TCP MaxNet – Implementation and Experiments on the WAN in Lab

Martin Suchara, Ryan Witt, Bartek Wydrowski

California Institute of Technology
Pasadena, CA 91125, U.S.A.
{suchara, witt, bartek} @caltech.edu

***Abstract*— We describe implementation and performance of TCP MaxNet, a new protocol which uses a multi-bit explicit signaling approach to congestion control. The MaxNet sender algorithm operates by adjusting its congestion window in response to explicit feedback from the most congested link encountered in the network. This scheme has numerous theoretical advantages over the ubiquitous practice of adjusting the congestion window based on the total amount of congestion in the path. We implement the MaxNet control scheme on top of the existing Linux TCP/IP protocol framework and evaluate its performance in the high bandwidth-delay environment of the WAN in Lab. Our experiments show that MaxNet possesses the desirable properties that theory predicts: very short router queues and fair sharing among multiple flows of different RTTs.**

I. Introduction

Congestion avoidance has been a central topic in computer network design over the past two decades. The most widely used congestion control algorithm today, TCP Reno, employs an Additive Increase Multiplicative Decrease (AIMD) scheme that increases its congestion window by one whenever a packet is successfully transmitted, and cuts the window in half whenever a packet is lost. While TCP Reno achieves excellent performance in low-speed, low-loss networks, it scales poorly as the bandwidth-delay products of networks increase. One reason for this is that packet sizes have stayed largely the same while hardware transmissions speeds have grown. The packet-wise Additive Increase means that it takes an increasingly long time to reach maximum throughput. When coupled with Multiplicative Decrease, which cuts the transmission rate in half when congestion loss is encountered, the slow ramp-up behavior severely limits full utilization of bandwidth. In general, using loss as a congestion signal is problematic. Though packet loss is a typical result of overflowing buffers on congested links, it cannot be used to prevent congestion before it occurs because the signal reaches the sender too late. Moreover, wireless links have inherent losses which introduce noise into the congestion signal and make AIMD even more costly. Consequently, the careful design of a new TCP protocol that gracefully scales with the capabilities of computer networks is very important. We provide the first implementation and experimental evaluation of the MaxNet congestion control scheme [1] [2] [3] which we call TCP MaxNet, adding it to a growing number of efforts such as FAST TCP [4] [5] and XCP [6] [7] that attempt to achieve this goal.

A number of intriguing theoretical properties of the MaxNet congestion control scheme have motivated our investigation: Scalability to networks with arbitrary bandwidth-delay products [1], max-min fair sharing of bandwidth between flows [2], short router queues and quick convergence after network state changes [3]. Like FAST TCP and XCP, MaxNet disassociates itself from the AIMD scheme that plagues the current generation of TCPs, and like XCP, MaxNet relies on explicit congestion signals from the routers to tell end hosts how to adjust their transmission rates. Explicit signaling schemes are vastly superior to AIMD in several ways [6] since they allow a sender to react to congestion well before any negative symptoms such as packet loss or prolonged RTT appear. Because of this early notification and reaction, an explicit signaling protocol is able to keep the average router queue length close to zero, which is something FAST is not able to do, since it relies on change in delay (and thus positive queues) to detect congestion. Additionally, the idea of explicit congestion signaling could be extended to allow the sender to nearly immediately (after one RTT) begin sending at a rate close to the spare capacity in the network.

TCP MaxNet differs significantly from other explicit signaling protocols such as ECN marking and XCP. Both XCP and TCP MaxNet have many more bits in their feedback signal than ECN, and Wydrowski, et. al. show in [3] that MaxNet has faster convergence properties than ECN marking. MaxNet also appears to have several advantages over XCP. The simplicity of the MaxNet control scheme lends itself to an easier implementation. Whereas XCP requires a fair amount of computation at the routers to operate properly, including code to deal with fractional congestion feedback [7], a MaxNet router requires few instructions which reduces delay and allows implementation on a wider variety of network devices. In addition, our implementation of MaxNet requires less control data and thus less network overhead than XCP.

