AUTHOR QUERY FORM

ELSEVIER	Journal: COMPNW Article Number: 4689	Please e-mail or fax your responses and any corrections to: E-mail: corrections.esch@elsevier.sps.co.in Fax: +31 2048 52799

Dear Author,

Please check your proof carefully and mark all corrections at the appropriate place in the proof (e.g., by using on-screen annotation in the PDF file) or compile them in a separate list. Note: if you opt to annotate the file with software other than Adobe Reader then please also highlight the appropriate place in the PDF file. To ensure fast publication of your paper please return your corrections within 48 hours.

For correction or revision of any artwork, please consult http://www.elsevier.com/artworkinstructions.

Any queries or remarks that have arisen during the processing of your manuscript are listed below and highlighted by flags in the proof. Click on the 'Q' link to go to the location in the proof.

Location in article	Query / Remark: click on the Q link to go Please insert your reply or correction at the corresponding line in the proof
Q1	Please confirm that given names and surnames have been identified correctly.

Thank you for your assistance.

Computer Networks xxx (2012) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Evaluating TCP-friendliness in light of Concurrent Multipath Transfer

Ilknur Aydin [a,b,*], Janardhan Iyengar [c], Phillip Conrad [d], Chien-Chung Shen [a], Paul Amer [a]

[a] Dept. of Computer & Info. Sciences, Univ. of Delaware, Newark, DE, USA
[b] Dept. of Computer Science, SUNY Plattsburgh College, Plattsburgh, NY, USA
[c] Dept. of Computer Science, Franklin & Marshall College, Lancaster, PA, USA
[d] Dept. of Computer Science & College of Creative Studies, Univ. of California, Santa Barbara, CA, USA

ARTICLE INFO

Article history:
Received 1 September 2010
Received in revised form 23 December 2011
Accepted 18 January 2012
Available online xxxx

Keywords:
Multihoming
SCTP
Concurrent Multipath Transfer
TCP-friendliness
TCP-friendly

ABSTRACT

In prior work, a CMT protocol using SCTP multihoming (termed SCTP-based CMT) was proposed and investigated for improving application throughput. SCTP-based CMT was studied in (bottleneck-independent) wired networking scenarios with ns-2 simulations. This paper studies the TCP-friendliness of CMT in the Internet. In this paper, we surveyed historical developments of the TCP-friendliness concept and argued that the original TCP-friendliness doctrine should be extended to incorporate multihoming and SCTP-based CMT.

Since CMT is based on (single-homed) SCTP, we first investigated TCP-friendliness of single-homed SCTP. We discovered that although SCTP's congestion control mechanisms were intended to be "similar" to TCP's, being a newer protocol, SCTP specification has some of the proposed TCP enhancements already incorporated which results in SCTP performing better than TCP. Therefore, SCTP obtains larger share of the bandwidth when competing with a TCP flavor that does not have similar enhancements. We concluded that SCTP is TCP-friendly, but achieves higher throughput than TCP, due to SCTP's better loss recovery mechanisms just as TCP-SACK and TCP-Reno perform better than TCP-Tahoe.

We then investigated the TCP-friendliness of CMT. Via QualNet simulations, we found out that one two-homed CMT association has similar or worse performance (for smaller number of competing TCP flows) than the aggregated performance of two independent, single-homed SCTP associations while sharing the link with other TCP connections, for the reason that a CMT flow creates a burstier data traffic than independent SCTP flows. When compared to the aggregated performance of two-independent TCP connections, one two-homed CMT obtains a higher share of the tight link bandwidth because of better loss recovery mechanisms in CMT. In addition, sharing of ACK information makes CMT more resilient to losses. Although CMT obtains higher throughput than two independent TCP flows, CMT's AIMD-based congestion control mechanism allows other TCP flows to co-exist in the network. Therefore, we concluded that CMT is TCP-friendly, similar to two TCP-Reno flows are TCP-friendly when compared to two TCP-Tahoe flows.

1. Introduction

A host is *multihomed* if the host has multiple network addresses [1]. We are seeing more multihomed hosts connected to the networks and the Internet. For instance, PCs with one Ethernet card and one wireless card, and cell phones with dual Wi-Fi and 3G interfaces are already common realities. Nodes with multiple radios and radios

* Corresponding author at: Dept. of Computer Science, SUNY Plattsburgh College, Plattsburgh, NY, USA.
E-mail addresses: ilknur.aydin@plattsburgh.edu (I. Aydin), jiyengar@fandm.edu (J. Iyengar), pconrad@cs.ucsb.edu (P. Conrad), cshen@cis.udel.edu (C.-C. Shen), amer@cis.udel.edu (P. Amer).

1389-1286/$ - see front matter © 2012 Published by Elsevier B.V.
doi:10.1016/j.comnet.2012.01.010

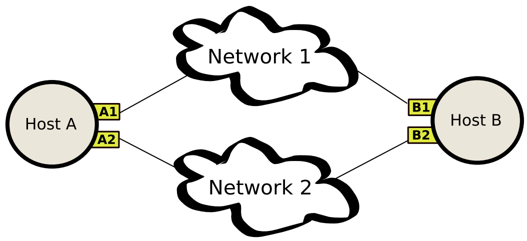


Fig. 1. Example of multihoming (with disjoint paths).

operating over multiple channels are being deployed [2,3]. In addition, Wi-Fi wireless interface cards are now so inexpensive that nodes with multiple Wi-Fi cards and wireless mesh networks (or testbeds) with multiple radios are practical [4,5].

A transport protocol supports multihoming if it allows multihomed hosts at the end (s) of a single transport layer connection. That is, a *multihome-capable transport protocol* allows a *set* of network addresses, instead of a single network address, at the connection end points. When each network address is bound to a different network interface card connected to a different physical network, multiple physical communication paths become available between a source host and a destination host (Fig. 1).

