TFRC: TCP Friendly Rate Control using TCP Equation Based Congestion Model

CS 218 W 2003

Oct 29, 2003

References

- S. Floyd, J. Padhye, J.Widmer “Equation Based Congestion Control for Unicast Applications”, Sigcomm 2000

- J. Padhye, V.Firoiu, D. Towsley, J. Kurose” Modeling TCP Throughput: a Simple Model and its Empirical Validation” Sigcomm 98

Congestion Control of UDP streaming traffic

- Uncontrolled UDP major threat to Internet stability
- Best effort streaming traffic must be rate-controlled in a way that it is **TCP-friendly**
- Existing schemes (eg, RAP – Rate Adaptation Protocol) do not include retx timeout and slow start; some use AIMD and window halfing (too abrupt)
- Also, some schemes do not scale as they react to each packet loss
- TFRC is TCP friendly in that it adjusts the rate by “mimicking” a TCP Reno connection using the **TCP “equation” model;** it provides smooth rate adaptation

TCP Equation Model (Padhye et al)

The equation was derived for TCP Reno;
it relates source rate (Throughput) T to:

•Round trip delay R (measured at source)
•Packet size s (measured at source)
•Packet loss (congestion) rate p (fed back by rcv each RTT)
•Retransmission time out t_{RTO} (measured at source)

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)}$$

Key Idea of TFRC

- Sender receives the feed back re **packet loss event rate p** from receiver every RTT
- Sender calculates **new value** of allowed sending rate; it increases/decreases current value to match the calculated rate
- In so doing, TFRC behaves like any other TCP Reno session (same equation); it produced the same external effects

Background on TCP cong. control Equation (from J. Padhye et al)

- A simple model relating T to RTT and p already existed (Floyd) – but did not account for TCP time out

$$B(p) = \frac{1}{RTT}\sqrt{\frac{3}{2bp}} + o(1/\sqrt{p})$$

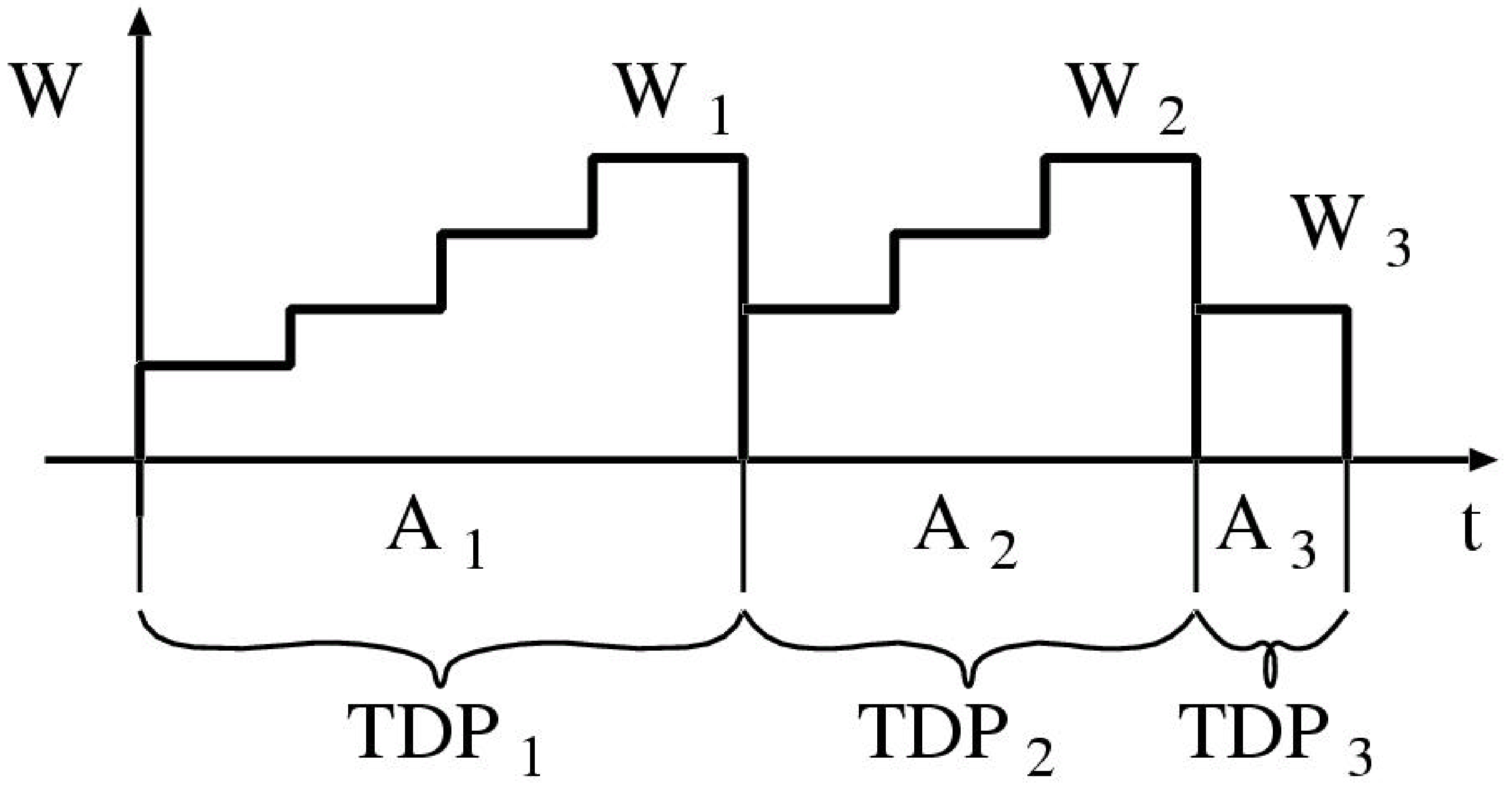
The main innovation of Padhye's work is to include the **Trto** and the **advertised window Wmax**

Trto is important as most of packet loss leads to Time out, rather than 3 Dup ACKs

The equation model

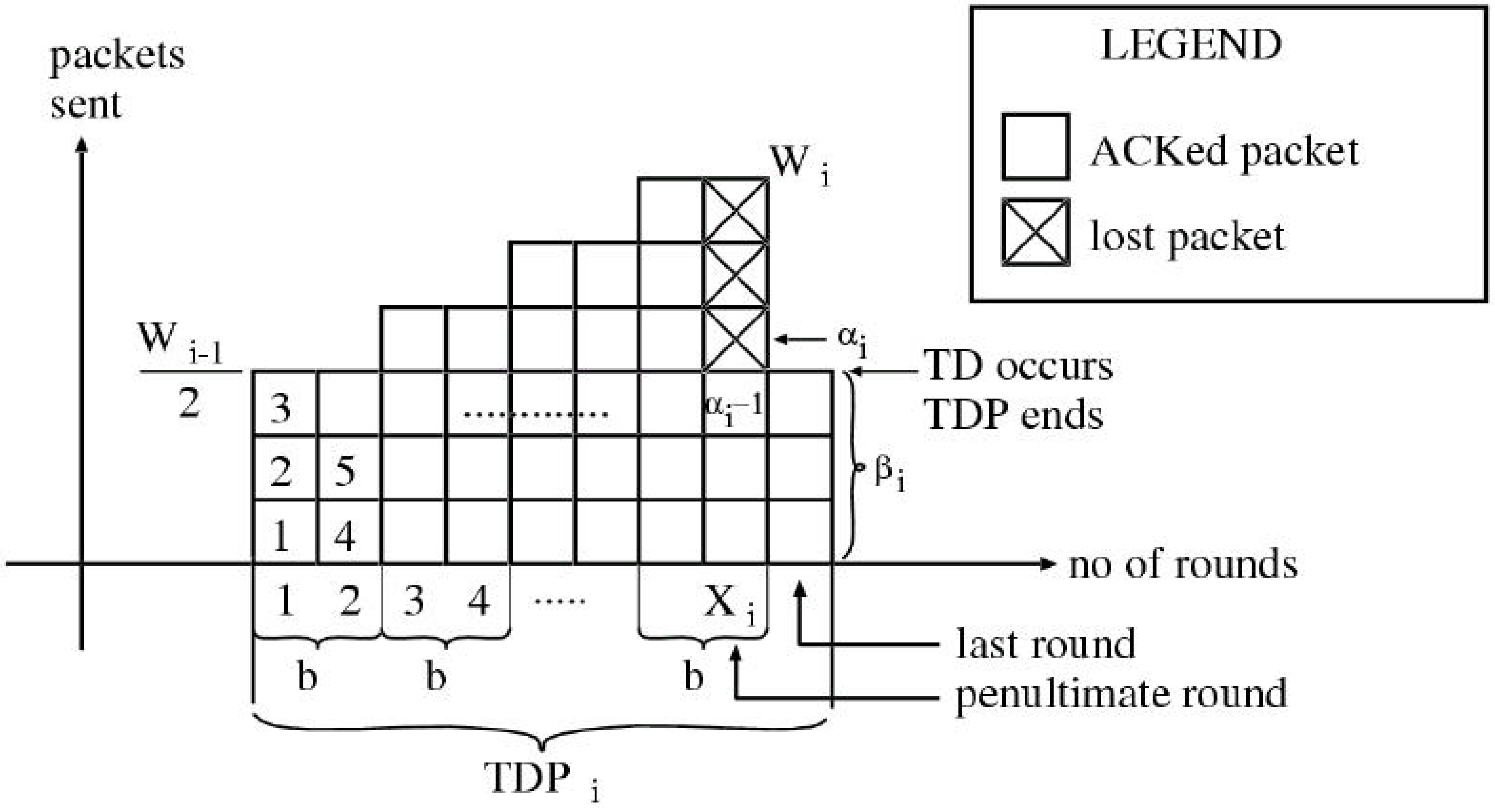
- Single “saturated” TCP sender pumping into a loaded bottleneck – the other flows are modeled only through bottleneck packet loss p
- TCP behavior modelled as a sequence of “rounds”
- The round begins when the sender sends out W pkts back-to-back (this takes < RTT)
- Round ends when receiver gets first ACK
- Packet loss p independent from round to round
- **First model**: the renewal interval is terminated by a Triple Dup ACK (TDP)

Model similar to the Markov model used for TCP Westwood – but, here, closed form



TDP = Markov renewal interval terminated by Triple Dup ACK; made up of several RTTs

Detail view of TDP model

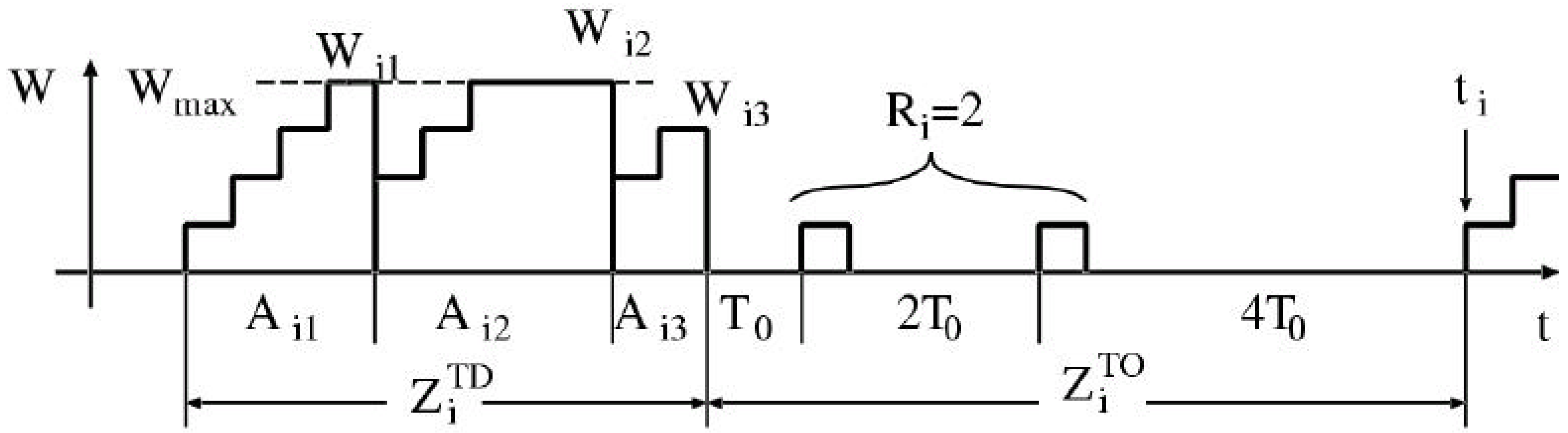


b = # of packets acked by a single ACK (typically b =2; see details on Padhye's paper

