta_0$ (it is possible that $\theta_0' < \theta_0$ when more than $n-f$ challengers generate traffic).
Security analysis. Since honest challengers share the same throughput $\theta_0' \leq \theta_0$, and the median of throughput is bounded by honest reports, the output bandwidth must be no larger than $(n-f)\theta_0' \leq (n-f)\theta_0 = \theta_{\mathsf{P}}$, thus the soundness holds. When more than $(n-f)$ challengers send the packets, honest rates will become lower than θ_0. But since anyone who send more than $\theta_0\Delta$ will be considered as malicious, to stay covert corrupted challengers can only send up to $\theta_0\Delta$ packets during the measurement phase. In other words, the final bandwidth has accuracy $1-\beta$, which is better than the accuracy $(1-2\beta)/(1-\beta)$ of the current protocol (Theorem 2) due to fairness.

A.2 Weaker Primitive: Without Signature

In our main PoB protocol, digital signatures are used to generate unforgeable packets and resolve disputes between prover and challengers in terms of the number of packets that are indeed received by the prover. We now discuss a case where digital signature schemes are not available, instead, packets are generated by a pseudo random generator with a seed picked by each challenger. We describe the key changes in the protocol below. The protocol guarantees an accuracy of $(1-3\beta)/(1-\beta)$.
Measurement phase. At the beginning of measurement phase, challengers generate a seed S_i and commit it to the verifier. Same as the main protocol, each challenger generates a sequence of k packets with rate θ_0. The q-th packet from the challenger C_i contains the sequence number q and q-th random number R_{iq} generated from seed S_i. The responses are generated in the same way as Algorithm 1.

Verification phase. After the measurements, each challenger reveals the seed S_i to the prover, who can check whether it matches the pre-committed seed from the verifier. Challengers verify responses by constructing Merkle tree just as Algorithm 2. The key change in the verification phase is the termination check. The verifier accepts the output as long as at least $(n-2f)$ packets are reported as received since f corrupted challengers may refuse to report the actual number of packets received by the prover and the prover can not prove the inconsistency, though the honest prover is required to terminate after receiving $(n-f)k$ packets. As a consequence, the correction factor becomes $(1-3\beta)/(1-2\beta)$.
Security analysis. The soundness proof of the new protocol is similar to the proof in Theorem 1. The only difference caused by the new termination rule is that an honest prover may get only $(n-2f)k$ reported packets while they actually receive $(n-f)k$ packets (corrupted challengers all report 0). Different from digital signatures, random packets generated by pseudo random generator does not keep unforgeability after the seed is revealed, thus there is no way to resolve disputes. However, during the measurement phase, the packets generated by honest challengers are still unforgeable, therefore even corrupted prover needs to receive at least $(n-3f)k$ packets from the honest challengers to generate responses, which means the median time $\Delta' \geq (n-3f)kb/\theta_{\mathsf{P}}$ and

$$\theta'_{\mathsf{P}} \leq \frac{(1-3\beta)(n-2f)kb}{(1-2\beta)(n-3f)kb/\theta_{\mathsf{P}}} = \theta_{\mathsf{P}}$$

The termination is guaranteed for honest provers due to the relaxed termination rule with the cost of lower accuracy. When the prover is uncorrupted, it still waits for $(n-f)k$ packets to generate the response. Under witholding attacks and the misreporting attacks, the honest prover can only prove the receipt of $(n-2f)k$ packets. Therefore, the protocol will output

$$\theta'_{\mathsf{P}} \geq \frac{(1-3\beta)(n-2f)kb}{(1-2\beta)(n-f)kb/\theta_{\mathsf{P}}} \geq \frac{(1-3\beta)}{(1-\beta)}\theta_{\mathsf{P}}$$

So accuracy is reduced to $\alpha = (1-3\beta)/(1-\beta)$.

A.3 Relaxed Threat Model

In §4.1, we assume adversary has access to extra high-bandwidth channels to every end node in the Internet, which enables unavoidable rushing attacks. In real world, though, maintaining such a backdoor network can be prohibitively expensive. A more realistic privilege adversary may have is a high-bandwidth channel to the network core. This means packets sent by corrupted participants can reach network core almost instantly, but will be added to the message queue together with other honest packets and be transmitted to destination following the order in queue.
Shuffle phase. With the relaxed threat model, we design a new class of protocols which further improve the accuracy ratio. When there are no extra links connecting the adversary with other nodes directly, corrupted challengers are not able to rush packets through extra links. But information share attack is still possible, to address which we design a shuffle phase to involve more than one challengers into the packets generating process.

We consider a slightly different system where challengers join a pool with shares and the number of shares of a challenger C_i is determined by its available bandwidth θ_i and the bandwidth unit θ_0. For example, a challenger with bandwidth 50 Mbps owns 5 shares in a pool with bandwidth unit 10 Mbps. Each share is assigned with a unique ID $\{1, \cdots, U\}$ (we assume $U \to \infty$). The Byzantine fraction of corrupted bandwidth units $\beta < \frac{1}{3}$.

During the shuffle phase, t shares are drew from the pool uniformly randomly to form a challenger group, we call such a protocol t-shuffle scheme. Suppose we have n challenger groups sampled from the pool, denoted as $\mathsf{C}_1, \cdots, \mathsf{C}_n$. Specifically, the owner of each bandwidth unit in the challenger group signs a sequence of packets using the same method in Algorithm 1, and sends these packets to the next challenger in the group, who also signs the packets to get a new sequence of packets. Then in the measurement phase, the last challenger serves as the sender to forward the sequence of packets signed by the entire group to the prover.