A multihome-capable transport protocol can accommodate *multiple paths* between a source host and a destination host within a *single* transport connection. Therefore, technically, a multihomed transport protocol allows simultaneous transfer of application data through different paths from a source host to a destination host, a scheme termed *Concurrent Multipath Transfer (CMT)*. Network applications can benefit from CMT in many ways such as fault-tolerance, bandwidth aggregation, and increased application throughput.

The current transport layer workhorses of the Internet, TCP and UDP, do not support multihoming. However, the *Stream Control Transmission Protocol (SCTP)* [6,7] has built-in multihoming support. Since SCTP supports multihoming natively, SCTP has the capability to realize CMT for the network applications. In this paper, we study TCP-friendliness of SCTP-CMT in the Internet.

TCP is the de facto reliable transport protocol used in the Internet. Following the infamous Internet *congestion collapse* in 1986, several congestion control algorithms were incorporated into TCP to protect the stability and health of the Internet [8]. As a direct response to widespread use of non-TCP transport protocols, the concept of *TCP-friendliness* emerged [9]. Briefly, TCP-friendliness states that the sending rate of a non-TCP flow should be approximately the same as that of a TCP flow under the same conditions (RTT and packet loss rate) [10]. In addition, a non-TCP transport protocol should implement some form of congestion control to prevent congestion collapse. Since the 1990s, new developments, such as multihoming and CMT, challenge this traditional definition of TCP-friendliness which was originally introduced for single-path end-to-end connections. For instance, recently, there is substantial activity in the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF) mailing lists (such as *tmrg, tsvwg, iccrg,* and *end2end-interest*) discussing the definition of TCP-friendliness and other related issues (such as compliance with TCP-friendly congestion control algorithms, what can cause congestion collapse in the Internet, Internet-friendly vs. TCP-friendly algorithms, fairness of "flow rate fairness").

In this paper, we survey the historical development of TCP-friendliness and argue that the existing definition should be extended to incorporate SCTP CMT and multihoming. Since SCTP CMT is based on (single-homed) SCTP, we first investigate TCP-friendliness of single-homed SCTP.[1] We then study TCP-friendliness of SCTP CMT according to the traditional definition of TCP-friendliness [9] using QualNet [12] simulations. Note that we developed SCTP and SCTP-based CMT simulation modules in QualNet [13]. We also verified the correctness of our SCTP QualNet module against SCTP ns-2 module [14] before we ran our simulations (see [15] for details).

This paper is organized as follows. Section 2 presents a primer on SCTP and CMT. Section 3 presents the historical development and the formal definition of TCP-friendliness. Section 4 elaborates on the TCP-friendliness of single-homed SCTP. Section 5 evaluates the TCP-friendliness of CMT. Section 6 concludes this paper with summary of our results and future work.

2. Primer on SCTP and CMT

SCTP was originally designed to transport telephony signaling messages over IP networks. Later on the IETF reached consensus that SCTP was useful as a general purpose, reliable transport protocol for the Internet. SCTP provides services similar to TCP's (such as connection-oriented reliable data transfer, ordered data delivery, window-based and TCP-friendly congestion control, flow control) and UDP's (such as unordered data delivery, message-oriented). In addition, SCTP provides other services neither TCP nor UDP offers (such as multihoming, multistreaming, protection against SYN flooding attacks) [16]. In the SCTP jargon, a transport layer connection is called an *association*. Each SCTP packet, or *SCTP protocol data unit (SCTP-PDU)*, contains an SCTP *common header* and multiple data or control *chunks*.

2.1. SCTP multihoming

One of the innovative features of SCTP is its support of multihoming where an association can be established between a *set* of local and a *set* of remote IP addresses as opposed to a *single* local and a *single* remote IP address as in a TCP connection. In an SCTP association, each SCTP endpoint chooses a *single* port. Although multiple IP addresses are possible to reach one SCTP endpoint, only one of the IP addresses is specified as the *primary* IP address to transmit data to the destination endpoint.

The *reachability* of the multiple destination addresses are monitored by SCTP with periodic *heartbeat* control

[1] Note that, although SCTP has "similar" congestion control mechanisms as TCP, subtle differences exist between (single-homed) SCTP and TCP.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

chunks sent to the destination IP addresses. The application data is sent *only* to the primary destination address of an SCTP endpoint. However, if the primary address of an SCTP endpoint fails, one of the alternate destination addresses is chosen to transmit the data dynamically, a process termed *SCTP failover*.

In Fig. 1, both Host A and Host B have two network interfaces, where each interface has one single IP address A1, A2 and B1, B2, respectively. Each interface is connected to a separate (i.e., physically disjoint) network (Network[1] and Network[2]). Therefore, two end-to-end paths exist between Host A and Host B (A1 to B1 and A2 to B2). One SCTP association accommodates all of the IP addresses of each host and multiple paths between the hosts as follows: $([A1, A2 : portA], [B1, B2 : portB])$. Note that, two different TCP connections are needed to accommodate all the IP addresses and the two paths in the same figure, namely $([A1 : portA], [B1 : portB])$ and $([A2 : portA], [B2 : portB])$.

2.2. Concurrent Multipath Transfer (CMT)

Although the standard SCTP [6] supports multiple IP addresses to reach a destination host, only one of the IP addresses, named *the primary IP address*, is used as a destination at any time, to originally transmit application data to a destination host. The IP addresses other than the primary IP address are only used for retransmitting data during failover for the purpose of fault tolerance. Therefore, in reality, the standard SCTP does not fully utilize its potential to facilitate CMT for applications. Research efforts on the concurrent use of the multiple paths within an SCTP association continue [17–22]. The SCTP-based CMT[2] proposed by Iyengar et al. [19,23] is the first SCTP research effort aiming to increase application throughput through concurrency.

