Magic Train: Design of Measurement Methods Against Bandwidth Inflation Attacks (Supplemental Material)

Peng Zhou[‡], Rocky K. C. Chang[§], Xiaojing Gu[†], Minrui Fei[‡] and Jianying Zhou[ξ]
Shanghai University[‡], The Hong Kong Polytechnic University[§],
East China University of Science and Technology[†], The Institute for Infocomm Research[ξ]
pzhou@shu.edu.cn, csrchang@comp.polyu.edu.hk, xjing.gu@ecust.edu.cn, mrfei@staff.shu.edu.cn, jyzhou@i2r.a-star.edu.sg

APPENDIX A
NOTATION TABLE

TABLE 1: A summary of the main notations used in magic train paper.

Notations	Meanings
(p_i, q_i)	Round-trip packet consisting of a request packet p_i and a response q_i
$\|p_i\|$	The size of p_i
t_i	The round-trip delay of (p_i, q_i)
Δt_{ij}	$\Delta t_{ij} = t_j - t_i$, the time dispersion between two round-trip packets
Δt	The time dispersion between two consecutive round-trip packets
b_{ij}	Bandwidth calculated using the packets from (p_i, q_i) to (p_j, q_j)
P	$P = \{(p_i, q_i) \| i = 1, 2, \ldots, N\}$, a round-trip packet train
N	The number of packets (or the length) of a packet train
$Pr[N]$	The probability of generating a packet train of length N
$b(P)$	Bandwidth calculated using P
G_k	The k-th time an attacker guesses the length of a packet train
$\mathbb{P}$	$\mathbb{P} = \{P^m \| 1 \leq m \leq M\}$, a set of magic trains
P^m	The m-th packet train in $\mathbb{P}$
M	The number of packet trains in a train set
b^m	Bandwidth calculated using the m-th magic train in $\mathbb{P}$
$V(b)$	The standard deviation of b^ms over $\mathbb{P}$
t_i^m	The i-th round-trip delay of the m-th packet train in the train set $\mathbb{P}$
c_i	The capacity measured using the time dispersion $\Delta t_{(i-1)i}$
$V(c)$	The standard deviation of c_is in P
H_b	The threshold used for Multi-Magic-Train based Detection (MMTD)
H_t	Threshold for RTT Pair/Estimation based Detection (RTTPD/RTTED)
H_d	The threshold used for packet dropping detection
H_c	The threshold used for magic delay algorithm

APPENDIX B
THE PROOF OF EQUATION (5)

Proof. We consider the adversary can have only the knowledge of $\hat{N}$ and $\check{N}$. At the 1-st round guessing, she/he can just choose a random value between $\hat{N}$ and $\check{N}$ to estimate N. Obviously, the probability that the adversary can successfully guess N at this round is:

$$Pr[G_1 = N|N] = \frac{1}{\hat{N} - \check{N}}$$

If the adversary can successfully guess N at the K-th round, all the first $K-1$ rounds of guessing should be failed. The summary of guessed values for those failed guessing should be smaller than N (i.e., $\sum_{k=1}^{K-1} G_k < N$), because the adversary cannot successfully guess N at the K-th round if $\sum_{k=1}^{K-1} G_k > N$. We consider the adversary is smart enough. (S)He can intelligently narrow the guessing range from $\hat{N} - \check{N}$ to $\hat{N} - \sum_{i=1}^{k} G_i$ when (s)he guesses the remaining length $N - \sum_{i=1}^{k} G_i$ after k rounds of failed guessing. By this consideration, the probability that the adversary succeeds in guessing $N - \sum_{k=1}^{K-1} G_k$ at the K-th round is:

$$Pr_K = Pr[G_K = N - \sum_{k=1}^{K-1} G_k | N - \sum_{k=1}^{K-1} G_k] = \frac{1}{\hat{N} - \sum_{k=1}^{K-1} G_k}$$

While the probability that the adversary fails in guessing $N - \sum_{i=1}^{k-1} G_k$ at the k-th round ($1 < k \leq K-1$) is:

$$\overline{Pr}_k = Pr[G_k < N - \sum_{i=1}^{k-1} G_i | N - \sum_{i=1}^{k-1} G_i] = \frac{N - \sum_{i=1}^{k-1} G_i}{\hat{N} - \sum_{i=1}^{k-1} G_i}$$

where $\overline{Pr}_1 = Pr[G_1 < N|N] = \frac{N - \check{N}}{\hat{N} - \check{N}}$.

Therefore, the probability that the adversary successfully guess N at the K-th round (where $K > 1$) is:

$$Pr[\sum_{k=1}^{K} G_k = N|N] = Pr_K \cdot \overline{Pr}_1 \cdot \prod_{k=2}^{K-1} \overline{Pr}_k$$

Eqn. (5) is proved. □

APPENDIX C
MAGIC TRAIN IMPLEMENTATION

C.1 Prototype Implementation

To investigate the effectiveness of our magic train design, we have developed a prototype tool in Linux. The tool employs RAW socket [1] to generate measurement packets and calls LibPcap [2] to monitor measurement traffic. We also implement the proposed detection algorithms in this tool (i.e., MMTD, RTTPD, RTTED and magic delay algorithm). Figure 1 shows the flowchart of our implementation. The tool takes an input M from users to generate M magic trains for measurement, and runs corresponding algorithms to detect possible bandwidth inflation attacking variants in order. Our implementation contains $2,657$ lines of C++ code in total (reported by CLOC [3]). The code is open sourced*. Moreover,

*. The source code is given at https://github.com/zpbrent/magictrain

to emulate magic train's effectiveness against inflation attacks, we also implement an attacking suite which consists of all the bandwidth inflation attacking variants based on libnetfilter_queue library [4].