The aim of this paper is to report on our successful implementation of TCP MaxNet and provide experimental measurements confirming the conclusions provided by the theoretical analyses. In order to effectively evaluate TCP MaxNet's performance versus classic TCPs, we need a network environment capable of providing the high bandwidth-delay product links on which differences between MaxNet and

AIMD schemes appear. Such an environment was provided by the newly constructed WAN in Lab [8], a standalone network laboratory environment comprising an array of routers, servers and long-distance high-speed optical links. In Section II, we present a high level overview the MaxNet congestion control scheme and describe how the various parts of the protocol interact. In Section III, we describe our implementation of MaxNet on top of the Linux TCP/IP protocol stack and discuss the challenges we faced working within the limitations of the Linux kernel. In Section IV, we describe the performance measurements collected from our experiments on the WAN in Lab and provide directions for further research. Section V summarizes our results.

II. MaxNet Architecture

Congestion control in MaxNet comprises two principal components: an Active Queue Management (AQM) algorithm which resides at each router, and a source algorithm which controls transmission rates and resides at the sender and receiver. This section describes both algorithms and explains how they interact.

A. AQM Algorithm

Routers use the AQM algorithm to calculate congestion level for each of their outbound links. The congestion level of a particular link is indicated by the integrator function

$$Price(t+1) = Price(t) + \frac{1}{C}(X(t) - \mu C), \qquad (1)$$

where C is the output capacity of the link in bps, μ is the percentage utilization of this link when we consider it "full" and decide to back off from increasing our transmission rate, and $X(t)$ denotes the speed at which packets are being enqueued at this link in bps. $Price(t)$ is a very good indicator of congestion on the link at time t. A closer inspection of (1) reveals that $Price(t)$ corresponds to this link's expected queue length *in seconds* at time t. $Price(0)$ is defined to be zero since the queue is initially empty. The value calculated in this equation is later fed back to the sender and if it is the maximum of all such values on the path, it is used to calculate the sender's congestion window. Conveniently, this router algorithm is fully distributed across the routers and no per flow state is needed.

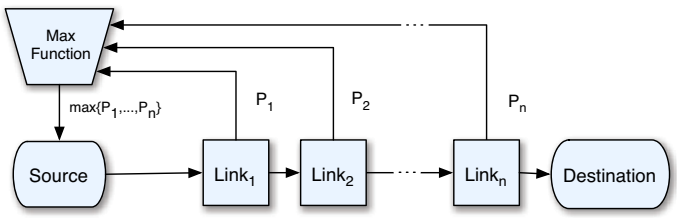

Fig. 1. Conceptual congestion control scheme of MaxNet. The level of congestion at each link, $P_1, P_2, \cdots, P_n$, is fed back to the sender, which then uses the max of the values to adjust its congestion window size.

1) Estimate the target window size:

$$\eta \leftarrow \eta + \frac{\texttt{p1}}{\texttt{cwnd}} - \frac{\texttt{p2}\times\texttt{price}}{\texttt{baseRTT}} \times \texttt{interval}$$

$$\texttt{estWnd} \leftarrow \texttt{p3} \times exp\left(\eta - \frac{\texttt{p4}\times\texttt{price}}{\texttt{baseRTT}}\right)$$

2) Set the congestion window:

```
tmp ← cwnd - estWnd
if tmp > maxDecrement then
  cwnd ← cwnd - maxDecrement
else if tmp < -maxIncrement then
  cwnd ← cwnd + maxIncrement
else
  cwnd ← estWnd
end if
```

Variables and parameters:

η	state variable
price	price received in the most recent packet
estWnd	calculated target window size
cwnd	window size that is used by the sender
p1, p2, p3, p4	parameters
interval	integration period

Fig. 2. Pseudocode of the source algorithm. The calculated window size estWnd is inversely related to the congestion feedback Price received in ACK packets. The window size of the sender, however, is not allowed to increase or decrease by more than maxIncrement and maxDecrement number of packets respectively, so that we avoid overreacting to the signal.