Because paths may have different end-to-end delays, *naively*[3] transmitting data to multiple destination addresses (over different paths) within an SCTP association will often result in out-of-order arrivals at a multihomed SCTP receiver. Out-of-order arrivals have negative effects on SCTP throughput due to spurious fast retransmissions, and prevent congestion window growth even when ACKs continue arriving at the sender. CMT [23] proposed algorithms namely Split Fast Retransmit (SFR), Cwnd Update for CMT (CUC), and Delayed ACK for CMT (DAC) to mitigate the effects of reordering at the receiver.

The availability of multiple destination addresses in an SCTP association allows an SCTP sender to select one destination address for the retransmissions. However, in standard SCTP since only the primary destination address is used to send new data, there is no sufficient information about the condition of all other paths. On the other hand, since CMT simultaneously uses all the paths, a CMT sender maintains accurate information regarding the condition of all the paths. Therefore, a CMT sender can better select a path to send retransmissions. CMT proposed and evaluated several retransmission policies. RTX-CWND is one of the proposed policies and sends a retransmission to the active destination address with the highest cwnd value. In this paper, we used RTX-CWND as the retransmission policy of CMT in our simulation studies.

3. TCP-friendliness: background and definition

In a computer network, *congestion* occurs when the demand (load or traffic the data sources pump into the network) is close to or larger than the network capacity. As a result of congestion, (i) the network throughput in terms of what the traffic sinks receive, decreases even though the load in the network increases, (ii) the packet delay in the network increases (as the router queues become longer), and (iii) packet loss increases (since router queues become full and start dropping packets). When no action is taken to prevent or reduce congestion, the network can be pushed into a state called *congestion collapse*, where little or no useful end-to-end communication occurs.

Congestion collapse was first defined and described as a possible threat for TCP/IP-based networks by Nagle in 1984 [24]. The first congestion collapse of the Internet was observed in 1986 when data throughput between Lawrence Berkeley Lab to UC Berkeley significantly dropped to pathetic levels [8]. The original TCP specification [25] only included a *flow control* mechanism to prevent a transport sender from overflowing a transport receiver. TCP did not have any mechanism to reduce the (total traffic) load in the network, when network is yielding signs of congestion. In 1988, V. Jacobson et al. proposed several algorithms (including *slow start* and *congestion avoidance*) based on the *conservation of packets* principle and *AIMD (Additive Increase, Multiplicative Decrease)* mechanisms to address the TCP flaws to prevent congestion collapse [8].

The conservation of packets principle states that "*once the system is in equilibrium (i.e., running stably with full data transit rate), a new packet will not be put into the network unless an old packet leaves the network*" [8]. Jacobson used *ACK clocking* to estimate if an old packet has left the network so that a new packet can be put into the network. TCP's *slow start* algorithm helps TCP come to an equilibrium point (i.e., starting the ACK clocking) quickly by increasing the sending rate of the data source by $1MSS$ per received T-ACK.[4] Once the system is in equilibrium, the *congestion avoidance* algorithm takes over. During congestion avoidance, if there is no sign of congestion (i.e., no packet losses), a TCP source increases its sending rate by $(1 \times MSS/cwnd)$ per received ACK[5] (what is called *additive increase*). When there is a sign of congestion though, the TCP source reduces its sending rate to half of the previous sending rate (what is called *multiplicative decrease*). In their seminal paper [26], Chiu and Jain explain that if all the traffic sources in the network obey the AIMD principle, the network will not have congestion collapse and the bandwidth in the network will be "equally" shared among the

[2] From now on, any mention of *CMT*, *SCTP-based CMT*, or *SCTP CMT* refers to the CMT proposed in [19,23].

[3] That is, simply using the standard SCTP without any modifications.

[4] That is during slow-start, TCP doubles its sending rate per RTT. Therefore, in contrast to its name, during slow start TCP's congestion window opens up exponentially.

[5] Note that during the congestion avoidance phase, TCP congestion window is incremented a total of 1 MSS per RTT, i.e., a linear increase.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

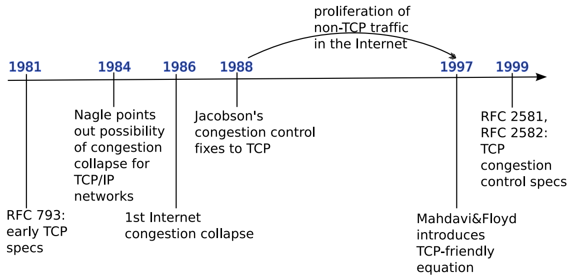


Fig. 2. History of events that led to the doctrine of TCP-friendliness.

flows in the network. The TCP's congestion control algorithms developed by Jacobson were later revised and standardized by the IETF as RFCs 2581 [27] and 2582 [28].[6]

In 1980s the traffic in the Internet was mostly composed of applications running over TCP. Therefore, the congestion control mechanisms of TCP (as explained above) were sufficient to control the congestion in the Internet. However, as the Internet evolved, non-TCP traffic (such as streaming and multimedia applications running over UDP) began consuming a larger share of the overall Internet bandwidth, competing unfairly with the TCP flows, and essentially threatening the Internet's health and stability. As a response, the notion of *TCP-friendliness* emerged [9].