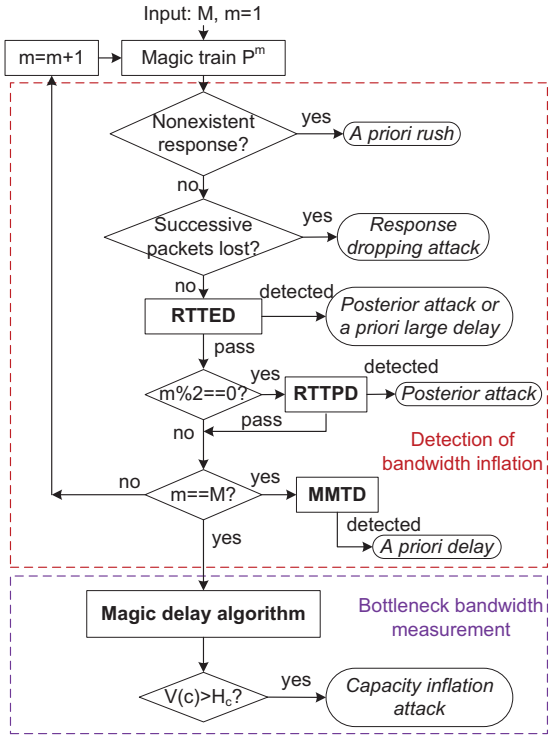


Fig. 1: Flowchart of magic train prototype implementation.

C.1.1 Implementation tune-up

In our magic train implementation, we generate the TCP packets with a variable payload to balance the measurement intrusive characteristic and accuracy. In fact, since we implement our magic train for commodity computers and run it in the user space of Linux operating system, the time resolution we can accurately capture cannot be smaller than one millisecond (we have also confirmed this limitation in our experiments). As the size of an TCP packet with full payload is 1.5Kbytes or say 12Kbits, the maximum capacity we can accurately measure using magic delay algorithm is 12Mbps (our magic delay algorithm uses consecutive packet pair dispersion for calculation, see Eqn. (11) in Section 4.4.1) and the maximum available bandwidth we can accurately measure using MMTD algorithm is $12 \times \check{N}$Mbps (our MMTD algorithm calculates available bandwidth using the time dispersion between the first and last packets in each magic train, see Eqn. (4) in Section 4.1). Since we set $\check{N} = 10$ in our experiments, the maximum bandwidth we can accurately measure is 120Mbps. It is worth noting that the 120Mbps (for securing available bandwidth measurement) and the 12Mbps (for securing capacity measurement) are not the design limitations of our magic train and magic delay algorithm, they are implementation constraints. For example, if we implement our methods in NetFPGA platform [5] which can achieve one nanosecond time resolution, our magic train can accurately measure a $12 \times \check{N}$Tbps network path and our magic delay algorithm can handle a 12Tbps capacity. We leave this implementation in our future research.

Moreover, if we generate a magic train with $\check{N} = 10$ full payload packets for measurement, it can result a burst of 120Kbits traffic on the network path. This traffic may not be considered as intrusive if the network path is 12Mbps, but it is actually a heavy load to a path on 120Kbps or lower. To make our magic train measurement non-intrusive as much as possible, we propose an adaptive payloading method which can generate magic train packet with payloads depending on real-time bandwidth measurement results. That is, magic train is initiated with full payload packets but will reduce the payloads in the next round of measurement if the first measurement reports a lower bandwidth. To ensure the one millisecond time resolution, we can calculate the updated payload length as $|p_i| = \frac{b(P)}{1000 \times \check{N}}$, where $|p_i|$ is the size of magic train measurement packet, $b(P)$ is the bandwidth reported by the first round of measurement and $\check{N}$ is the possibly shortest length of magic train.

APPENDIX D
DISCUSSION

Here, we further discuss some potential design choices and limitations of our magic train. In particular, we first discuss how to estimate an appropriate threshold for our detection in Section 6.1 (see the main manuscript). We then explain why we cannot use the bandwidth from a verifier to a prover's nearby honest hosts to approximate the bandwidth to the prover in Section D.1. At the end, we analyze how we can extend our magic train to detect uplink bandwidth inflation attacks in Section D.2.

D.1 Why not measure bandwidth from nearby hosts

In the design of our magic train, we assume there exist some honest hosts near the target prover (e.g., the same country, the same ISP or the same /24 etc), and moreover exploit these hosts to estimate the true RTT from a verifier to the prover to avoid RTT manipulation (see the RTTED algorithm in Section 4.2.1). Careful readers may concern why we do not estimate the bandwidth directly from these honest hosts and hence disclose inflation attacks. The major reason has two folds. First, unlike the RTT which is mainly determined by the cable length from the verifier to the prover, the bandwidth (or capacity) is determined by the bottleneck device in between the verifier and the prover. Although the verifier has the similar cable length to the prover and the prover's nearby hosts, it is not necessary that the verifier can share the same bottleneck device with the prover and the prover's nearby hosts. For example, if the bottleneck device is the prover itself, none of its nearby hosts can share this bottleneck to the verifier. Second, since RTT of the first measurement packet is an increasing function of the length of magic train under posteriori/large-delay-a-priori attacking conditions, we can increase the length of our magic trains to have a large enough difference between the enlarged RTT and the estimated RTT for detection. However, unlike the RTT case, enlarging the length of magic trains cannot necessarily cause a more

obvious difference between the inflated bandwidth and the estimated bandwidth for detection. We therefore cannot ensure the detection accuracy if we use the bandwidth estimated from nearby hosts to detect inflation attacks.