B. Source Algorithm

Senders use the source algorithm to adjust their congestion window size in response to the feedback from the routers calculated by (1). While a vast majority of TCP protocols adjust its sending rate based on the sum of the congestion levels on the end-to-end path, MaxNet only uses the congestion level from the most severely congested bottleneck (i.e. the *max*imum of the prices, thus the term MaxNet) as described in Fig. 1. The window control algorithm we choose is given in Fig. 2. The target congestion window is calculated in the first step. The true window size of the sender is set in the second step. It is generally equal to the window size calculated in the first step but it is not allowed to increase or decrease by more than maxIncrement and maxDecrement number of packets respectively. Parameters p1 and p3 determine the rate of convergence, the higher the value the faster the convergence rate. Similarly, parameters p2 and p4 determine the severity of reaction to the congestion signal. Proper tuning of the constants is essential to achieve convergence and scalable performance. The source algorithm uses the control law proposed in [9] and corresponds to a particular choice of utility function of the general MaxNet framework in [1] [2] [3]. Wydrowski et. al. prove that this framework guarantees max-min fairness and stability for networks of arbitrary topology, delay, loss,

number of sources and capacities.

III. Implementation

We choose to implement the MaxNet control scheme on top of Linux's existing TCP/IP framework. Working with such a robust and mature codebase significantly speeds up the development cycle from what it would have been had we chosen to develop a new protocol from scratch. Because we implement on top of TCP, we refer to our implementation as TCP MaxNet, though the MaxNet congestion control scheme could ostensibly be applied to any transport protocol concerned with congestion control, or even developed as a standalone protocol.

A. Communication Between the AQM and Source Algorithms

We suggest introducing a new TCP Option and use 6 bytes in each packet header to carry information about the level of congestion on the end-to-end path. The option format we use is depicted in Fig. 3. The purpose of the first byte is to advertise that this is TCP MaxNet option and the second byte advertises the length of the remaining fields. It is very important for performance of the protocol that the size of our option does not exceed 8 bytes (including the 2 leading bytes), as we will later explain, and thus we are left with 6 bytes divided between price and echo field.

Routers compare the price field in each packet that is dequeued with the price calculated by their AQM algorithm. If the calculated price exceeds the price advertised in the packet, the calculated value is written in the price field. The option is subsequently processed at the destination, price is written into the echo field and sent back to the source in an ACK. Once the source receives the ACK it reads the echo value and uses it to adjust its window size. The code of the sender and receiver is identical, and our design allows duplex TCP connections.

B. Implementation of the AQM Algorithm

The router code is implemented as `iproute2` dynamically loadable module for Linux. The module is loaded using the `tc` program as an interface to the kernel and it accepts several parameters. `capacity` is the link capacity C from (1) in Kbps, `mu` is the efficiency parameter μ from (1) and `frequency` is the time interval between price calculations in microseconds. For practical purposes it is not desirable to set the frequency too small. Too small a time interval makes the calculation sensitive to bursts. Moreover, less frequent price calculation, say every 3 ms will not jeopardize the convergence properties of the algorithm since the time required to signal the price to the sender is typically much higher than 3 ms. The

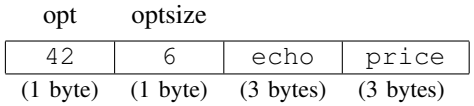

Fig. 3. MaxNet option format.

module calculates $Price$ on a specified link according to (1). However, since price calculation is called from dequeue(), the code might not be called every `frequency` microseconds. We solved this problem by checking the elapsed time dt in dequeue() and performing the following calculation if $dt >$ `frequency`:

$$Price(t+dt) = Price(t) + \frac{X(t)}{C} - \mu dt. \qquad (2)$$

Since the module is implemented as part of the Linux kernel, the code has to be developed to work in an environment without native floating point support. We perform all calculations with sufficient accuracy by first multiplying all parameters by an appropriate constant to convert the values into a convenient range. For example, to calculate μdt, where $\mu = 0.96$ and $dt = 10,000$, μ is converted to $(96*1024/100) = 983$. The final result $C\mu$ is divided by 1024 before next use, which can be done without loss of accuracy if the typical value is much larger than 1024.