In this way, as long as one of the challengers in the group is uncorrupted, the information share attack no longer works since corrupted challengers can only share partial information about how to generate packets to the prover. In other words, only those groups in which all bandwidth units belong to corrupted challengers can still mount the information share attack. On the other hand, the multi-signed packets scheme implies that even one corrupted challenger can withhold the packets. We call a challenger group is good if all bandwidth units in the group belong to honest challengers, and a challenger group is bad if all bandwidth units in the group belong to corrupted challengers. We denote the number of good and bad groups as random variables G and B. For each challenger group C_i,

$$P(\mathsf{C}_i \text{ is good}) = (1-\beta)^t$$
$$P(\mathsf{C}_i \text{ is bad}) = \beta^t$$

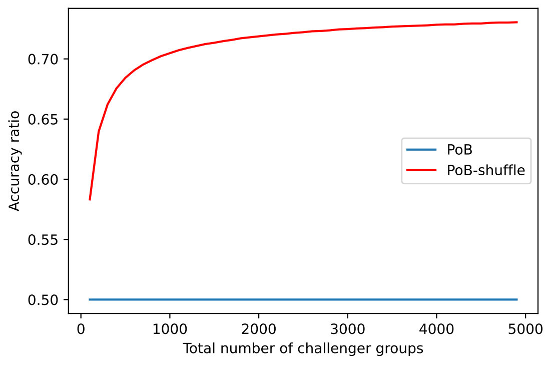
Full protocol. After shuffle phase, the last challengers in all groups forward packets to the prover and start timer for transmission process. The time to send each packet is the same as Algorithm 1. On collecting $cnt \geq g = (\theta_{\mathsf{P}}/\theta_0)k$ packets (g is a liveness parameter), the prover computes the hash of all packets (from different challengers) as the first response, receiving which challengers will stop timer and log time. In the verification phase, prover need to broadcast all received packets to all challengers so that they can verify the hash and submit reports to the verifier. Then similar to Algorithm 2, verifier waits for time reports from at least $n-f$ challengers, and takes the median of RTT Δ' as time measurement. The final bandwidth of output is given by the following formula.

$$\theta'_{\mathsf{P}} \leftarrow \frac{cnt \cdot p \cdot (g-b)}{\Delta' \cdot g} \tag{3}$$

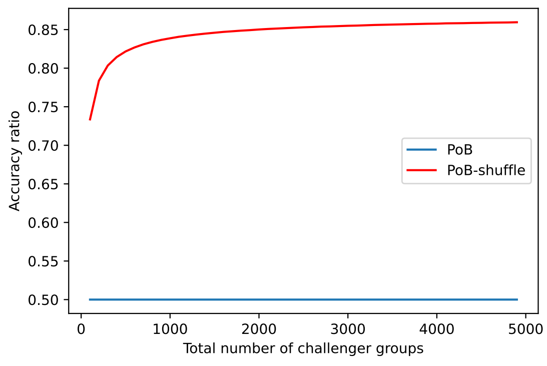
where g is the liveness parameter, b is the accuracy parameter, p is the packet size.
Security analysis. The parameters defined in equation (3) determines the security and accuracy of PoB with shuffle. Formally, to provide soundness we require

$$\Pr(\theta'_{\mathsf{P}} > \theta_{\mathsf{P}}) = \Pr\left(\frac{g-b}{g-B} > 1\right) = \Pr(B > b) \tag{4}$$

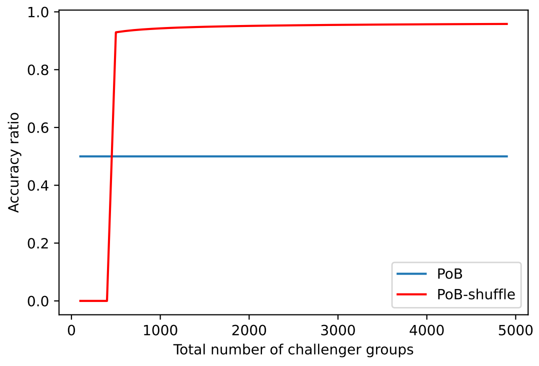
to be negligible. Meanwhile, we want to achieve approximate completeness and maximize the accuracy ratio $(g-b)/g$, which implies $\Pr(G < g)$ should also be negligible to ensure termination. Among



(a) $t = 2$



(b) $t = 3$



(c) $t = 5$

Figure 8: The comparison of accuracy ratio between PoB and t-shuffle protocols. Byzantine ratio $\beta = 1/3$, we calculate the best accuracy ratio under different number of challenger groups n. The maximal failure probability of PoB-shuffle protocol is $\epsilon = 0.5$.

n challenger group, we have

$$G = \sum_{i=1}^{n} \mathbb{I}[\texttt{C}\text{ is good}]$$
$$B = \sum \mathbb{I}[\texttt{C}\text{ is bad}]$$
$$E[G] = (1-\beta)^t n$$
$$E[B] = \beta^t n$$

where $\mathbb{I}$ is an indicator function. According to Chernoff bounds, for any $0 < \delta_g \leq 1$ and $0 < \delta_b \leq 1$, we have

$$P(G < (1-\delta_g)(1-\beta)^t n) = P(G < g) < \exp(-\Theta(\delta_g^2 n))$$
$$P(B > (1+\delta_b)\beta^t n) = P(B > b) < \exp(-\Theta(\delta_b^2 n))$$

Thus by choosing appropriate g, b, we get a protocol with accuracy

$$\alpha = \frac{g-b}{g} = 1 - \frac{(1+\delta_b)\beta^t}{(1-\delta_g)(1-\beta)^t}$$

We calculate the case when $\beta = 1/3$ and set error probability $\epsilon = 0.5$. We search for parameters g, b to reach optimal accuracy ratio α given different number of challenger groups $n \in [100, 5000]$. The results are shown in Figure 8.