D.2 Defeating uplink bandwidth inflation attack

As discussed in Section 2.2, our magic train can be slightly modified to detect the potential uplink bandwidth inflation, even though the train is originally designed for detecting downlink bandwidth inflation. In contrast to downlink bandwidth inflation, the uplink bandwidth cannot be enlarged by provers. Instead, a network device located in between the verifier and uplink bottleneck should take this attacking job. We have discussed in Section 2.2 that this kind of attack happens with a small probability, since benign network devices lack enough incentive to do this attacking while the compromise of a network device nearby a particular verifier is difficult and expensive even if the attacker is an experienced hacker. However, we also study here to extend our magic train to tackle this kind of uplink bandwidth inflation behaviors. The big challenge of this extension is that the attacking device is close to the verifier and can receive trigger packet (Figure 1(b)) before prover's responses. As a result, if the attacker can decode the trigger packet and infer prover's responses in advance, she/he can generate fake response packets with arbitrary inter-packet delay, and hence enlarge the reported uplink bandwidth to any value. Magic train cannot detect such a strong inflation since it will neither cause a large delay of the first response nor induce irregular dispersions. The key to this problem is preventing the attacking device from knowing the content of trigger packet and guessing the response behavior of the prover. For this reason, we can implement our magic train using HTTPS protocol, which can encrypt the trigger packets and thwart the attacking device to impersonate as prover. It is worth noting that magic train over HTTPS cannot defeat uplink bandwidth inflation if the prover and the attacking device collude. We leave the study of this collusion based uplink bandwidth inflation attack as an open problem for future research.

APPENDIX E
DETECTION RESULTS

E.1 Testbed topology

We deploy a testbed in our Lab for evaluation. This testbed consists of five computers to emulate a verifier, a prover, two cross traffic generators and a network router. We deploy a Dummynet [6] in the router emulating machine to emulate various networking conditions, such as bandwidth, delay, packet loss rate and reordering rate etc. We also install D-ITG [7] to generate cross traffic with different patterns and average bitrates. We show our testbed topology in Figure 2.

E.2 How smart adversaries defeat symmetry-based bandwidth inflation detection

The basic idea of CDF symmetry-based detection algorithm is to split cumulative distribution function (CDF) of a number

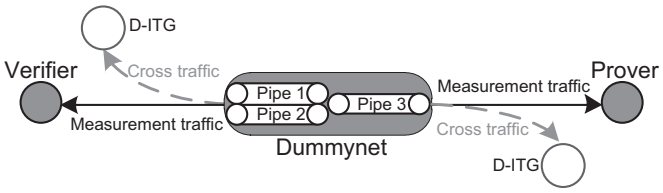


Fig. 2: Testbed topology.

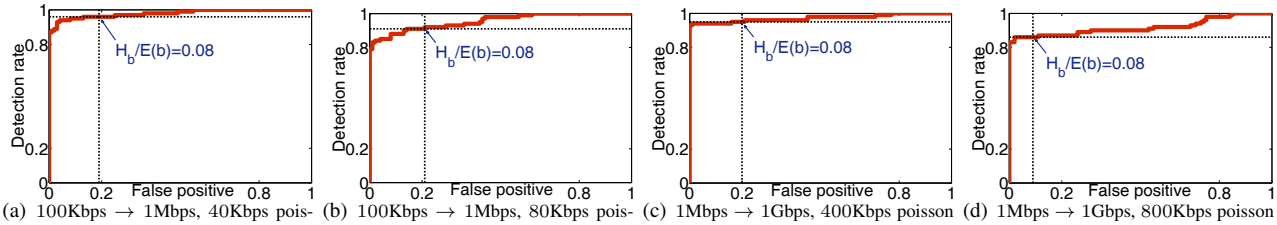
of packet pair measurement dispersion (i.e., Δt) samples into two sub distributions at the point where CDF$= 0.5$ (i.e., upper CDF> 0.5 and lower CDF< 0.5). Then calculating the Jeffery divergence (JD) factor to compare whether these two sub CDFs are similar. A smaller JD factor indicates the two distributions are more similar and the measurement is more likely of being attacked. To investigate how this algorithm becomes ineffective in the presence of smart adversaries, we slightly modify our magic train tool to implement packet pair method and also add a new traffic shaper module to our attacking tool set. We then conduct $1,000$ rounds of packet pair measurement experiments on our testbed configured with a 100Kbps bandwidth limit and a cross traffic whose pattern follows poisson distribution and average bitrate is 40Kbps. Our attacking tool is deployed to inflate this 100Kbps bandwidth to $1,000$Kbps. We calculate the JD factor strictly following the algorithm described in [8]'s IV.C section. When we disable our attacking tool, the JD factor of the normal measurement is 102.8. While when we enable naive attack, the JD factor directly drops to 0.0086 which causes the attack being detected. Moreover, when we launch smart attack by inserting some churns to mimic cross traffic's impacts, we successfully increase the JD factor to 124.9 which is larger than the normal one 102.8. Figure 3 demonstrates how our churn mimicking technique can defeat the symmetry-based detection. As can be seen, by inserting mimicking churns, we can largely increase the asymmetry between the upper CDF and lower CDF (Figure 3(b)), and can also successfully increase the bandwidth to the expected value $1,000$Kbps (Figure 3(d)).