TDP model

$$B(p) = \frac{1}{RTT}\sqrt{\frac{3}{2bp}} + o(1/\sqrt{p})$$

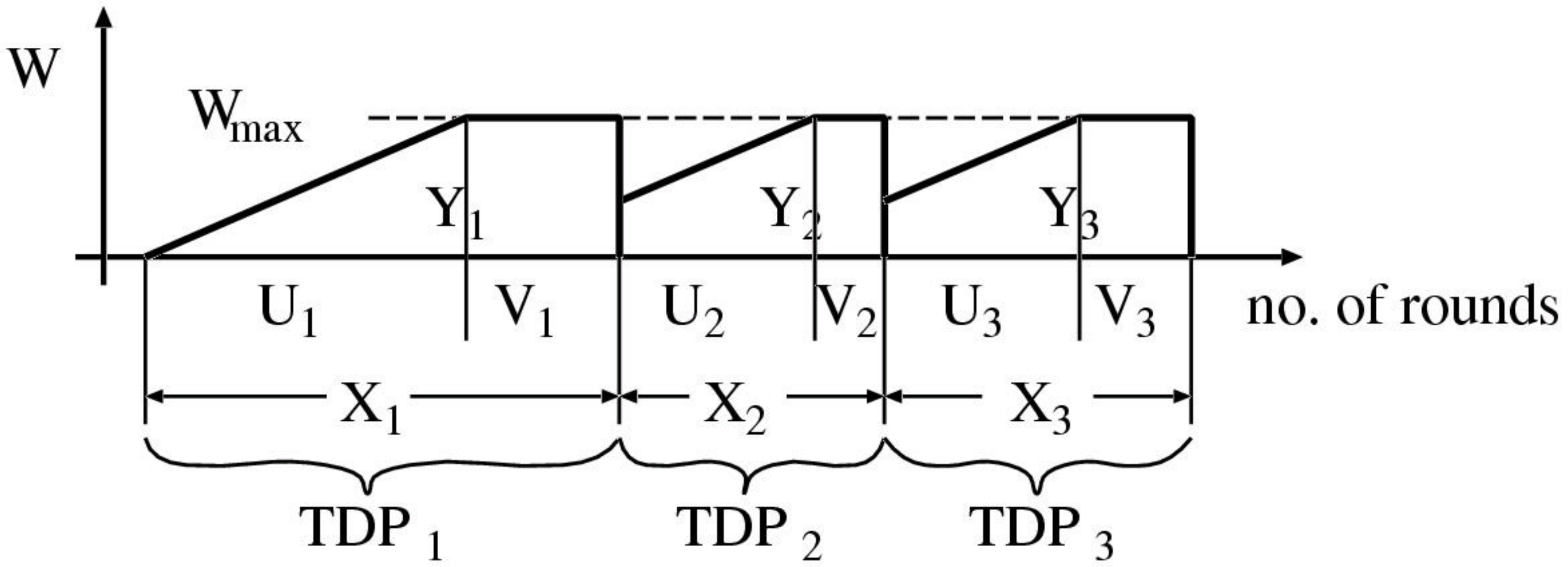
Next, include T_{rto} in model



Now, the renewal interval is more complicated..

$$B(p) \approx \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}$$

Finally, the advertised Window



$$B(p) \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right)$$

Measurements and Trace Analysis

- Empirical validation from 37 TCP connections between 18 hosts in the US and Europe
- Measurement data gathered with TCP-Dump at sender; analyzed with UMASS tools
- From results, the importance of timeouts is obvious

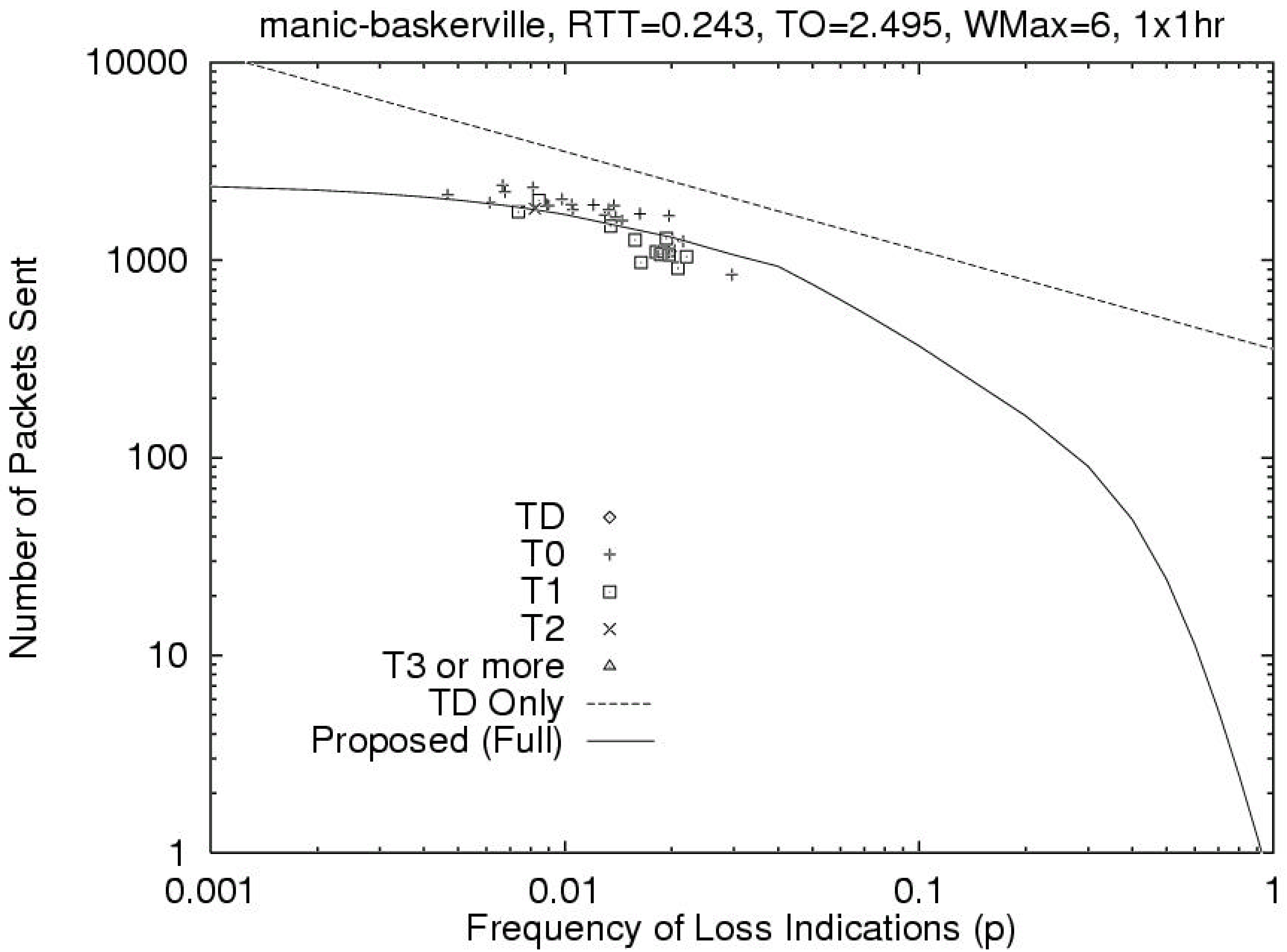
Receiver	Domain	Operating System
ada	hofstra.edu	Irix 6.2
afer	cs.umn.edu	Linux
al	cs.wm.edu	Linux 2.0.31
alps	cc.gatech.edu	SunOS 4.1.3
babel	cs.umass.edu	SunOS 5.5.1
baskerville	cs.arizona.edu	SunOS 5.5.1
ganef	cs.ucla.edu	SunOS 5.5.1
imagine	cs.umass.edu	win95
manic	cs.umass.edu	Irix 6.2
mafalda	inria.fr	SunOS 5.5.1
maria	wustl.edu	SunOS 4.1.3
modi4	ncsa.uiuc.edu	Irix 6.2
pif	inria.fr	Solaris 2.5
pong	usc.edu	HP-UX
spiff	sics.se	SunOS 4.1.4
sutton	cs.columbia.edu	SunOS 5.5.1
tove	cs.umd.edu	SunOS 4.1.3
void	US site	Linux 2.0.30