When the router changes the price value in the packet, the checksum of the packet has to be updated. We implement an incremental procedure that accepts the old checksum, old price and new price value and outputs the new checksum [10].

C. Implementation of the Source Algorithm

The client side modification involves integrating the new TCP protocol into the kernel TCP code. We introduce a system control variable that allows us to switch the algorithm on and off in runtime. Parameters `p1` through `p4` from Fig. 2 were also implemented as system control variables, allowing us to change the values promptly. Per connection variables such as η and `estWnd` from Fig. 2 are retained between the calls of an algorithm that calculates the congestion window in a control structure located in `struct tcp_sock`.

The main part of the code calculates new window size according to the algorithm from Fig. 2. The calculation is performed after receiving a new ACK. The calculated window size is subsequently saved in struct maxnet. Whenever the standard TCP algorithm changes the value of the congestion window, TCP MaxNet overrides the window size with its calculated value. The only exception is a timeout when we follow the standard TCP behavior. It is important to notice that the calculated window is enforced after loss, whereas the standard Linux TCP implementation decreases the window size. The decrease is not necessary for TCP MaxNet, because it decreases its sending rate as soon as it receives positive price from the link and thus it does not experience congestion based loss.

The window size calculation requires evaluation of an exponential function with a real parameter in the Linux kernel, where only integer calculations are available. We solve this problem by proper scaling of the variables, and by hard-coding a lookup table of a finite number of values of the exponential. The lookup table, consisting of several hundred records, must be declared as a global variable in order to save limited stack space.

The maximum length of the TCP Options in the Linux kernel is limited to 60 bytes. 20 bytes are used for the TCP header, 12 bytes are used for the timestamp option and 28 bytes remain for selective acknowledgements (SACKs) and TCP MaxNet options. In standard TCP implementations, one can have up to 3 SACK blocks per packet header. They use 4 bytes plus 8 bytes per block, requiring up to 28 bytes and not leaving any space for TCP MaxNet options. Therefore, we are forced to change the standard SACK implementation and allow at most 2 blocks per SACK. This leaves 8 bytes for TCP MaxNet options. It is important that the MaxNet option does not use up more than 8 bytes. Availability of at least 2 blocks per SACK is crucial for the protocol's performance under loss [11]. In Section IV we will show that TCP MaxNet achieves excellent performance under loss even with only 2 blocks per SACK.

IV. Performance

Performance evaluation of TCP MaxNet and comparison of performance to two other protocols, BIC TCP and FAST TCP, has been conducted on a WAN network using WAN in Lab [8]. WAN in Lab is a wide area network consisting of an array of reconfigureble routers, servers and clients. The backbone of the network is connected by two 1600 km OC-48 links introducing a large amount of real propagation delay. Our experimental setup consisting of two Linux routers and four Linux end hosts is depicted in Fig. 4. TCP MaxNet router code is installed on the routers (labled Bottleneck Router 1 and Bottleneck Router 2) and the client side code is installed on the end hosts. In order to limit the throughput at the Bottleneck Router 1 and Bottleneck Router 2 and to control the buffer size we use a token bucket filter. The bottleneck throughput in our experiments is limited to 10 Mbps. Since our topology allows us to introduce 14 ms and 28 ms one way propagation delay, we limit the buffer size in the router to be 30 KB. This is slightly less than the bandwidth delay product for both paths. Appropriate choice of buffer size is important. Loss based algorithms, such as BIC TCP, need to fill the buffers before decreasing their sending rate as a reaction to loss, and thus TCP MaxNet could gain an unfair advantage in an experiment with unrealistically large buffers.