E.3 How magic delay algorithm filters out queuing delays

We give an example to show how our magic delay algorithm can filter out queuing delays caused by cross traffic and thus can measure capacity in adversarial network conditions in Figure 4. In this example, our magic delay algorithm leads all the $\Delta t_{12}, \Delta t_{23}, \ldots, \Delta t_{56}$ to converge to the value around 0.12 second ($1.5 \times 8Kbits/100Kbps = 0.12second$, red crossing symbol in Figure 4) after $M = 50$ rounds of minimize operation over each t_i.

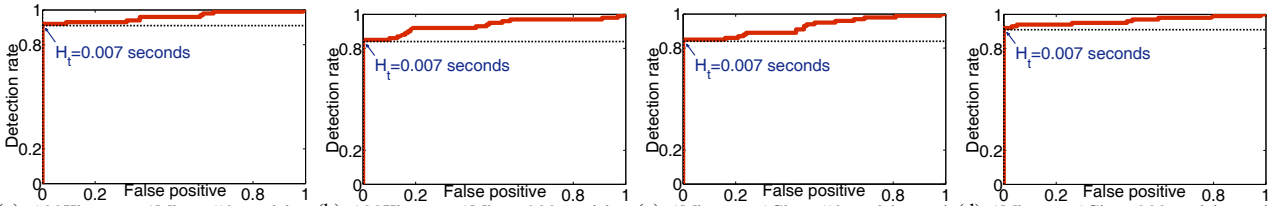
E.4 Detection results on testbeds

We show the detection results obtained from the experiments on our testbeds in Figure 5, Figure 6, Figure 7, Table 2, Figure 8 and Table 3.



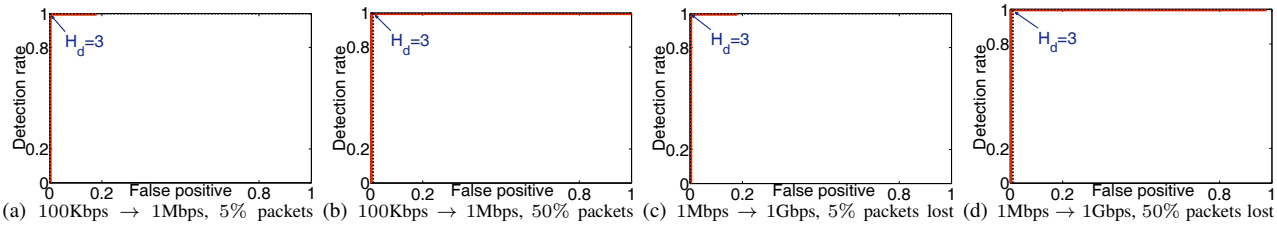
(a) 100Kbps → 1Mbps, 40Kbps poisson cross traffic. (b) 100Kbps → 1Mbps, 80Kbps poisson cross traffic. (c) 1Mbps → 1Gbps, 400Kbps poisson cross traffic. (d) 1Mbps → 1Gbps, 800Kbps poisson cross traffic.

Fig. 5: The ROC plots for a priori delay attack detection using MMTD. When H_b equals to 0.08 times of $E(b)$, MMTD can keep the detection rate larger than 85% and meanwhile keep the false positive smaller than 20%. (50ms delay, 5% packets lost and reordered)



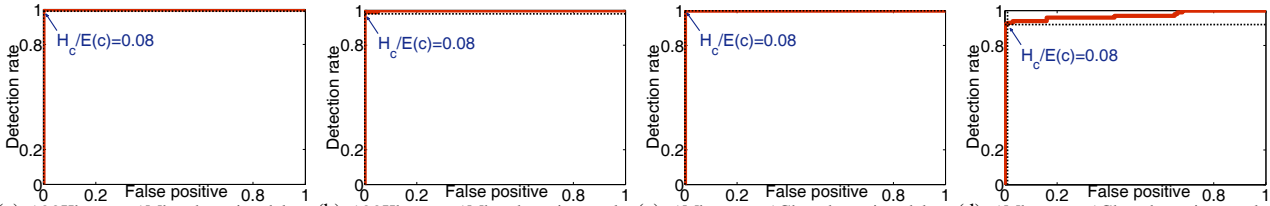
(a) 100Kbps → 1Mbps, 50ms delay and 40Kbps poisson cross traffic. (b) 100Kbps → 1Mbps, 200ms delay and 40Kbps poisson cross traffic. (c) 1Mbps → 1Gbps, 50ms delay and 400Kbps poisson cross traffic. (d) 1Mbps → 1Gbps, 200ms delay and 400Kbps poisson cross traffic.

Fig. 6: The ROC plots for posteriori attack detection using RTTPD. When H_t equals to 0.007 seconds, RTTPD can keep the detection rate larger than 85% and meanwhile keep the false positive 0%. (5% packets lost and reordered)



(a) 100Kbps → 1Mbps, 5% packets lost and 40Kbps poisson cross traffic. (b) 100Kbps → 1Mbps, 50% packets lost and 40Kbps poisson cross traffic. (c) 1Mbps → 1Gbps, 5% packets lost and 400Kbps poisson cross traffic. (d) 1Mbps → 1Gbps, 50% packets lost and 400Kbps poisson cross traffic.

Fig. 7: The ROC plots for packet dropping attack detection. When H_d equals to 3 consecutive dropping packets, we can keep the detection rate larger than 95% and meanwhile keep the false positive smaller than 5%. (50ms delay)



(a) 100Kbps → 1Mbps by prior delay, 80Kbps poisson cross traffic. (b) 100Kbps → 1Mbps by mix attack, 80Kbps poisson cross traffic. (c) 1Mbps → 1Gbps by prior delay, 800Kbps poisson cross traffic. (d) 1Mbps → 1Gbps by mix attack, 800Kbps poisson cross traffic.