Definition 1. The TCP-friendliness doctrine [10] states that *a non-TCP flow should not consume more resources (bandwidth) than what a conforming TCP flow would consume under the same conditions (segment size, loss, and RTT)*. In addition, *a non-TCP transport protocol should implement some form of congestion control to prevent congestion collapse.*

In 1997, Floyd and Mahdavi introduced the *TCP-friendly equation*[7] [9] (Eq. (1)) which roughly calculates the bandwidth consumed by a TCP flow (conforming with the TCP congestion control algorithms). In 1998, Padhye et al. extended this equation to include timeout events [31]. Fig. 2, summarizes the chronology of events that led to the doctrine of TCP-friendliness.

$$bandwidth\ consumed = \frac{1.22 * MSS}{RTT * \sqrt{loss}} \qquad (1)$$

4. TCP-friendliness of single-homed SCTP

This section investigates TCP-friendliness of single-homed SCTP via QualNet simulations. SubSection 4.1 emphasizes our contributions and motivation to study TCP-friendliness of single-homed SCTP. Subsection 4.2 elaborates the differences between the protocol specifications of TCP and SCTP as well as the conformance of QualNet TCP and SCTP simulation models with respect to the protocol specifications. Subsections 4.3 and 4.4 describe our experimental framework and results and analysis, respectively.

4.1. Motivation and contributions

This section investigates TCP-friendliness of single-homed SCTP. The experiments conducted considers the basic case where only one *competing* pair of TCP and single-homed SCTP[8] flows exist in the network. As mentioned earlier, one of our main goals in this paper is to investigate "TCP-friendliness" of SCTP CMT [19,23]. Since CMT is based on single-homed SCTP, we believe that the first step in understanding TCP-friendliness of CMT is to understand TCP-friendliness of single-homed SCTP. Therefore, the results in this section serve as the first step of the experimental framework in Subsection 5.1. In addition, there exists little research about SCTP vs. TCP in the context of TCP-friendliness. This work also intends to bridge this gap. Furthermore, the comparison work in this section can also be considered as a model for the question of *"how to compare two transport protocols (especially from a congestion control perspective)?"*. In this section, we consider a topology where a single *tight link* [32] is shared by the flows in the network.

4.2. SCTP vs. TCP mechanics

SCTP's congestion control algorithms are designed to be "similar" to TCP's. However, there are subtle differences between the two that can make one transport protocol behave more aggressively than the other under certain circumstances. A few reports provide the similarity and the differences between SCTP and TCP mechanisms [16,33–35]. In this section, we highlight *some* of such subtle differences between single-homed SCTP (based on RFC 4960 [6]) and TCP flavors (based on RFCs 2581 [27], 2582 [28], and 2018 [36]) that we believe are directly related to the discussion of TCP-friendliness.

4.2.1. Comparing transport protocol overheads

- *Transport PDU headers*– A TCP-PDU has 20 bytes of header (without any options), whereas, an SCTP-PDU has 12 bytes of common header plus (data and/or control) chunk headers. For example, an SCTP data chunk header has 16 bytes. If an SCTP-PDU carries a single data chunk, the total header size will be 28 bytes, which is 40% larger than the header of TCP-PDU (without any options).
- *Message-based vs. byte-based transmission*– For SCTP, a *chunk* is the basic unit of transmission. SCTP sender wraps each A-PDU in one chunk.[9] SCTP receiver delivers each received A-PDU in the same way to the receiving application. That is, SCTP preserves message (A-PDU) boundaries between sender and recei-

[6] Note that RFC 2582 is obsoleted by RFC 3782 [29] in 2004.

[7] Note that [30] defined another term called *TCP-compatible flow*. However, based on the definition given in the document, TCP-compatible flow is the same as what was earlier defined as TCP-friendly in [9].

[8] In this paper, unless otherwise stated, SCTP refers to *single-homed* and *single-stream* SCTP associations as in [6].

[9] Note that, if the size of A-PDU is bigger than MTU, then an SCTP sender fragments the A-PDU into multiple chunks. Then, the SCTP receiver reassembles the A-PDU before delivering it to the receiving application.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

ver. In contrast, TCP does byte-based transmission. A TCP sender does not maintain message (A-PDU) boundaries, and for example can concatenate the end portion of one A-PDU with the beginning portion of another A-PDU as the bytes fit into one single TCP-segment (TCP-PDU) during transmission. In the same way, a TCP receiver delivers some or all of an A-PDU to the receiving application, with one system call. The impact of message-based vs. byte-based transmission on the relative performance of SCTP vs. TCP is that, as A-PDU size decreases, the overhead of SCTP per
A-PDU will increase compared to TCP.[10] However, for the simulations in this section, we try to make the SCTP and TCP to be as similar as possible for the sake of TCP-friendliness discussion. Therefore, we do not consider the impact of A-PDU size.[11]

- *Transport protocol ACKs*– SACK[12] is a built-in feature in SCTP while TCP needs to use *SACK option* [36]. SCTP defines a special *SACK chunk* to report gaps in the received data. There are two issues with SCTP's SACK chunk compared to TCP's SACK option. (i) SACK chunk has a relatively large size compared to TCP SACK option. The size of a SACK chunk header without any *gap ACK blocks* is 12 bytes. For example, if there is no gap observed in the received data, SCTP-PDU carrying only a single SACK chunk will be 24 bytes.[13] On the other hand, TCP-PDU carrying no data (and options) will be 20 bytes. (ii) TCP SACK option can have at most 4 SACK blocks[14] (limiting the size of TCP header to 60 bytes in total), while SCTP SACK chunk can have larger number of gap ACK blocks, as long as the size of the SCTP-PDU is smaller than Path MTU. Hence, as the path loss (especially in a high bandwidth path) gets higher, SCTP SACK can better inform the SCTP sender about the missing data compared to TCP, at the expense of increased overhead.
- In addition to the differences of protocol overhead between the basic SCTP and TCP specifications, as mentioned above, we note that QualNet 4.5.1 implements RFC 1323 [38] for high performance TCP. Therefore, the *TCP window scaling* option,[15] is implemented together with the *TCP timestamps* option, which adds 12 extra[16] bytes to the TCP header of every TCP-PDU, making the TCP header 32 bytes.