The parameters of the XCP router code, described in detail in previous section, are chosen as follows: `capacity` $=$ $10,000$ Kbps, `mu` $= 0.96$, `frequency` $= 10,000$ ms. The link speed of 10 Mbps at the routers requires `capacity` $=$ $10,000$ Kbps and `mu` $= 0.96$ implies 96% target utilization of the link. While choice of a higher value of `mu` tends to give better performance, it causes unstability in some cases because the algorithm overestimates performance of the link. The parameters of TCP MaxNet client code are chosen as follows: `p1` $= 6$, `p2` $= 1$, `p3` $= 2$, `p4` $= 15$, `maxIncrement` $= 2$, `maxDecrement` $= 20$. Choice of these parameters is dictated by our requirements for a fast convergence to the target window size for transmissions at various speeds as well as by the requirement of stability of the algorithm.

Traffic on the network is generated in our experiments by `netperf`. We compare performance of TCP MaxNet to performance of BIC TCP, a loss based congestion avoidance algorithm which is now the default for Linux, and FAST TCP, a delay based algorithm.

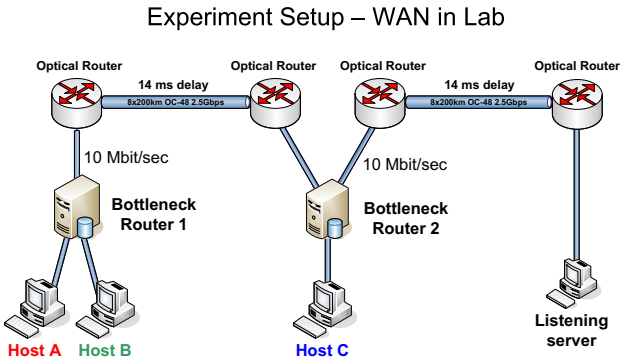

Fig. 4. Wan in Lab architecture. End hosts connected through Linux routers with 14 ms and 28 ms of one way propagation delay.

A. Convergence and RTT

In the first experiment, we observe the convergence and stability of the sending rate of the sender. Traffic is sent from Host C to the listening server over a link with 14 ms one way propagation delay. During the experiment, immediate raw throughput of the protocol and RTT is recorded after arrival of each new acknowledgement. Raw throughput is calculated as number of packets in flight divided by the RTT. Average values of raw throughput in one second increments are depicted in Fig. 5 and average values of RTT in 25 ms increments are depicted in Fig. 6. While TCP MaxNet achieves slightly lower raw throughput than the other protocols, the convergence rate to the target speed as well as stability of the throughput is comparable for all three protocols. The raw throughput of the other protocols may be slightly higher because it also includes packets that are dropped in the routers. This is a very rare event when TCP MaxNet is used. TCP MaxNet is able to transfer data with much lower latency than the other protocols. BIC TCP, as well as all other loss based congestion protocols, increase the sending rate untill all the buffers along the end-to-end path are filled with data and overflow. Therefore, loss based algorithms experience very high latency and the RTT of BIC TCP exceeded 70 ms, twice the latency of TCP MaxNet. As follows from Fig. 6, FAST TCP experiences lower latency because it does not need to fill the buffer at the router in order to fully utilize the capacity of the link. However, the latency of FAST TCP is still much higher than that of TCP MaxNet. TCP MaxNet is able to achieve very low average latency of 29 ms on a link with one way propagation delay 14 ms because the explicit signal from the router causes the sender to decrease its window size whenever queue starts to build up in the router, thus preventing queue buildup.