Fig. 8: The ROC plots for capacity inflation attack detection using magic delay algorithm. When H_c equals to 0.08 times of $E(c)$, we can keep the detection rate larger than 90% and meanwhile keep the false positive smaller than 5%. (50ms delay, 5% packets lost and reordered)

E.5 Detection results on PlanetLab and the Internet

E.5.1 PlanetLab nodes setup

We list the information for the PlanetLab nodes we choose for our evaluation, as well as the network statistics from these Planetlab nodes to a computer hosted by our Hong Kong Lab (158.132.255.34) in Table 4. In our experiments, we deploy our magic train implementation in our Lab host 158.132.255.34 and launch our attacking tool in the four PlanetLab nodes for the evaluation.

E.5.2 Detection results on PlanetLab nodes

We show the detection results from our PlanetLab experiments in Figure 9, Figure 10 and Figure 11.

TABLE 4: Statistical information for the PlanetLab nodes we use.

Domain name	IP	RTT	Est. BW
planetlab2.cqupt.edu.cn	202.202.43.199	75ms	95.16Mbps
csplanetlab4.kaist.ac.kr	143.248.55.129	70ms	62.95Mbps
pl1.eng.monash.edu.au	130.194.252.8	277ms	82.65Mbps
planetlab6.goto.info.waseda.ac.jp	133.9.81.166	55ms	80.75Mbps

E.5.3 Accuracy of RTT estimation algorithm for RTTED

Our RTTED algorithm highly relies on RTT estimation algorithm to guarantee its detection capability. For this reason, we investigate how well our RTT estimation algorithm runs on the Internet. In particular, we choose the top 1,000 Tor routers ranked by up time [9], the Alexa top 1,000 web sites

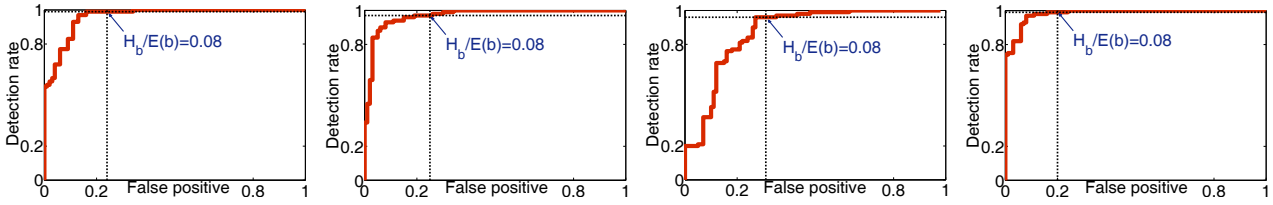
TABLE 2: Magic train's detection rates and false positives for detecting bandwidth inflation attacks in our controlled testbed. For posteriori attack detection, the results of RTTED algorithm are put at the left-hand side of |, while the results of RTTPD algorithm are put at the right-hand side of |.