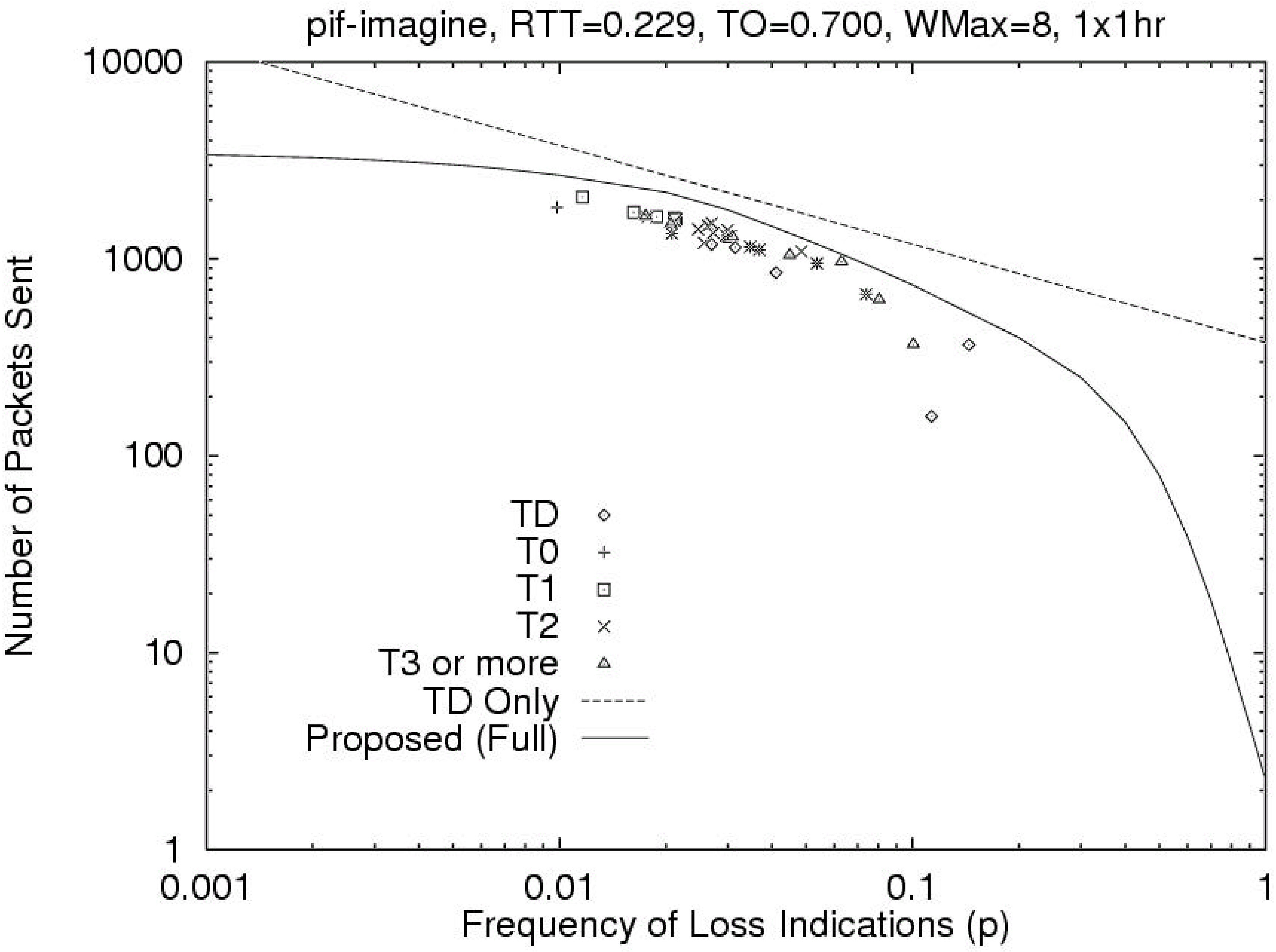
Sender	Receiver	Packets Sent	Loss Indic.	TD	TO	RTT	Time Out
manic	alps	54402	722	19	703	0.207	2.505
manic	baskerville	58120	735	306	429	0.243	2.495
manic	ganef	58924	743	272	471	0.226	2.405
manic	mafalda	56283	494	2	492	0.233	2.146
manic	maria	68752	649	1	648	0.180	2.416
manic	spiff	117992	784	47	737	0.211	2.274
manic	sutton	81123	1638	988	650	0.204	2.459
manic	tove	7938	264	1	263	0.275	3.597
void	alps	37137	838	7	831	0.162	0.489
void	baskerville	32042	853	339	514	0.482	1.094
void	ganef	60770	1112	414	696	0.254	0.637
void	maria	93005	1651	33	1618	0.152	0.417
void	spiff	65536	671	72	599	0.415	0.749
void	sutton	78246	1928	840	1088	0.211	0.601
void	tove	8265	856	5	843	0.272	1.356
babel	alps	13460	1466	0	1461	0.194	1.359
babel	baskerville	62237	1753	197	1556	0.253	0.429
babel	ganef	86675	2125	398	1727	0.201	0.306
babel	spiff	57687	1120	0	1120	0.331	0.953
babel	sutton	83486	2320	685	1635	0.210	0.705
babel	tove	83944	1516	1	1514	0.194	0.520
pif	alps	83971	762	0	760	0.168	7.278
pif	imagine	44891	1346	15	1329	0.229	0.700
pif	manic	34251	1422	43	1377	0.257	1.454

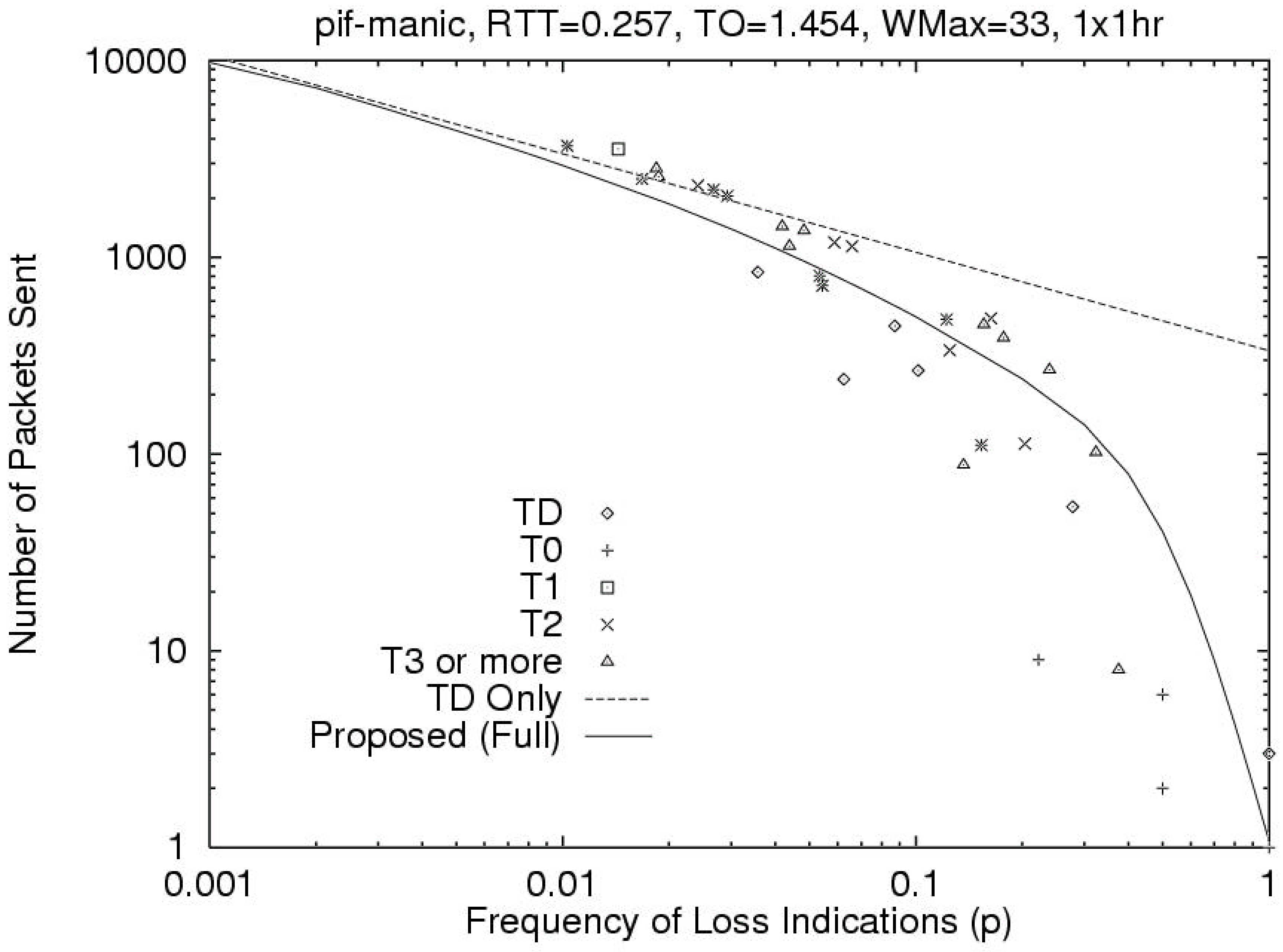
Validation Experiments based on 1hr traces.
Hourly traces were subdivided in 36 X 100s segments;
each segment maps into a point on the T vs p graph

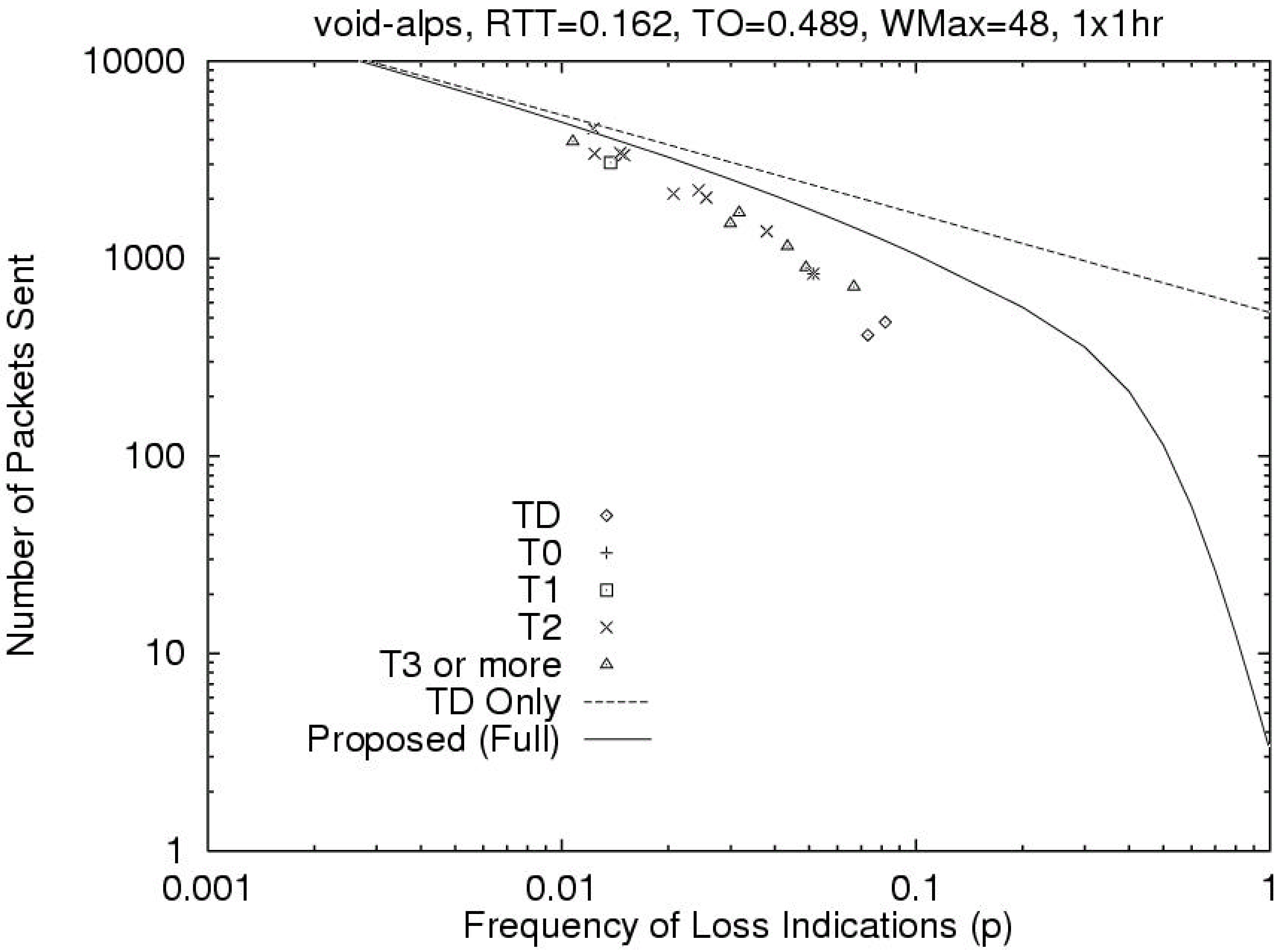
Sender	Receiver	Packets Sent	Loss Indic.	TD	TO	RTT	Time Out
manic	ada	531533	6432	4320	2112	0.141	2.223
manic	afer	255674	4577	2584	1993	0.180	2.3
manic	al	264002	4720	2841	1879	0.188	2.354
manic	alps	667296	3797	841	2956	0.112	1.915
manic	baskerville	89244	1638	627	1011	0.473	3.226
manic	ganef	160152	2470	1048	1422	0.215	2.607
manic	mafalda	171308	1332	9	1323	0.250	2.512
manic	maria	316498	2476	5	2471	0.116	1.879
manic	modi4	282547	6072	3976	2096	0.174	2.26
manic	pong	358535	4239	2328	1911	0.176	2.137
manic	spiff	298465	2035	159	1876	0.253	2.454
manic	sutton	348926	6024	3694	2330	0.168	2.185
manic	tove	262365	2603	6	2597	0.115	1.955