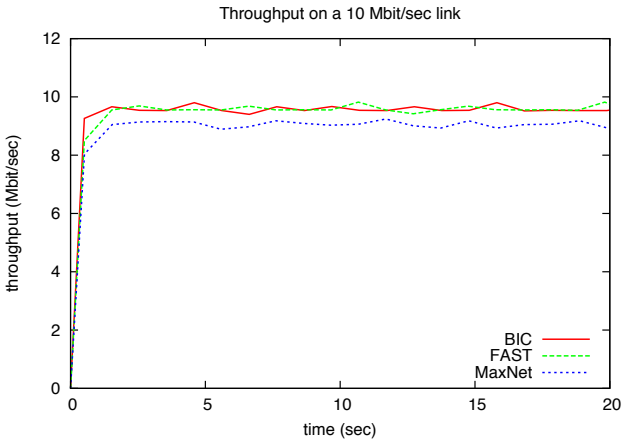

Fig. 5. Throughput of TCP MaxNet, FAST TCP and BIC TCP converging to 10 Mbps, the bottleneck capacity.

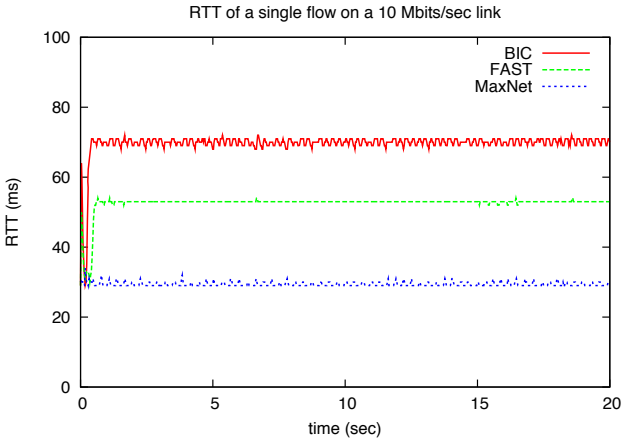

Fig. 6. RTTs of TCP MaxNet are much lower than RTTs of FAST TCP or BIC TCP indicating short queues in the routers.

B. Fairness

One objective of TCP protocols is to share common bottleneck bandwidth fairly under various circumstances. However, stability requirements of commonly used congestion avoidance algorithms impose dependence of the target throughput on RTT. Therefore, it is very common for TCP protocols not to share bottleneck capacity fairly among flows with RTTs that differ. We provide two experiments that assess fair sharing - one for flows with identical RTTs and the other for flows with different RTTs. Our results show that TCP MaxNet is able to share common bottleneck capacity fairly in both cases.

Each of the leftmost three plots in Fig. 7 shows the transfer rates of four different TCP flows sharing a common bottleneck. All the four flows in each experiment have identical RTT and use the link between Host C and the listening server with has 14 ms one way propagation delay. Results for TCP MaxNet, FAST TCP and BIC TCP are shown respectively. We conclude that all the three TCP algorithms are able to share the bottleneck speed fairly in this case. Also the transmission speed converges quickly to the target speed for all the protocols, even though FAST TCP and BIC TCP tend

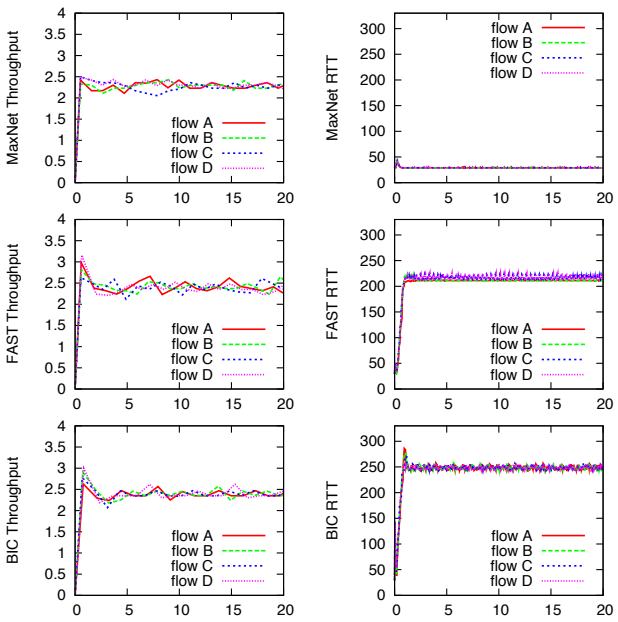

Fig. 7. Multiple TCP flows with identical RTTs share common bottleneck capacity fairly. The average throughput for each of the TCP MaxNet flows ranged between 2.27 Mbps and 2.31 Mbps.