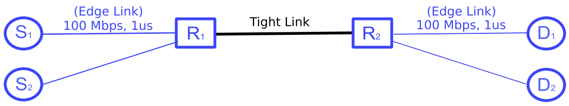


Fig. 3. Simulation topology for single-homed SCTP experiments.

4.2.2. Comparing congestion control mechanisms

- *How to increase cwnd*: Per RFC 2581, a TCP sender increases its congestion window (cwnd) based on the *number of ACK packets* received.[17] In contrast, SCTP counts the *number of bytes* acknowledged within each received ACK packet. Counting the number of ACK packets received rather than the number of bytes acknowledged within each ACK packet causes bigger performance issues for TCP especially when delayed ACKs [1] are used. Note that, we used delayed ACKs in our simulations.
- *When to increase cwnd:* During congestion avoidance, SCTP increases its cwnd only if the cwnd is in full use. This can make SCTP less aggressive in sending data.
- *Initial cwnd size*: Initial TCP cwnd size is 1–2 segments according to RFC 2581.[18] SCTP's initial cwnd size at slow start or after long idle periods is set to $min(4 * MTU, max(2 * MTU, 4380 bytes))$, which will be larger than TCP's initial window size.
- *When to apply slow start vs. congestion avoidance:* SCTP increases its cwnd according to the slow start algorithm when $cwnd \leqslant ssthresh$, and applies the congestion avoidance algorithm, otherwise. On the other hand, RFC 2581 let an implementation choose between slow start and congestion avoidance when $cwnd = ssthresh$.[19]

In summary, messaging overhead of SCTP might be higher compared to TCP (especially if no TCP options used). However, SCTP is a newer protocol compared to TCP; hence, some of TCP's enhancements (such as SACKs, ABC [39], initial congestion window size [40]) that came after RFCs 2581 and 2582 are already built-in features in SCTP. Therefore, it should not be surprising to see that SCTP throughput may be better than TCP's under identical conditions (further on this issue in Subsection 4.4).

4.3. Experimental framework

In the following sub-sections, we describe different aspects of the simulation framework used in this section.

4.3.1. Topology

We designed an experimental framework to explore TCP-friendliness of SCTP in a single shared tight link topology as depicted in Fig. 3. In the figure, two edge links use

[10] Assuming that SCTP does no bundling, and application over TCP connection does not use PUSH flag in TCP header.
[11] Interested readers can look into [37] for the impact of A-PDU size on SCTP vs. TCP throughput.
[12] In addition to the cumulative ACK, transport receiver also selectively sends other missing TSNs.
[13] 12 bytes for common header, 12 bytes for SACK chunk.
[14] Note that, when the TCP PDU also carries a TCP timestamp option, the limit of SACK blocks within a TCP SACK option becomes 3. Time stamp option is activated in our simulations for TCP.
[15] Which let us to have send and receive buffer sizes ⩾64 K.
[16] 10 bytes for the timestamps option, 2 bytes for two *TCP no operation* option.
[17] Note that the ABC (Appropriate Byte Counting) enhancement for TCP is later introduced with RFC 3465 [39]. However, QualNet 4.5.1 does *not* implement ABC in TCP.
[18] Note that, RFC 3390 [40] later on updated TCP's initial cwnd size to be up to 4 K; however, QualNet 4.5.1 does not implement RFC 3390 and keeps TCP initial cwnd size at 2 segments.
[19] QualNet 4.5.1 applies the congestion avoidance algorithm when $cwnd = ssthresh$. Hence, this is the same behavior as in the SCTP specification.

fast Ethernet (with 100 Mbps bandwidth capacity and 1 micro second one-way propagation delay). The tight link is modeled as a full-duplex point-to-point link, with a 45 ms one-way propagation delay. This way, RTT of the paths in the network is 90 ms, similar to US coast-to-coast [41] values. Three tight link bandwidth values (5, 10, and 20 Mbps) were tested for all the cases in Section 4. However, the tight link bandwidth did not influence the results. No artificial packet losses are introduced in the tight link or the edge links. Therefore, all the losses in the simulations are due to buffer overflows of congested traffic at routers R1 and R2. The buffer size at routers R1 and R2 is set to the bandwidth-delay product of the path. We use drop tail queues at the routers. We run simulations with QualNet.[20]

4.3.2. Network traffic

The traffic in the network is composed of two flows. The flows are applications transmitting data over an SCTP association or a TCP connection from S_1 to D_1 and S_2 to D_2, respectively. The traffic flows are greedy (i.e., the applications at the sending hosts S_1 and S_2 always have data to send). In addition, the receiving applications at hosts D_1 and D_2 are always ready to consume whatever the transport layer protocol can deliver. Therefore, the sending rate of the traffic sources is not limited by the application but by the network. The size of each application message (or A-PDU) is 1200 bytes. Similarly, TCP-MSS (maximum segment size) is set to 1212 bytes[21].

4.3.3. Transport protocol parameters

While comparing SCTP and TCP (Subsection 4.2), we tried our best to make the transport protocol parameters as close as possible in the simulations. Table 1 lists what parameters are used in common and per transport layer protocol, respectively. For TCP, we studied both TCP SACK (TCPS) [36] and TCP NEWRENO (TCPNR) [28]. We assumed unlimited send and receiver buffer[22] size at the transport layer so that buffer size is *not* a limiting factor for the transport protocol throughput.