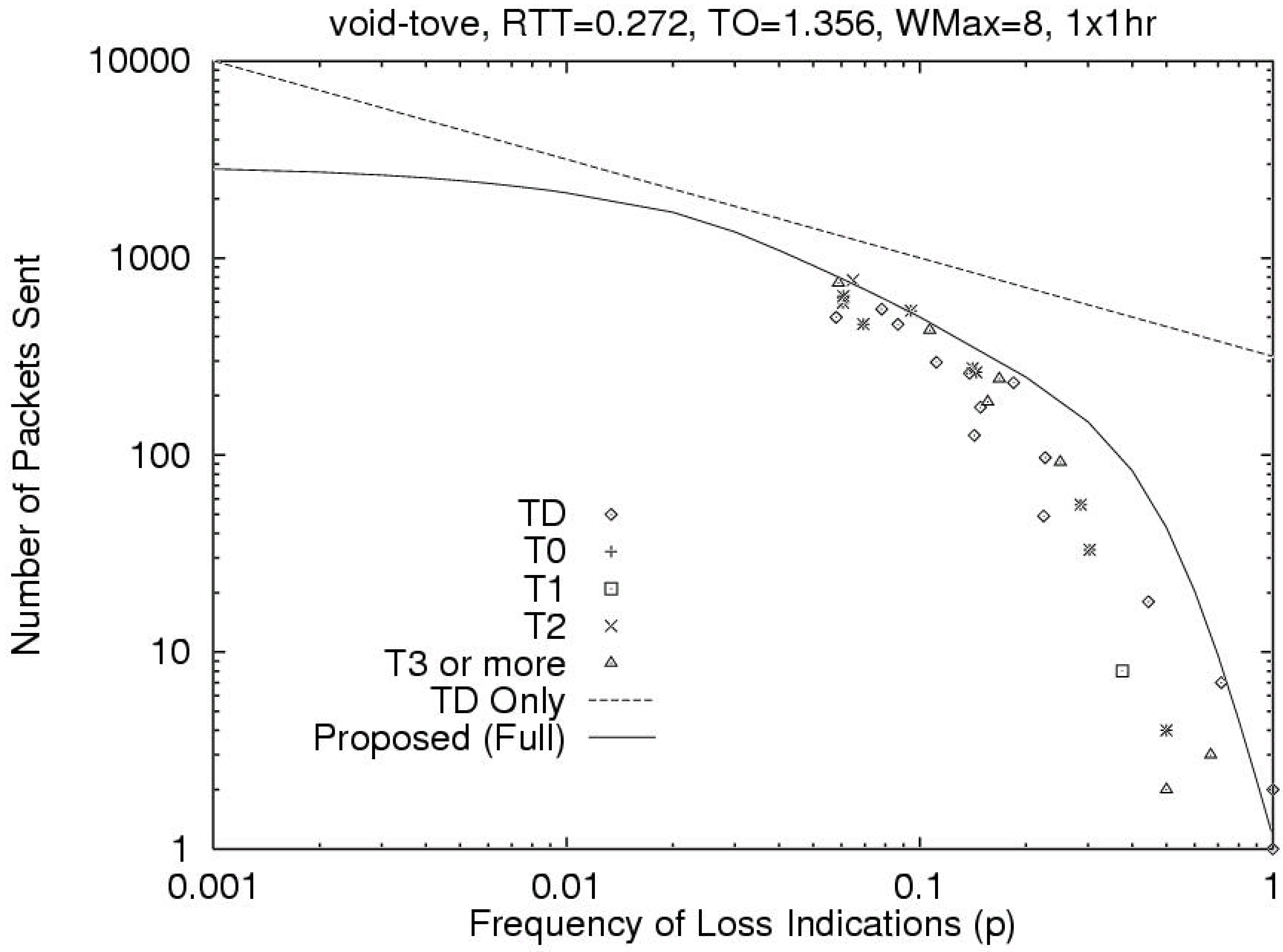
Summary data for the 100s traces

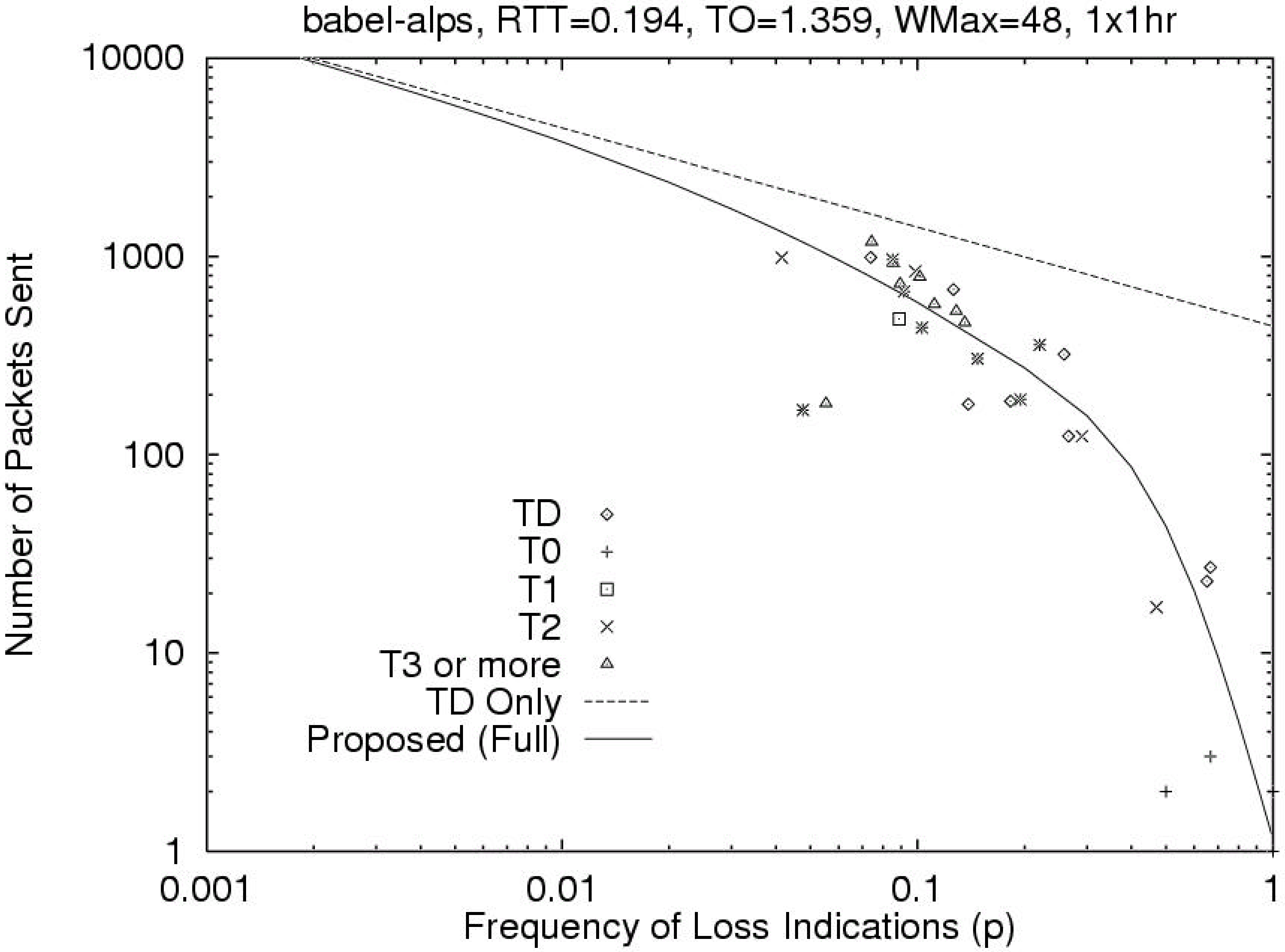

manic-baskerville, RTT=0.243, TO=2.495, WMax=6, 1x1hr
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD ◇
T0 +
T1 ⊡
T2 ×
T3 or more △
TD Only
Proposed (Full)

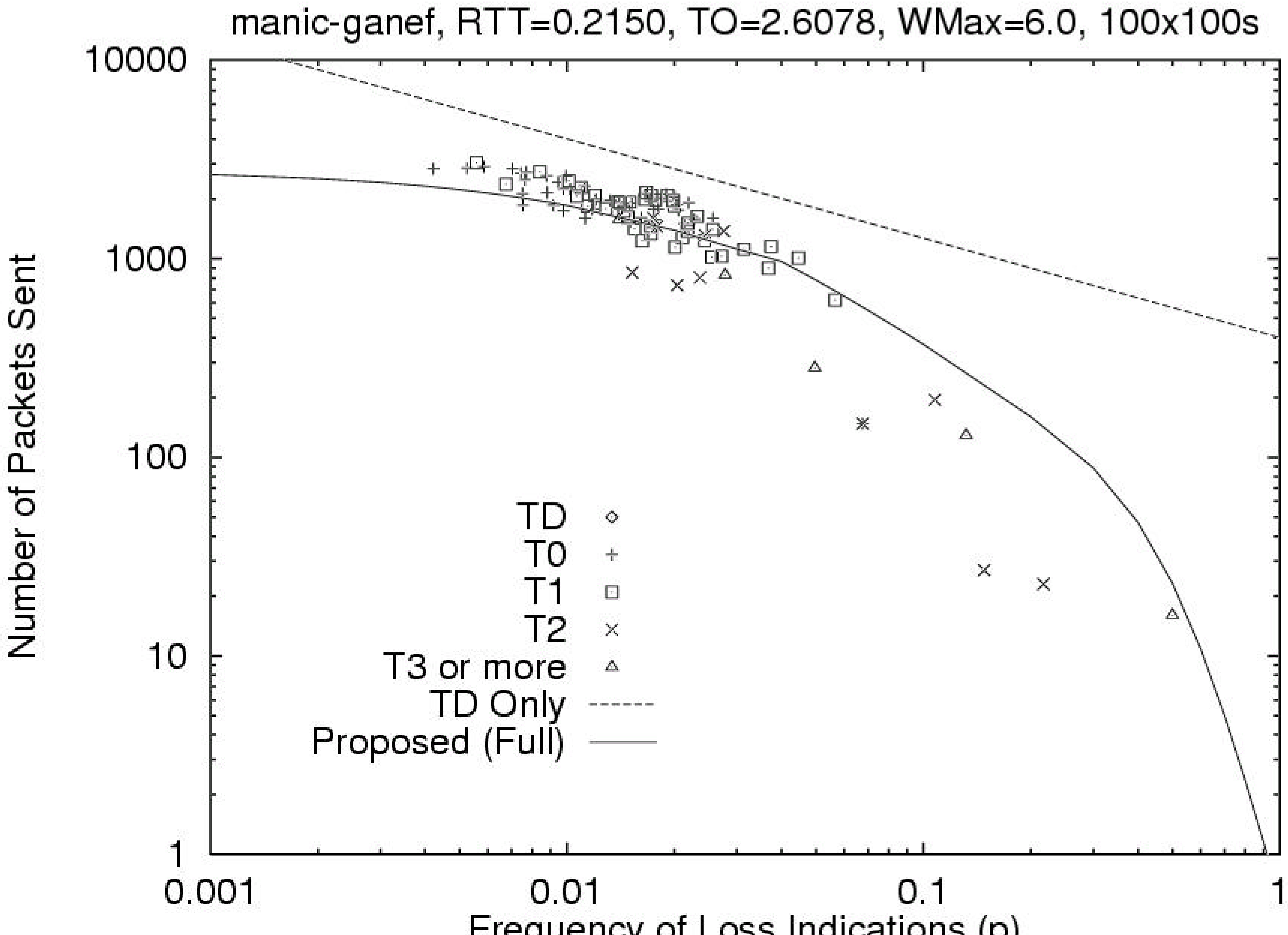

pif-imagine, RTT=0.229, TO=0.700, WMax=8, 1x1hr
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD
T0
T1
T2
T3 or more
TD Only
Proposed (Full)


pif-manic, RTT=0.257, TO=1.454, WMax=33, 1x1hr
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD
T0
T1
T2
T3 or more
TD Only
Proposed (Full)