to overestimate the target capacity shortly after the start of the transmission, resulting in heavy loss of packets at the router. The raw throughputs of FAST TCP and BIC TCP exceed 10 Mbps for a brief period of time after the start of the transmissions because raw throughput depends on the number of packets in flight and the immediate value does not reflect packets that may be dropped by the router. RTT for the same experiment is depicted in the next three plots in Fig. 7. Similarly as in the experiment from section A, TCP MaxNet achieves much lower latency than the other protocols. While the average RTT for TCP MaxNet remains at about 29 ms, the RTT of FAST TCP and BIC TCP exceds 200 ms and 250 ms respectively.

Sharing of bandwidth among flows with differing RTTs is depicted in Fig. 8. The leftmost three plots each show throughput of two flows on links with round trip latency 57 and 29 ms. The rightmost three plots in Figure 8 show RTT values corresponding to the two flows. The flows are initiated between Host A, Host C and the listening server. While TCP MaxNet shares the bottleneck speed equally, both TCP Maxnet and BIC TCP prefer the flow with lower RTT. The right hand half of Fig. 8 shows RTTs for the experiment, and, once again, TCP MaxNet is able to achieve very low latency by keeping the buffers in both routers used in this experiment empty.

C. Performance under Loss

Performance of TCP MaxNet under loss is evaluated by measuring throughput of a TCP flow between host C and the listening server. Loss is introduced by `netem` in the router on all its interfaces. Average throughput of the protocol for a flow with duration 10 seconds and with loss at the router ranging from 1-5% is summarized in Table I. TCP MaxNet

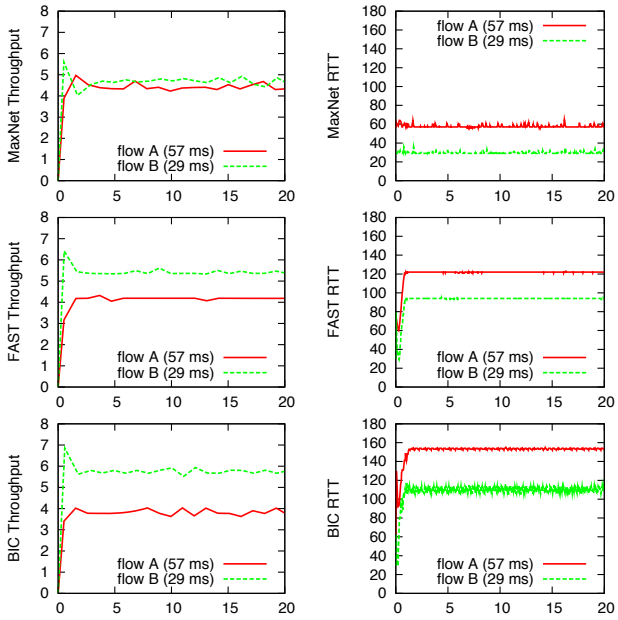

Fig. 8. Only TCP MaxNet was able to share common bottleneck fairly for flows on links with different porpagation delays. The average throughput of the TCP MaxNet flows was 4.40 Mbps and 4.68 Mbps respectively.

outperforms the other protocols in losy environments because both FAST TCP and BIC TCP assume that all loss is result of congestion, thus decreasing the sending rate more than necessary when non-congestion based loss is introduced.

loss rate	MaxNet	FAST	BIC
0%	8.89	9.36	9.47
1%	8.58	5.56	3.51
2%	8.25	4.75	2.36
3%	7.83	4.13	1.81
4%	7.65	3.07	1.39
5%	6.85	3.08	1.21

TABLE I

BEHAVIOR IN LOSSY CONDITIONS - THROUGHPUT IN MBPS.