4.3.4. The framework

Our goal is to understand how two flows (TCP and/or SCTP) share the available bandwidth in the network. We investigate two cases.

Case-I: The two flows in the network are *started at the same time*[23]. We use Case-I to investigate *how two flows grow together* by looking into all possible TCP-SCTP combinations.

(i) Start **two TCP** flows at the same time
(ii) Start **two SCTP** flows at the same time
(iii) Start **one SCTP** and **one TCP** flow at the same time

Case-II: Initially only one flow is started[24]. Then, we introduce another flow *when the earlier flow operates at steady-state*[25]. Hence, we explore *how one flow gives way to another flow*. We simulated four combinations in Case II.

(i) Start **one TCP** flow then start **another TCP** flow
(ii) Start **one SCTP** flow then start **another** SCTP flow
(iii) Start **one SCTP** flow then start **one TCP** flow
(iv) Start **one TCP** flow then start **one SCTP** flow

The simulation time for Case-I is 720 s, and the simulation time for Case-II is 2140 s. For Case-I, we looked into performance metrics between 60th and 660th seconds. For Case-II, we looked into performance metrics between 10th and 70th seconds (when there is only one flow in the network) as well as between 280th and 2080th seconds (when both flows operate at steady-state). Note that, we used both TCPS and TCPNR for the TCP-SCTP combinations above.

4.3.5. Performance metrics

The performance metrics we measured in the simulations are presented below. We looked into the long-term (steady-state) values of these metrics. In addition, we looked into the throughput metric over short time dura-

Table 1
Transport protocol parameters and their values used in the simulations.

Scheme	Parameter	Value
TCP specific	Window scaling option[a]	YES
	Timestamps option[b]	YES
	Other TCP parameters	QualNet 4.5.1 default values
SCTP specific	SCTP-RTO-INITIAL	3 s
	SCTP-RTO-MIN	1 s
	SCTP-RTO-MAX	60 s
	SCTP-RTO-ALPHA	0.125
	SCTP-RTO-BETA	0.25
	Heartbeats	OFF
	Bundling	NO
SCTP & TCP	Send Buffer	unlimited
	Receive Buffer	unlimited
	Clock Granularity	500 ms
	Initial ssthresh	$65{,}535 * 2^{14}$
	Delayed ACKs [27,1]	YES[c]

[a] The *window scaling option* is required for TCP to have a receiver buffer size bigger than 64 K. We activated the window scaling option for TCP flows so that TCP sending rate is not limited by the receiver buffer size.
[b] This parameter is automatically activated by QualNet, since QualNet 4.5.1 implements both window scaling and timestamps options (i.e., [38] together).
[c] The transport receiver sends a T-ACK for every other in-sequence TSN received or when the delayed ACK timer expires. The delayed ACK timer is set to 200 ms.

[20] Using svn revision 10 of the SCTP module in QualNet 4.5.1. Note that, QualNet's TCP model uses code converted from the FreeBSD 2.2.2 source code implementation of TCP.
[21] Note that, QualNet 4.5.1 complies with Section 4.2.2.6 of RFC 1122 [1] and calculates the maximum data that can be put into an TCP-PDU based on the *effective-MSS*. Since every TCP-PDU included timestamps option (extra 12 bytes) in our simulations, we set the TCP-MSS to 1212, to let TCP effectively send 1200 bytes of data in each PDU, similar to SCTP.
[22] Send buffer size of each transport protocol is set to $2x bandwidth - delay$ product. Receiver buffer size of each transport protocol is set to a large value such as, $65535 * 2^{14}$ bytes.
[23] In the simulations, we started the two flows at random times within $[0 \ldots RTT]$ to get different randomized results with the repetition of the experiments.
[24] At a random time between $[0 \ldots RTT]$.
[25] In the simulations, the latter flow is started at a random time between $80 sec + [0..RTT]$.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

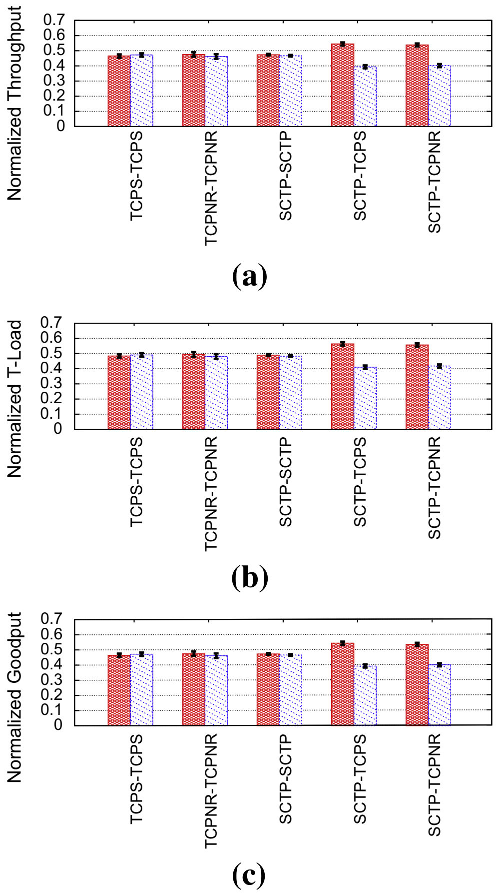


Fig. 4. (a) Normalized throughput, (b) T-Load, and (c) Goodput when *both flows start at the same time.*