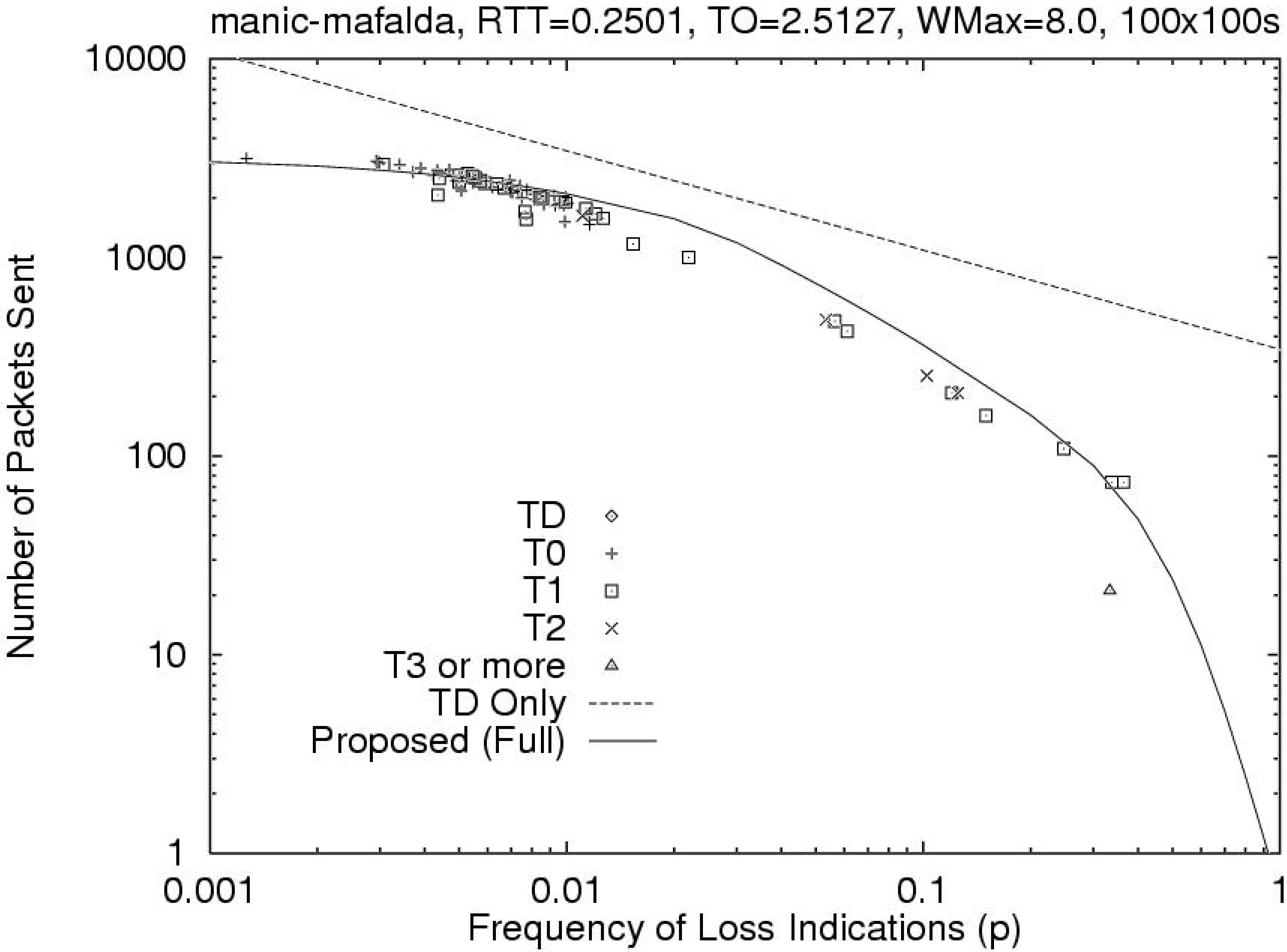

void-alps, RTT=0.162, TO=0.489, WMax=48, 1x1hr
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD
T0
T1
T2
T3 or more
TD Only
Proposed (Full)

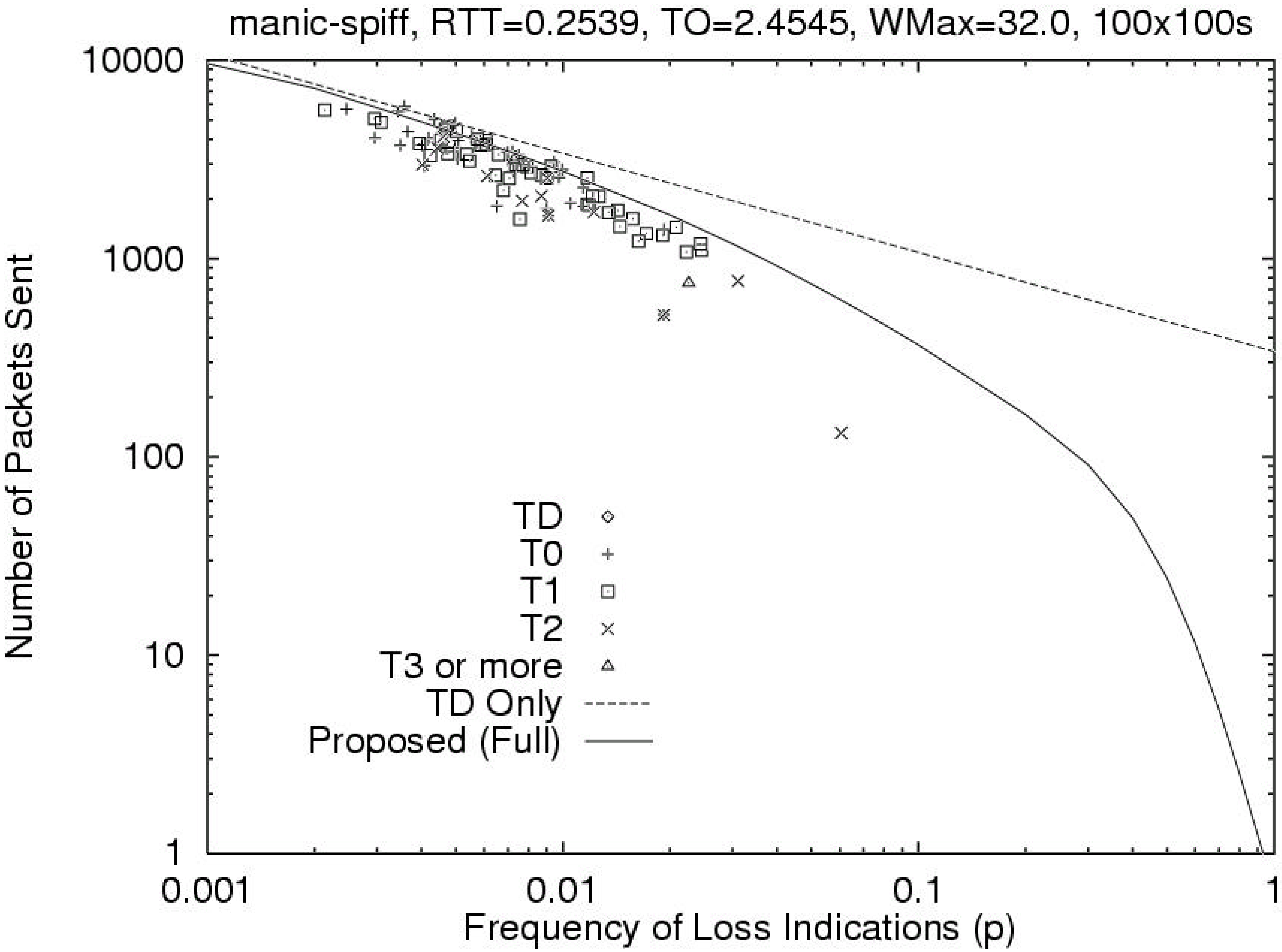

void-tove, RTT=0.272, TO=1.356, WMax=8, 1x1hr
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD
T0
T1
T2
T3 or more
TD Only
Proposed (Full)

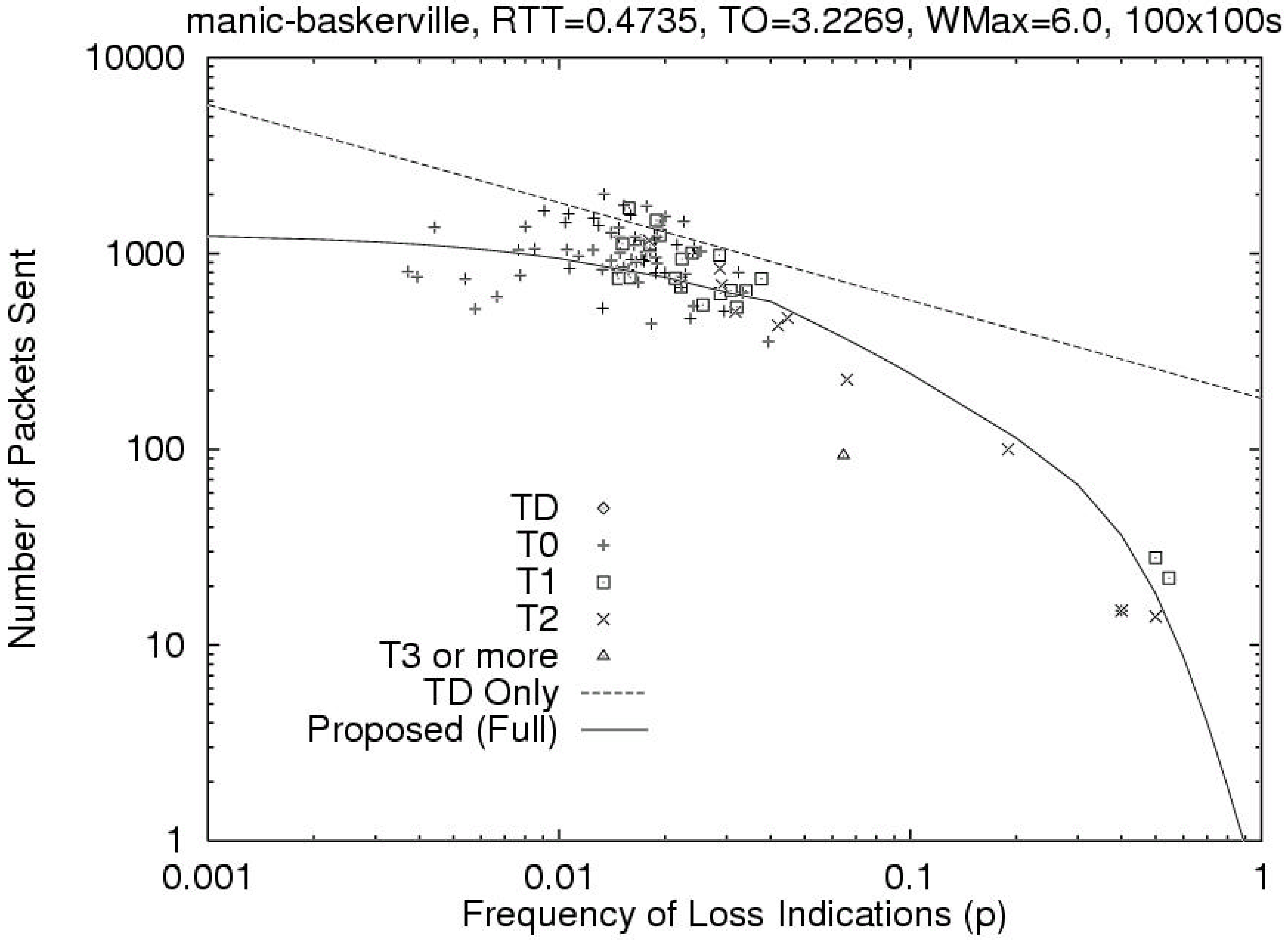

babel-alps, RTT=0.194, TO=1.359, WMax=48, 1x1hr
Number of Packets Sent
10000
1000
100
10
1
0.001
0.01
0.1
1
Frequency of Loss Indications (p)
TD
T0
T1
T2
T3 or more
TD Only
Proposed (Full)