Attacking type	Bandwidth		Cross traffic		Network condition			Detection result	
	Real	Inflated	Pattern	Ave. bitrate	Ave. round-trip delay	Loss rate	Reorder rate	Detection rate	False positive
A priori delay	100Kbps	1Mbps	None	None	50ms	0%	0%	100%	0%
A priori delay	100Kbps	1Mbps	None	None	50ms	5%	0%	100%	0%
A priori delay	100Kbps	1Mbps	None	None	50ms	10%	0%	100%	0%
A priori delay	100Kbps	1Mbps	None	None	50ms	0%	5%	100%	0%
A priori delay	100Kbps	1Mbps	None	None	50ms	0%	10%	100%	0%
A priori delay	100Kbps	1Mbps	None	None	50ms	5%	5%	100%	0%
A priori delay	100Kbps	1Mbps	Normal distribution	40Kbps	50ms	0%	0%	91%	0%
A priori delay	100Kbps	1Mbps	Normal distribution	80Kbps	50ms	0%	0%	90%	12%
A priori delay	100Kbps	1Mbps	Poisson distribution	40Kbps	50ms	0%	0%	93%	0%
A priori delay	100Kbps	1Mbps	Poisson distribution	80Kbps	50ms	0%	0%	90%	15%
A priori delay	100Kbps	1Mbps	Poisson distribution	40Kbps	50ms	5%	5%	86%	0%
A priori delay	100Kbps	1Mbps	Poisson distribution	80Kbps	50ms	5%	5%	85%	4%
A priori delay	1Mbps	1Gbps	None	None	50ms	0%	0%	100%	0%
A priori delay	1Mbps	1Gbps	None	None	50ms	5%	5%	100%	0%
A priori delay	1Mbps	1Gbps	Poisson distribution	400Kbps	50ms	0%	0%	86%	0%
A priori delay	1Mbps	1Gbps	Poisson distribution	400Kbps	50ms	5%	5%	86%	0%
A priori delay	1Mbps	1Gbps	Poisson distribution	800Kbps	50ms	5%	5%	87%	19%
A priori rush	100Kbps	1Mbps	None	None	50ms	0%	0%	100%	0%
A priori rush	100Kbps	1Mbps	Poisson distribution	40Kbps	50ms	5%	5%	100%	0%
A priori rush	1Mbps	1Gbps	None	None	50ms	0%	0%	100%	0%
A priori rush	1Mbps	1Gbps	Poisson distribution	400Kbps	50ms	5%	5%	100%	0%
Posterior attack	100Kbps	1Mbps	None	None	50ms	0%	0%	100%|91%	0%|0%
Posterior attack	100Kbps	1Mbps	None	None	50ms	5%	5%	95%|94%	0%|0%
Posterior attack	100Kbps	1Mbps	Poisson distribution	40Kbps	50ms	5%	5%	94%|92%	0%|0%
Posterior attack	100Kbps	1Mbps	Poisson distribution	40Kbps	100ms	5%	5%	99%|93%	0%|0%
Posterior attack	100Kbps	1Mbps	Poisson distribution	40Kbps	200ms	5%	5%	93%|85%	0%|0%
Posterior attack	1Mbps	1Gbps	None	None	50ms	0%	0%	100%|93%	0%|0%
Posterior attack	1Mbps	1Gbps	None	None	50ms	5%	5%	86%|91%	0%|0%
Posterior attack	1Mbps	1Gbps	Poisson distribution	400Kbps	50ms	5%	5%	89%|85%	0%|0%
Posterior attack	1Mbps	1Gbps	Poisson distribution	400Kbps	100ms	5%	5%	87%|89%	0%|0%
Posterior attack	1Mbps	1Gbps	Poisson distribution	400Kbps	200ms	5%	5%	87%|92%	0%|0%
A priori large delay	100Kbps	1Mbps	None	None	50ms	0%	0%	100%	0%
A priori large delay	100Kbps	1Mbps	None	None	50ms	5%	5%	100%	0%
A priori large delay	100Kbps	1Mbps	Poisson distribution	40Kbps	50ms	5%	5%	100%	0%
A priori large delay	100Kbps	1Mbps	Poisson distribution	40Kbps	100ms	5%	5%	100%	0%
A priori large delay	100Kbps	1Mbps	Poisson distribution	40Kbps	200ms	5%	5%	100%	0%
A priori large delay	1Mbps	1Gbps	None	None	50ms	0%	0%	100%	0%
A priori large delay	1Mbps	1Gbps	None	None	50ms	5%	5%	100%	0%
A priori large delay	1Mbps	1Gbps	Poisson distribution	400Kbps	50ms	5%	5%	100%	0%
A priori large delay	1Mbps	1Gbps	Poisson distribution	400Kbps	100ms	5%	5%	98%	0%
A priori large delay	1Mbps	1Gbps	Poisson distribution	400Kbps	200ms	5%	5%	97%	0%
Response dropping	100Kbps	1Mbps	None	None	50ms	0%	0%	100%	0%
Response dropping	100Kbps	1Mbps	Poisson distribution	40Kbps	50ms	5%	5%	100%	0%
Response dropping	100Kbps	1Mbps	Poisson distribution	40Kbps	50ms	50%	5%	100%	1%
Response dropping	1Mbps	1Gbps	None	None	50ms	0%	0%	100%	0%
Response dropping	1Mbps	1Gbps	Poisson distribution	400Kbps	50ms	5%	5%	100%	0%
Response dropping	1Mbps	1Gbps	Poisson distribution	400Kbps	50ms	50%	5%	100%	1%

TABLE 3: Magic delay algorithm's detection rates and false positives for detecting capacity (bottleneck bandwidth) inflation in our controlled testbed.

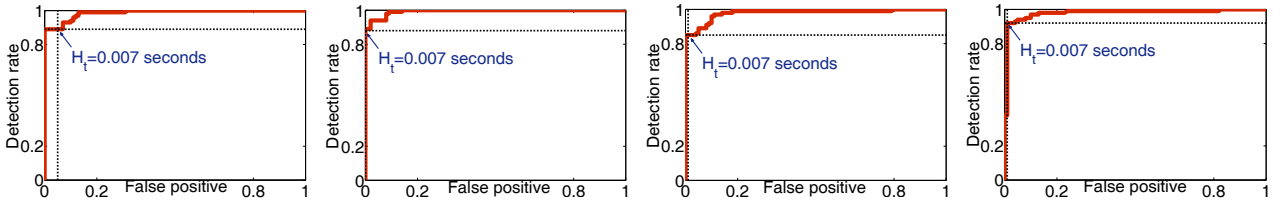
Attacking type	Bandwidth		Cross traffic		Network condition			Detection result	
	Real	Inflated	Pattern	Ave. bitrate	Ave. round-trip delay	Loss rate	Reorder rate	Detection rate	False positive
A priori delay	100Kbps	1Mbps	Poisson distribution	80Kbps	50ms	5%	5%	100%	0%
A priori delay	1Mbps	1Gbps	Poisson distribution	800Kbps	50ms	5%	5%	100%	1%
Mixed attack	100Kbps	1Mbps	Poisson distribution	80Kbps	50ms	5%	5%	99%	0%
Mixed attack	1Mbps	1Gbps	Poisson distribution	800Kbps	50ms	5%	5%	93%	1%

[10] and random $1,000$ BitTorrent peers from a featured file seed [11] as our prover IPs, since those networking systems usually balance their services according to bandwidth. We measure the RTT from a verifier located in our Lab to each of those prover IPs, as well as the RTTs to prover IPs' $\backslash 24$ host, nearby router (the second last recognized hop reported by traceroute), the same country hosts (in average) and the same ISP (AS code is the same) hosts (in average). Figure 12 illustrates scatter plot for our RTT estimation algorithms on Tor routers. Each red point represents a prover's RTT (x axis) and its corresponding estimated RTT (y axis). A red point locating at the dark blue diagonal line indicates our algorithm can accurately estimate the RTT of a prover, while the red point staying with the two dark dot lines means our