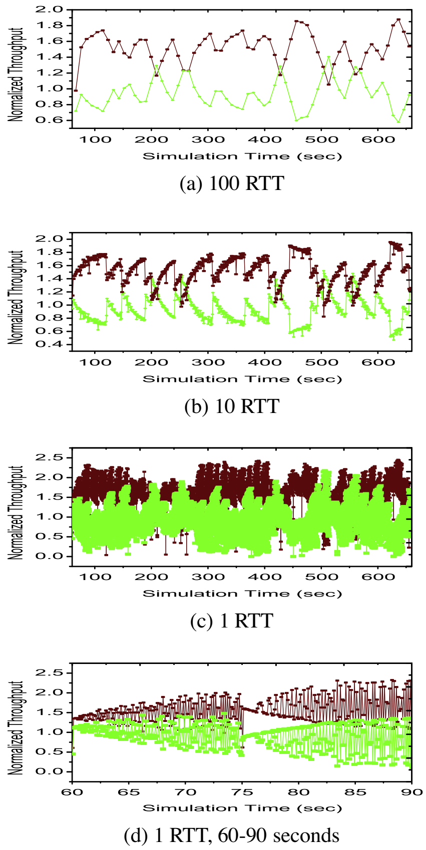


Fig. 5. Throughput of TCPS (green or light color) and SCTP (red or dark color), starting at the same time, for different time intervals. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

tions (1, 10, and 100 RTT) – see Figs. 5–8. The long-term metric values in Figs. 4–6 and Tables 2,3 are averages of 30 runs with 95% confidence intervals.

- *Throughput* – Transport layer data (including original, fast-retransmitted, and re-transmitted data that can be potentially lost) sent to the network by the transport protocol of the sending host per unit time. Throughput of a transport flow is shown as a *fraction* of the tight link bandwidth obtained by the flow in the graphs (i.e., throughput per flow is *normalized* with the tight link bandwidth).
- *Load by the transport protocol or Transport Load (T-Load)* – The actual number of bits sent to the network by the transport protocol per unit time. This includes all the transport layer headers, original, fast-retransmitted, and re-transmitted transport layer data and transport layer ACKs that can be potentially lost. T-Load per transport flow is normalized with the tight link bandwidth.
- *Goodput* – The application layer throughput measured at the receiving host. That is the number of bits delivered to the application layer of the receiving host by the transport layer per unit time. Goodput per transport flow is normalized with the tight link bandwidth.

While the metrics above (throughput, t-load, and goodput) are measured per flow in the network, the following metrics (fairness index, link utilization, and system utilization) are aggregated metrics and measured per configuration.

- *Fairness Index* – This metric is defined by Jain [42] to show fairness (i.e., the "equality" of resource sharing) in a system. Fairness index is a value between 0 and 1, with 1 showing the most fair (equal) allocation of the resources in the system. Assuming λ_i is the rate (throughput) of transport flow *i*, the fairness index of the network is given by Eq. (2), where *n* is the total number of flows in the network.

$$FIndex = \frac{\left(\sum_{i=1}^{n}\lambda_i\right)^2}{n * \left(\sum_{i=1}^{n}\lambda_i^2\right)} \quad (2)$$

- *Link Utilization* – We use Eq. (3) to calculate link utilization (*for the tight link*), where λ_i is throughput of transport flow *i* and *n* is the total number of flows in the network. Our aim in using this metric is to see if the transport flows pump enough data traffic into the

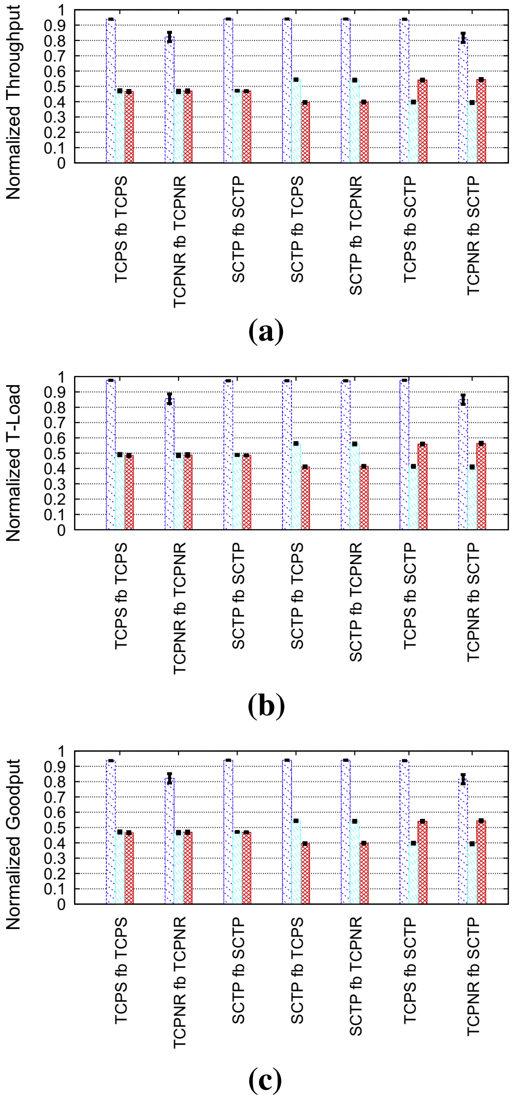


Fig. 6. (a) Normalized throughput, (b) T-Load, and (c) Goodput when *one flow is followed by (fb) another flow*. The first (blue) bar refers to the first flow in steady state when there is no other flow in the network, second (green) bar refers to the first flow in steady state after the second flow is introduced into the network, and third (red) bar refers to the second flow in steady state. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