D. Directions for Further Research

In the future, it would be desirable to compare performance of MaxNet and XCP. XCP is of special interest to us because of its similarity to MaxNet and its exceptional performance in practice [7]. In light of the theoretical suggestions that MaxNet achieves universal max-min fairness and recent research that suggests XCP does not share bandwidth fairly in certain network topologies [12], it would be of interest to determine if MaxNet can sustain its advantage when deployed in large computer networks. More research and testing, however, are needed in relation to the deployment of the algorithm, particularly in mixed environments. The greatest obstacle to performing wide scale testing of the algorithm is the fact that it requires router modifications in addition to modifications of the end-hosts.

V. CONCLUSION

We are the first to provide working implementation of TCP MaxNet and to report on performance of the protocol. Our contribution is twofold. First we provide solutions to various challenges one faces when implementing the protocol, such as a method for calculating exponential with sufficient accuracy in the Linux kernel, and we point out important changes in the Linux kernel that are needed to achieve good performance under various conditions. Our second contribution is performance evaluation of the protocol. Our results confirm previous theoretical predictions and show that TCP MaxNet is for many reasons an attractive protocol. We demonstrate that TCP MaxNet is capable of fair sharing of bottleneck capacity even when the propagation delays of the competing flows vary. Moreover, we demonstrate that TCP MaxNet achieves extraordinary performance in lossy environment. Finally, we show that MaxNet TCP flows have low latency and result in short router queue sizes.

ACKNOWLEDGMENT

We would like to acknowledge support of the Networking Laboratory at the California Institute of Technology. We also would like to acknowledge George Lee, who helped us to set up experiments on the WAN in Lab. This is part of the FAST and WAN in Lab project at Caltech supported by NSF (EIA-0303620), Cisco ARO, and AFOSR.

REFERENCES

[1] B. Wydrowski, L. L. H. Andrew, and M. Zuckerman, "MaxNet: A congestion control architecture for scalable networks," *IEEE Commun. Lett.*, vol. 7, pp. 511–513, Oct. 2003.

[2] B. Wydrowski and M. Zuckerman, "MaxNet: A congestion control architecture for MaxMin fairness," *IEEE Commun. Lett.*, vol. 6, pp. 512–514, Nov. 2002.

[3] B. Wydrowski, L. L. H. Andrew, and I. M. Y. Mareels, "MaxNet: Faster flow control convergence," 2005, unpublished.

[4] C. Jin, D. X. Wei, and S. H. Low, "TCP FAST: motivation, architecture, algorithms, performance," in *Proceedings of IEEE Infocom*, Mar. 2004.

[5] C. Jin, D. Wei, S. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh, "FAST TCP: From theory to experiments," *IEEE Network*, vol. 19, pp. 4–11, Feb. 2005.

[6] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, Pittsburgh, PA, 2002.

[7] Y. Zhang and T. Henderson, "An implementation and experimental study of the eXplicit Control Protocol (XCP)," in *Proceedings of IEEE Infocom*, Miami, Florida, Mar. 2005.

[8] "WAN in Lab," 2005. [Online]. Available: http://wil.cs.caltech.edu/

[9] F. Paganini, Z. Wang, J. C. Doyle, and S. H. Low, "Congestion control for high performance, stability and fairness in general networks," *IEEE/ACM Trans. Networking*, vol. 13, no. 1, pp. 43–56, Feb. 2005.

[10] A. Rijsinghani, "Computation of the Internet Checksum via Incremental Update," RFC 1624 (Informational), May 1994. [Online]. Available: http://www.ietf.org/rfc/rfc1624.txt

[11] P. Sarolahti and A. Kuznetsov, "Congestion control in linux tcp," in *Proceedings of 2002 USENIX Annual Technical Conference*, Monterey, CA, June 2002, pp. 49–62.

[12] S. Low, L. Andrew, and B. Wydrowski, "Understanding XCP: Equilibrium and fairness," in *Proceedings of IEEE Infocom*, Miami, Florida, Mar. 2005.