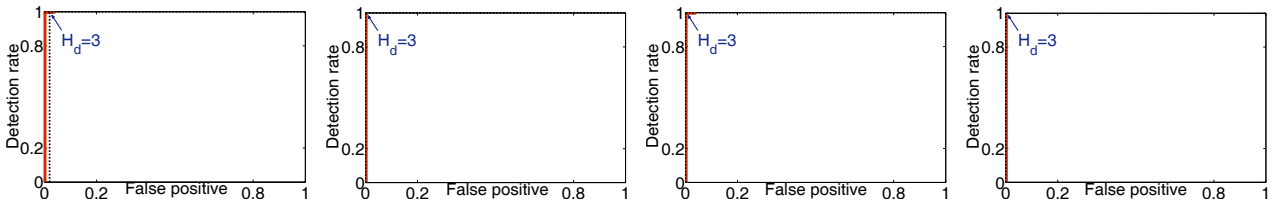
(a) planetlab2.cqupt.edu.cn. (b) csplanetlab4.kaist.ac.kr. (c) pl1.eng.monash.edu.au. (d) planetlab6.goto.info.waseda.ac.jp.

Fig. 9: The ROC plots for a priori delay attack detection using MMTD on PlanetLab nodes. When H_b equals to 0.08 times of $E(b)$, MMTD can keep the detection rate larger than 85% and meanwhile keep the false positive less than 30%.



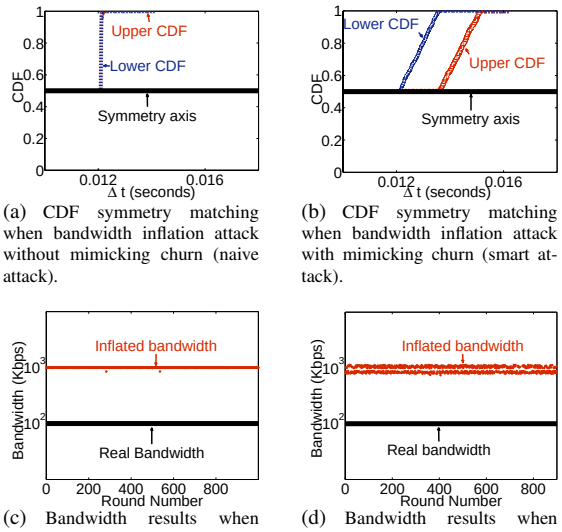
(a) planetlab2.cqupt.edu.cn. (b) csplanetlab4.kaist.ac.kr. (c) pl1.eng.monash.edu.au. (d) planetlab6.goto.info.waseda.ac.jp.

Fig. 10: The ROC plots for posteriori attack detection using RTTPD on PlanetLab nodes. When H_t equals to 0.007 seconds, RTTPD can keep the detection rate larger than 80% and meanwhile keep the false positive less than 10%.



(a) planetlab2.cqupt.edu.cn. (b) csplanetlab4.kaist.ac.kr. (c) pl1.eng.monash.edu.au. (d) planetlab6.goto.info.waseda.ac.jp.

Fig. 11: The ROC plots for packet dropping attack detection on PlanetLab nodes. When H_d equals to 3 consecutive dropping packets, we can keep the detection rate nearly 100% and meanwhile keep the false positive less than 5%.



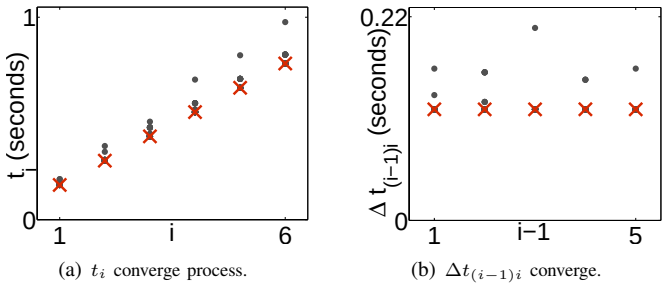
(a) CDF symmetry matching when bandwidth inflation attack without mimicking churn (naive attack).

(b) CDF symmetry matching when bandwidth inflation attack with mimicking churn (smart attack).

(c) Bandwidth results when bandwidth inflation attack without mimicking churn (naive attack).

(d) Bandwidth results when bandwidth inflation attack with mimicking churn (smart attack).

Fig. 3: How smart adversaries defeat symmetry-based bandwidth inflation detection.



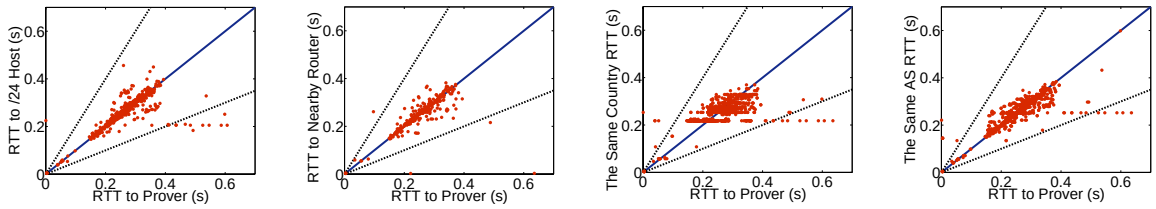
(a) t_i converge process. (b) $\Delta t_{(i-1)i}$ converge.