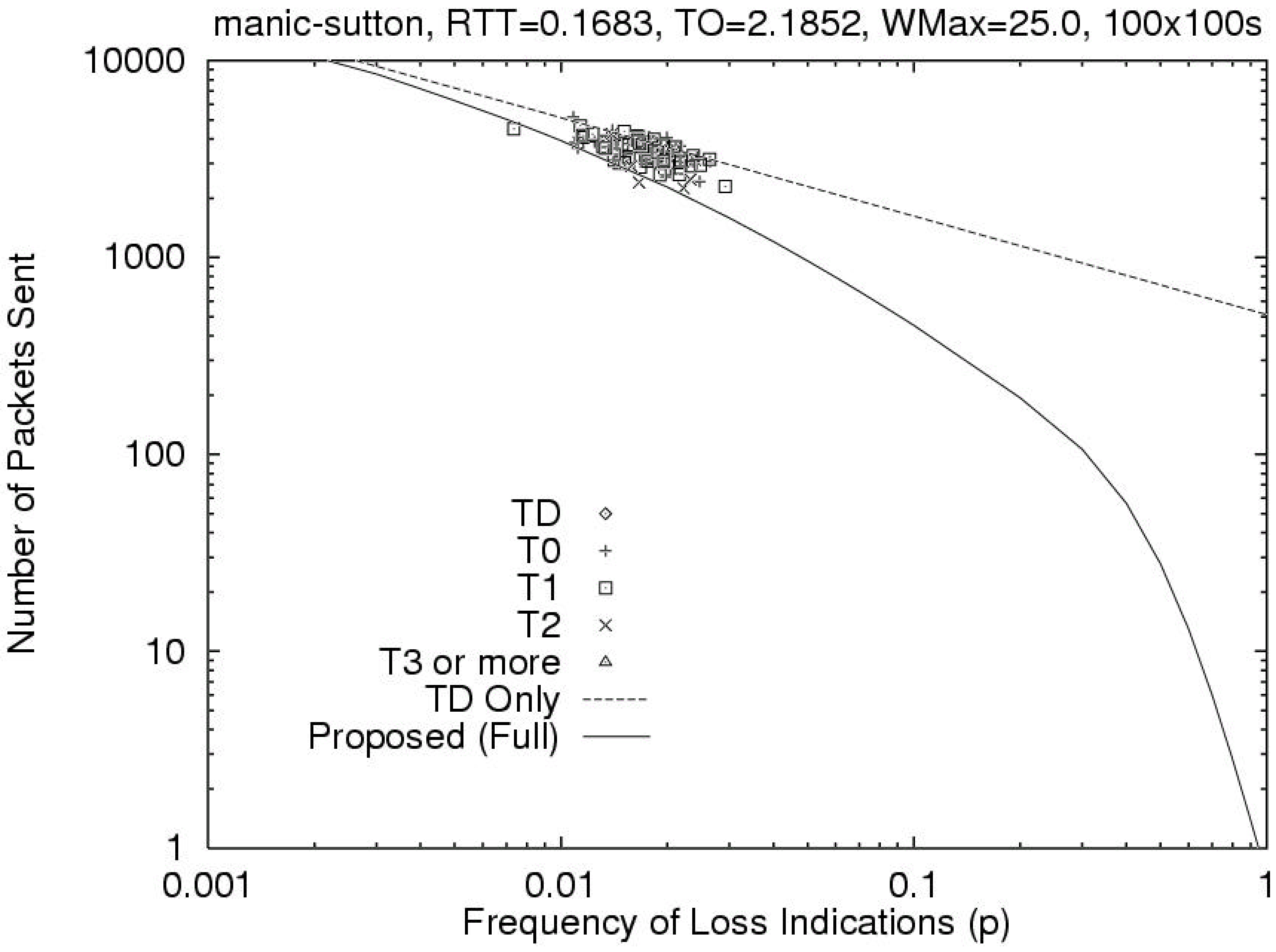


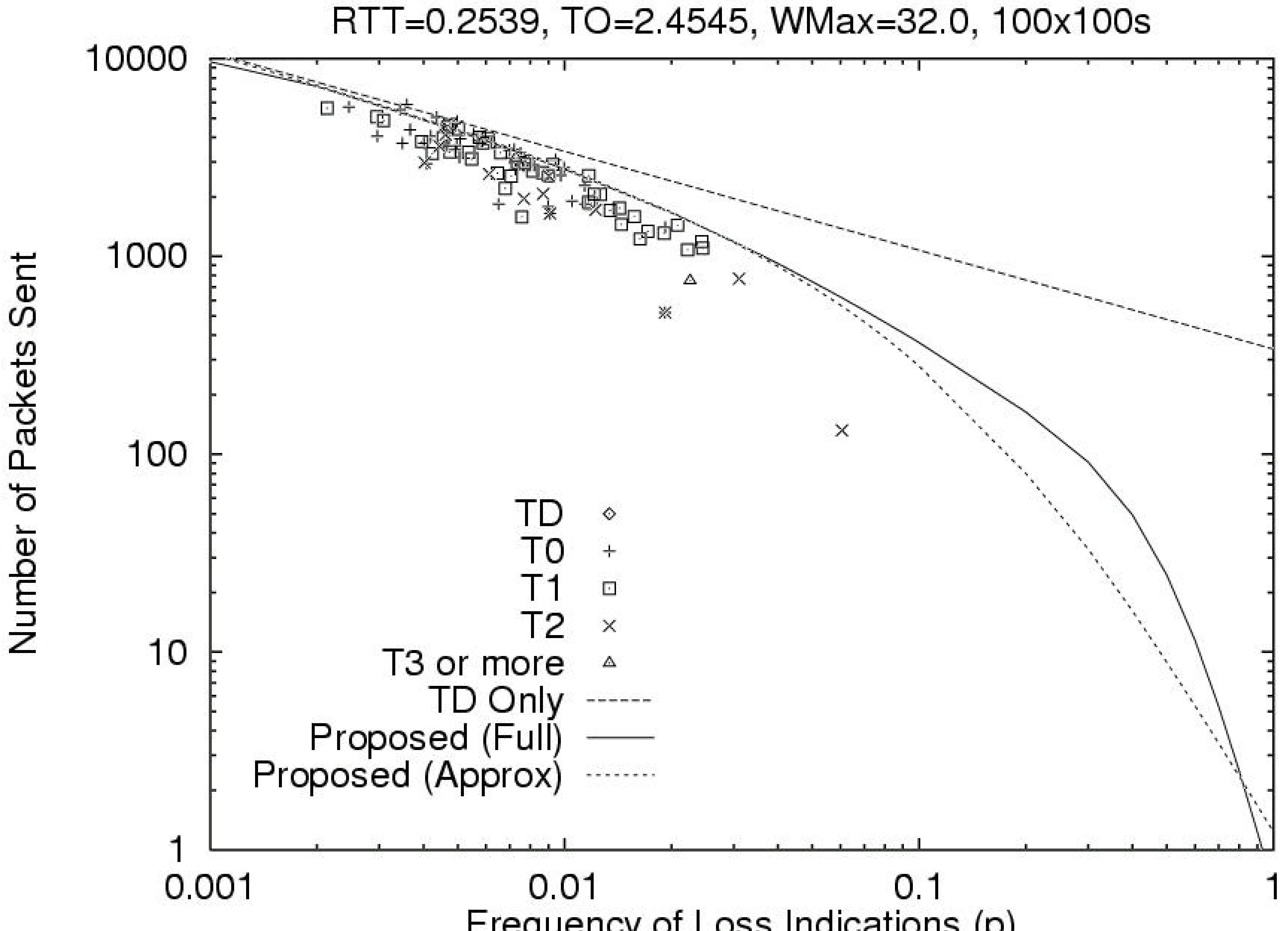
manic-ganef, RTT=0.2150, TO=2.6078, WMax=6.0, 100x100s
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD
T0
T1
T2
T3 or more
TD Only
Proposed (Full)


manic-mafalda, RTT=0.2501, TO=2.5127, WMax=8.0, 100x100s
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD ◇
T0 +
T1 ⊡
T2 ×
T3 or more △
TD Only
Proposed (Full)


manic-spiff, RTT=0.2539, TO=2.4545, WMax=32.0, 100x100s
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD ◇
T0 +
T1 ⊡
T2 ×
T3 or more △
TD Only
Proposed (Full)


manic-baskerville, RTT=0.4735, TO=3.2269, WMax=6.0, 100x100s
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD ◇
T0 +
T1 ⊡
T2 ×
T3 or more △
TD Only
Proposed (Full)


manic-sutton, RTT=0.1683, TO=2.1852, WMax=25.0, 100x100s
Number of Packets Sent
10000
1000
100
10
1
Frequency of Loss Indications (p)
0.001
0.01
0.1
1
TD ◇
T0 +
T1 ⊡
T2 ×
T3 or more △
TD Only
Proposed (Full)