Fig. 4: How magic delay algorithm filters out cross traffic.

estimated RTT is within $\pm 50\%$ distance from the real RTT. As can be seen, more than 95% Tor router's estimated RTTs are within this $\pm 50\%$ range. Moreover, we also show the RTT estimation results of Alexa top $1,000$ web sites and random $1,000$ BitTorrent peers in Figures 13 and 14, respectively. Though the estimated RTTs within $\pm 50\%$ range for Alexa web sites still keep in around 90%, they drop to about 80% for BitTorrent peers. This leads us to use a larger H_t to guarantee detection capability in BitTorrent, as well as a longer train (i.e., a larger N) to reduce potential false positives. Note that, in Table 2, we choose $H_t = 2.5 \times t_1$ (i.e., $\pm 150\%$ range) to show our RTTED results.

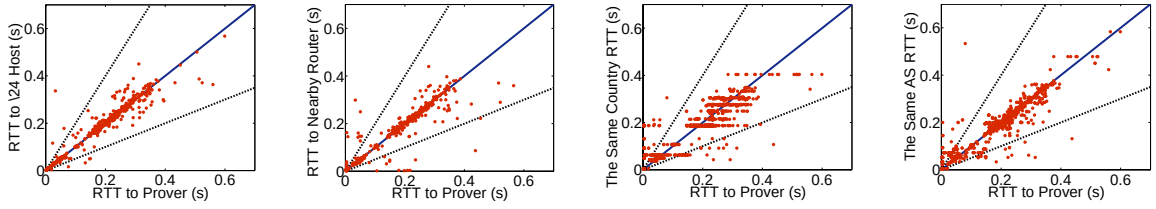
E.5.4 Supporting rate of round-trip linkable trains

We note that our design of round-trip linkable trains may not be fully supported by all Internet machines, since some of



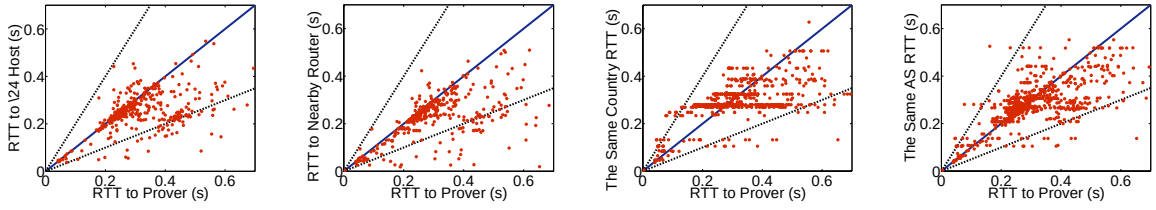
(a) \24 hosts. (b) Nearby router. (c) The same country. (d) The same ISP (AS).

Fig. 12: RTT estimation for Top $1,000$ Tor routers.



(a) \24 hosts. (b) Nearby router. (c) The same country. (d) The same ISP (AS).

Fig. 13: RTT estimation for Top $1,000$ Alexa web servers.



(a) \24 hosts. (b) Nearby router. (c) The same country. (d) The same ISP (AS).

Fig. 14: RTT estimation for random $1,000$ BitTorrent peers.

them (e.g., OF-Data-Train and OF-SYN-Train) do not follow legitimate TCP protocol and some others (e.g., IF-Time-Train) rely on TCP optional extension fields. Illegitimate TCP flows may be blocked by enterprise firewalls or not be responded by some TCP stacks, while TCP options may be disabled by some TCP implementations on default. For this reason, we test the support rates of our three candidate round-trip linkable trains. Our test runs on the same IP sets as our RTT estimation accuracy experiment (i.e., top $1,000$ Tor routers [9], Alexa top $1,000$ sites [10] and random $1,000$ BitTorrent peers [11]). We list the experimental results at Table 5. As can be seen, though some train candidates cannot be well supported by some systems (only 30.11% BitTorrent peers support OF-Data-Train), the overall supporting rates are good enough ($> 96\%$).

TABLE 5: Supporting rate of round-trip linkable train in the Internet.

	Tor routers	Alexa web sites	BitTorrent peers
OF-Data-Train	87.46%	72.26%	30.11%
OF-SYN-Train	93.73%	90.25%	60.98%
IF-Time-Train	97.37%	69.75%	90.85%
Either of Above	98.80%	96.98%	97.08%

REFERENCES

[1] Linux programmer's manual - raw(7) [online]. http://man7.org/linux/man-pages/man7/raw.7.html, 2012.

[2] The libpcap project [online]. http://sourceforge.net/projects/libpcap/, 2013.

[3] Cloc: Count lines of code [online]. http://cloc.sourceforge.net/, 2013.

[4] Netfilter [online]. http://www.iptables.org/projects/libnetfilter_queue/index.html, 2013.

[5] John W Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, and Jianying Luo. NetFPGA–an open platform for Gigabit-rate network switching and routing. In *Microelectronic Systems Education, 2007. MSE'07. IEEE International Conference on*, pages 160–161, 2007.

[6] The dummynet project [online]. http://info.iet.unipi.it/~luigi/dummynet/, 2013.

[7] D-itg, distributed Internet traffic generator [online]. http://traffic.comics.unina.it/software/ITG/, 2013.

[8] Ghassan Karame, Boris Danev, Cyrill Bannwart, and Srdjan Capkun. On the security of end-to-end measurements based on packet-pair dispersions. *IEEE Transactions on Information Forensics and Security*, 2013.

[9] Joseph B. Kowalski and Kasimir Gabert. Tor network status [online]. http://torstatus.blutmagie.de/, 2014.

[10] Alexa - actionable analytics for the web [online]. http://www.alexa.com/, 2014.

[11] Bittorrent - delivering the world's content [online]. http://www.bittorrent.com/, 2014.