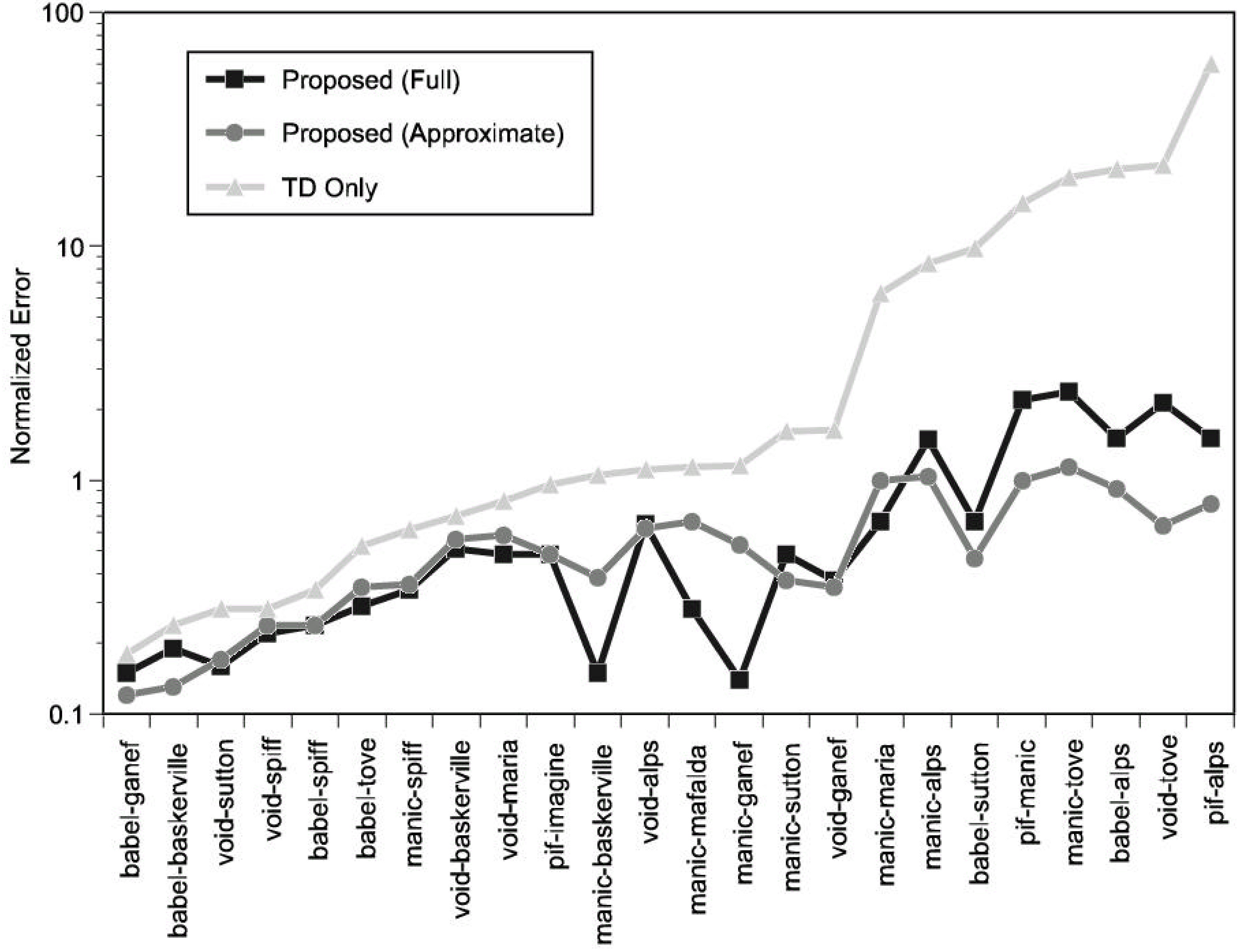

RTT=0.2539, TO=2.4545, WMax=32.0, 100x100s
Number of Packets Sent
Frequency of Loss Indications (p)
10000
1000
100
10
1
0.001
0.01
0.1
1
TD ◇
T0 +
T1 ⊡
T2 ×
T3 or more △
TD Only
Proposed (Full)
Proposed (Approx)

Full model:

$$B(p) = \begin{cases} \dfrac{\frac{1-p}{p} + E[W] + \hat{Q}(E[W])\frac{1}{1-p}}{RTT(\frac{b}{2}E[W_u]+1) + \hat{Q}(E[W])T_0\frac{f(p)}{1-p}} & \text{if } E[W_u] < W_{max} \\ \dfrac{\frac{1-p}{p} + W_{max} + \hat{Q}(W_{max})\frac{1}{1-p}}{RTT(\frac{b}{8}W_{max} + \frac{1-p}{pW_{max}} + 2) + \hat{Q}(W_{max})T_0\frac{f(p)}{1-p}} & \text{otherwise} \end{cases}$$

Approximate model:

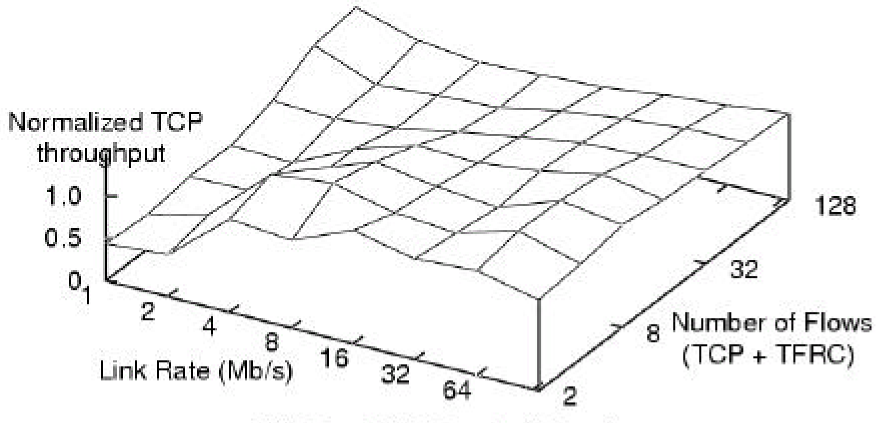
$$B(p) \approx \min\left(\frac{W_{max}}{RTT}, \frac{1}{RTT\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1+32p^2)}\right)$$


Proposed (Full)
Proposed (Approximate)
TD Only
Normalized Error
100
10
1
0.1
babel-ganef
babel-baskerville
void-sutton
void-spiff
babel-spiff
babel-tove
manic-spiff
void-baskerville
void-maria
pif-imagine
manic-baskerville
void-alps
manic-mafalda
manic-ganef
manic-sutton
void-ganef
manic-maria
manic-alps
babel-sutton
pif-manic
manic-tove
babel-alps
void-tove
pif-alps

Back to TFRC

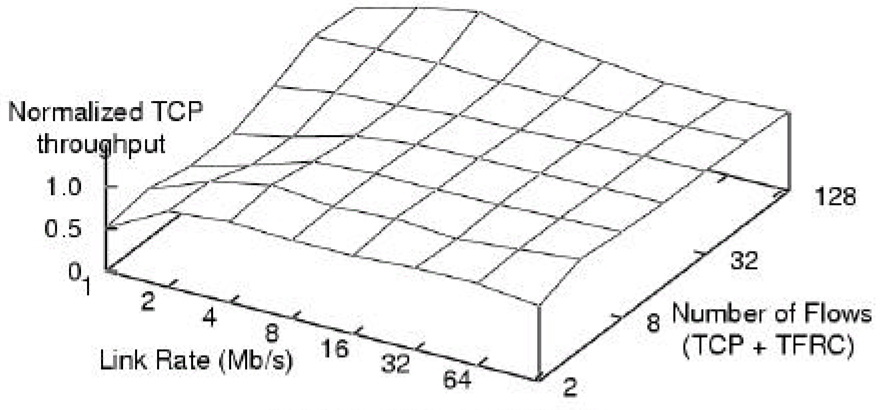
- **Sender**: measures various parameters; calculates the TCP-like rate corresponding to the measured parameters
- **Receiver**: provides feedback to sender to allow it to calculate RTT; also calculates loss event rate p
- The p rate computation critical for performance of TFRC.
- **Average Loss Interval**: weighted average of loss rate over the last N loss intervals (loss interval = interval of packets between loss episodes)

NS Simulation results: TCP SACK +TFRC fair sharing
Normalized TCP Thr =1 means perfect fairness



TFRC vs TCP, DropTail Queuing

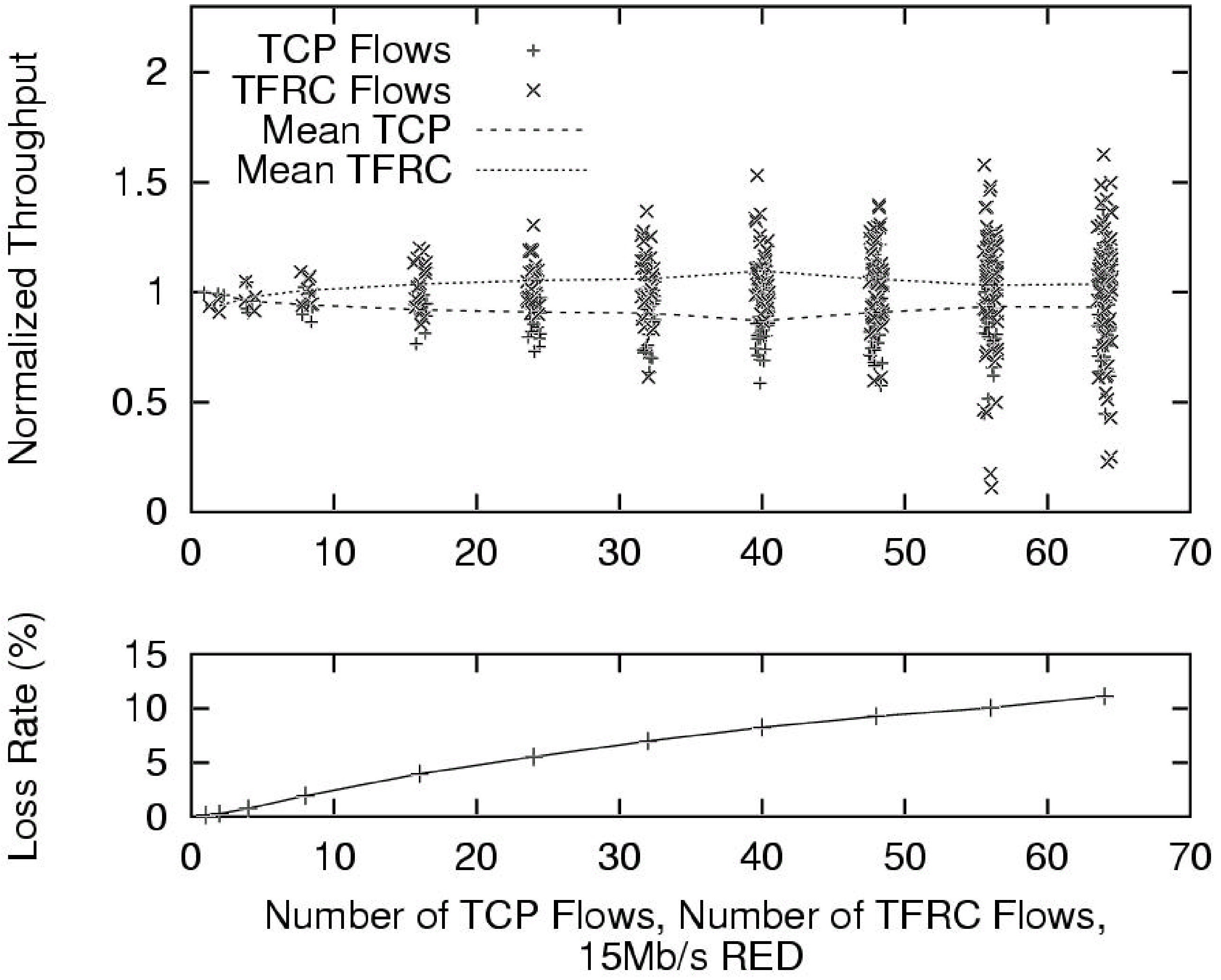
N TCP flows +
N TFRC flows

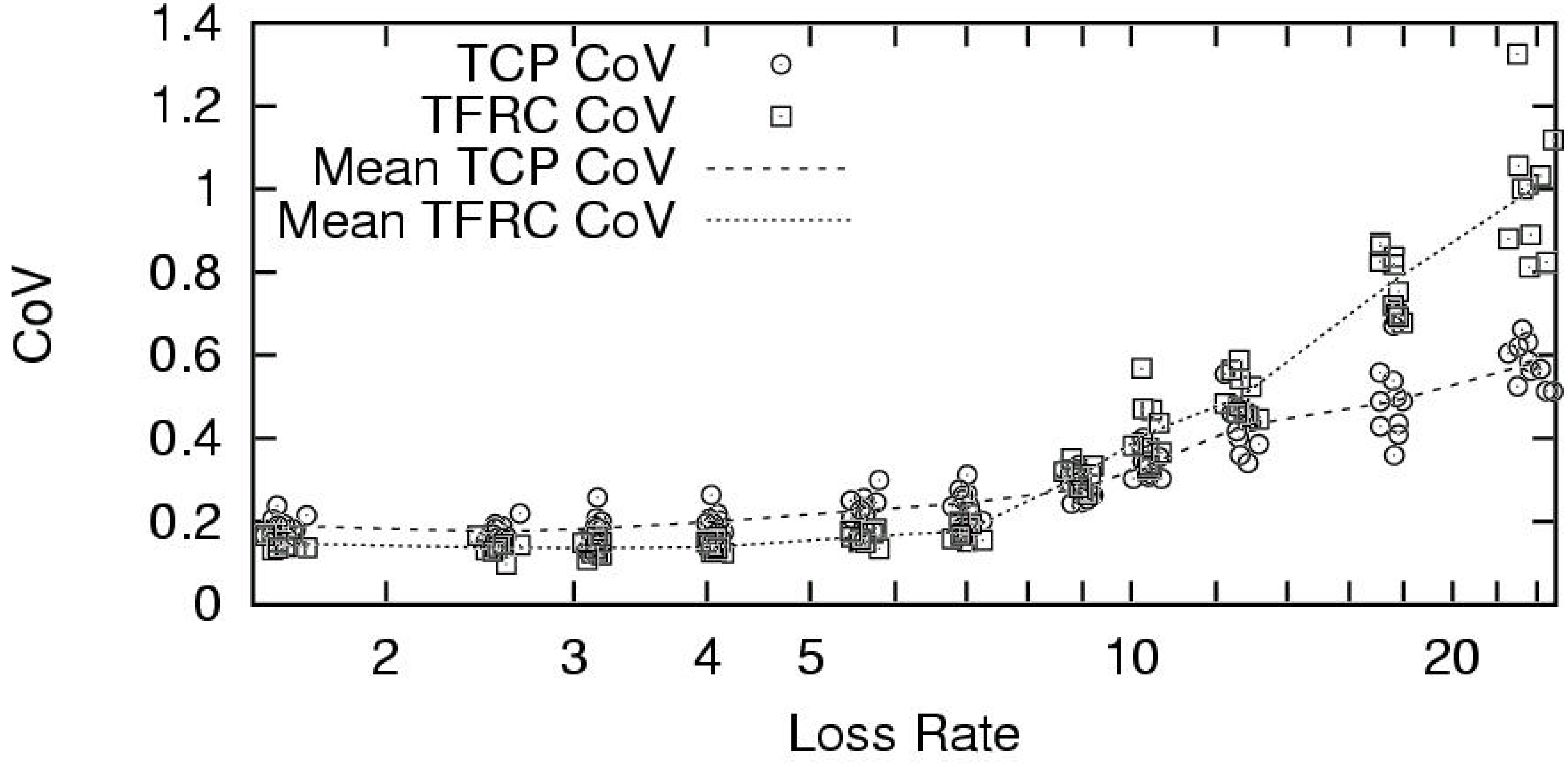


TFRC vs TCP, RED Queuing

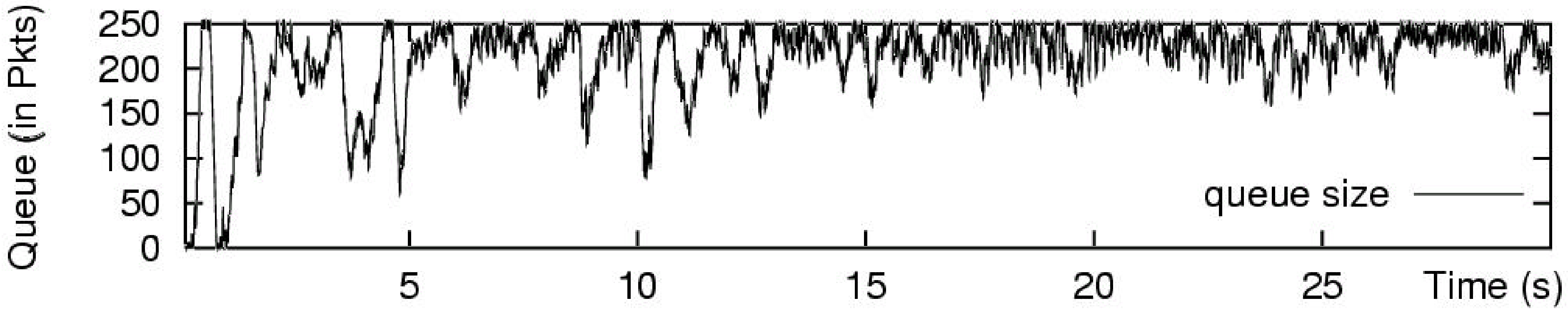
TFRC less aggressive than TCP

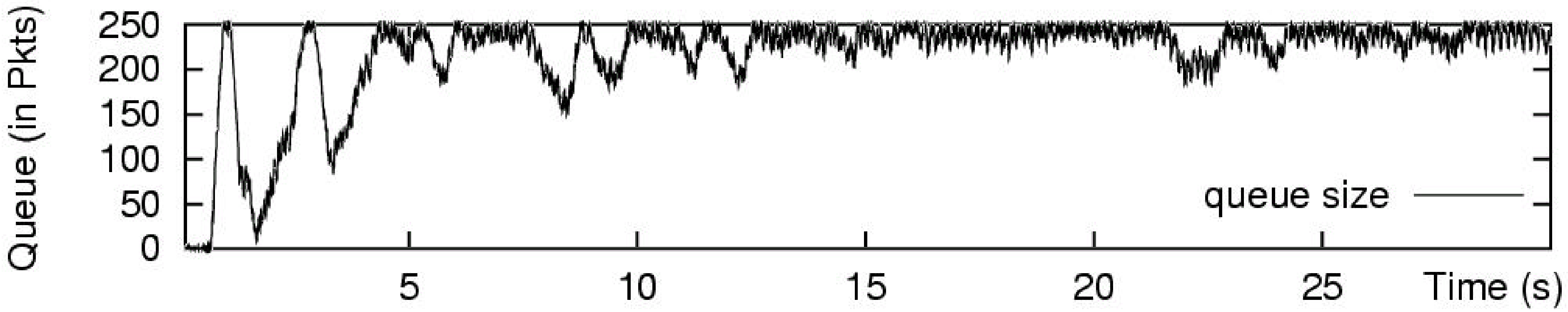
TFRC internally unevenly "fair"




TCP CoV
TFRC CoV
Mean TCP CoV
Mean TFRC CoV
CoV
0
0.2
0.4
0.6
0.8
1
1.2
1.4
2
3
4
5
10
20
Loss Rate

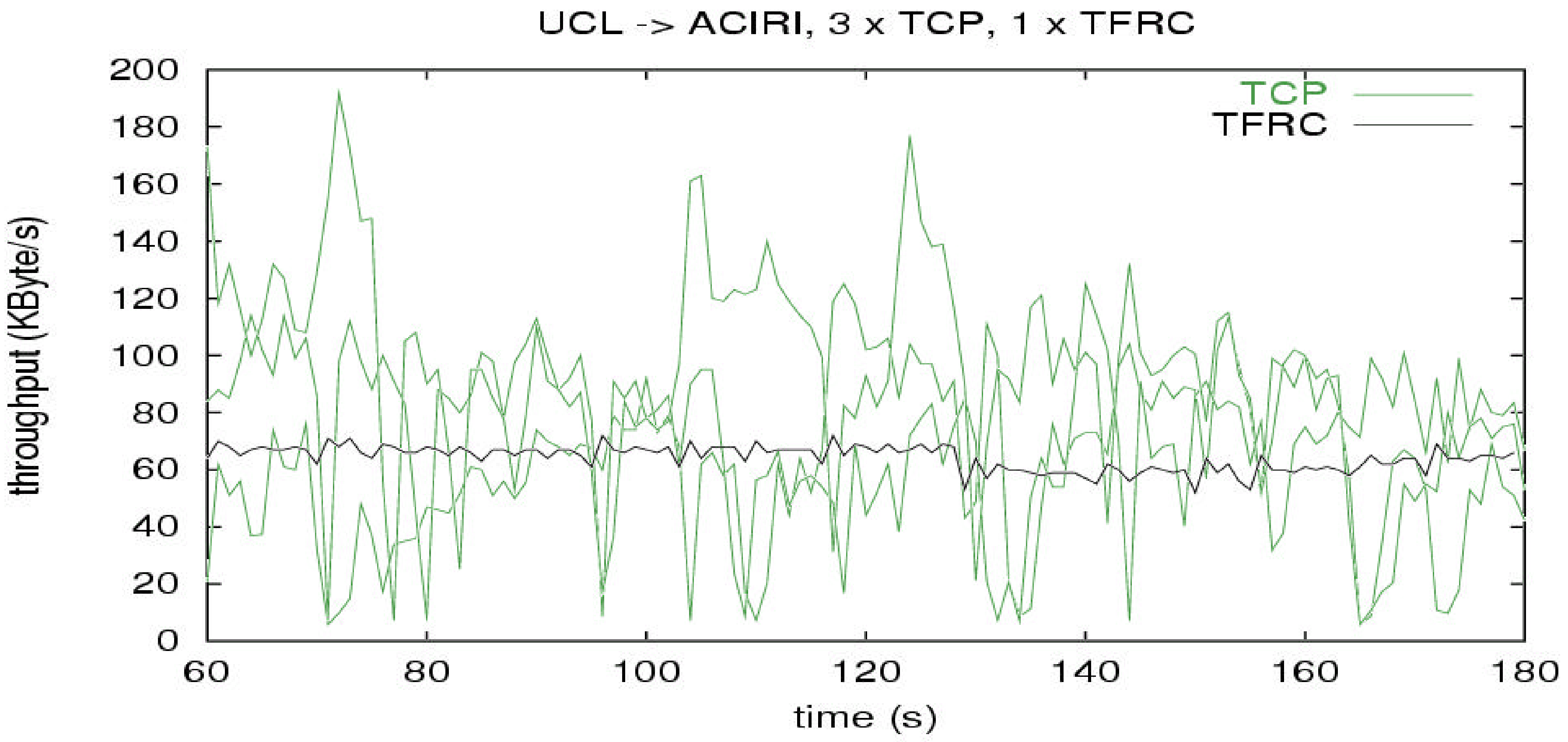
40 “long lived” flows **simulation**: the 40 flows start in the first 20 s. We show bottleneck queue dynamics





Comment: TFRC (bottom) is as stable as TCP (top).
TCP drop rate =4.9%; TFRC drop rate = 3.5%

Internet Measurements: 3 TCP connections – London to Berkeley. Throughput measured over 1 sec intervals



TFRC much more stable than TCP

Conclusions

- TFRC valuable for best effort unicast streaming
- Simulation and Implementation code available for testing
- Multicast extension very attractive
- Need to include ECN in eq. model
- What about random link loss?