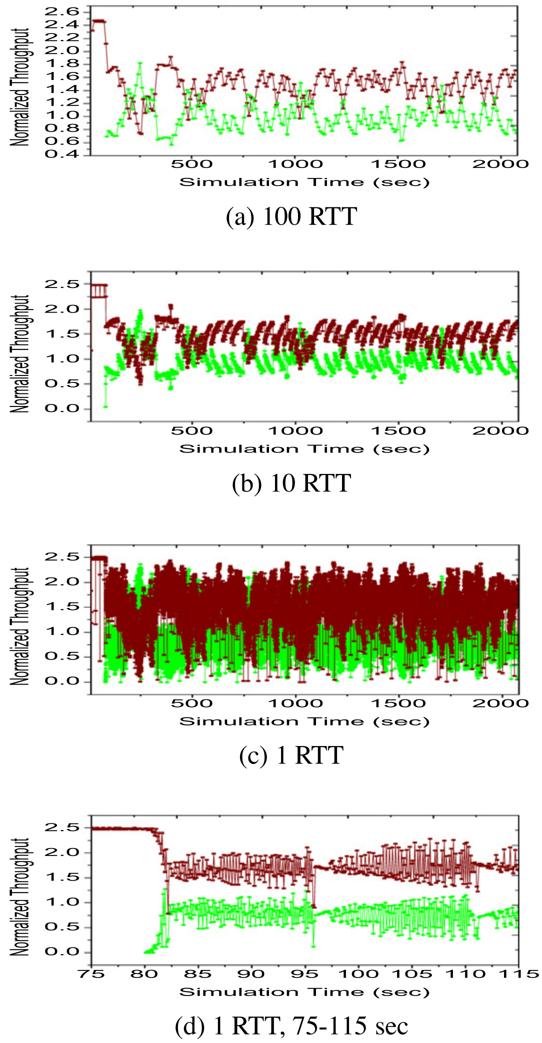


Fig. 7. Throughput of SCTP (red or dark color) followed by TCPS (green or light color), for different time intervals. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

network. We want the link utilization to be high so that the network operates close to its capacity[26].

$$LinkUtil = \frac{(\sum_{i=1}^{n} \lambda_i)}{Tight\ Link\ Bandwidth} \quad (3)$$

- *System Utilization* – This metric is calculated using Eq. (4), where n is the number of flows in the network, α_i is the goodput, and γ_i is the t-load of the transport flow i. Essentially, this metric shows how much of the total load in the network is converted to useful work (i.e., the data received by the applications). One of the signs of congestion collapse is, although there is traffic (load) in the network, the load is not converted into useful work and the network is busy transmitting unnecessary data traffic. Therefore, the higher the system utilization, the further away the system is from congestion collapse.

$$SysUtil = \frac{(\sum_{i=1}^{n} \alpha_i)}{(\sum_{i=1}^{n} \gamma_i)} \quad (4)$$

4.4. Simulation results and analysis

In this section we present the results from two sets of experiments we performed. Subsections 4.4.1 and 4.4.2 discuss the results (i) when both flows in the network start at the same time, and (ii) when one flow starts after the other flow at steady-state, respectively.

[26] That is close to the "knee" as Chiu and Jain suggested [26].

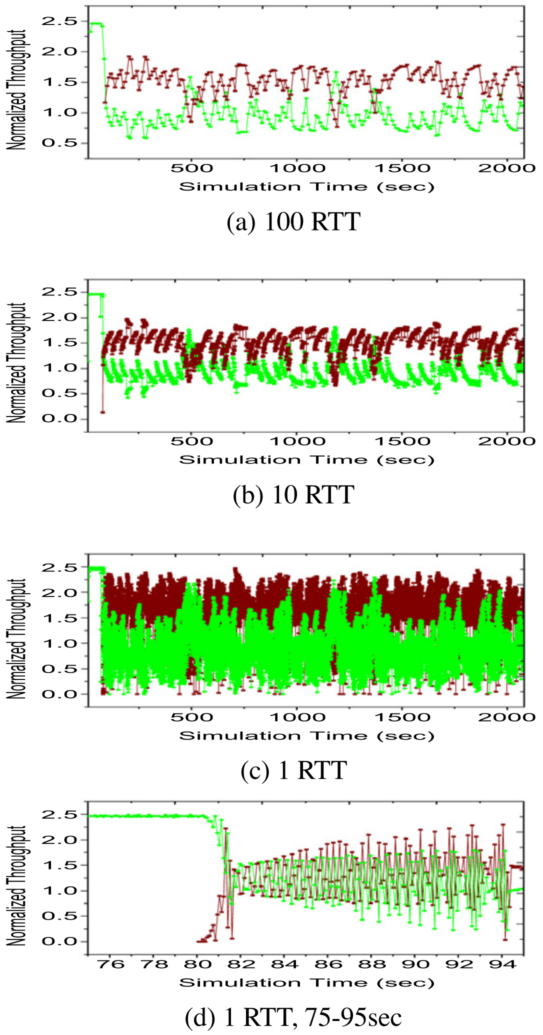


Fig. 8. Throughput of TCPS (green or light color) followed by SCTP (red or dark color), for different time intervals. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4.4.1. Flows starting at the same time

Results for the flows starting at the same time are presented in Figs. 4 and 5 and Table 2.

- *Two TCP flows start together*: From Fig. 4 and Table 2, we observe that two TCP flows (for both TCPS and TCPNR flavors) share the link bandwidth pretty equally, irrespective of increase in bandwidth, as depicted with close individual throughput values per flow in the network as well as the high fairness index values. TCP congestion control algorithms allow aggregated flows to pump enough traffic into the network (where link utilization values are more then 93%). In addition, the system utilization is high confirming that TCP is busy sending useful data traffic (i.e., no signs of congestion collapse).
- *Two SCTP flows start together*: Similar to the two TCP flow case, we observe that two SCTP flows starting at the same time also share the bandwidth equally (Fig. 4), irrespective of increase in tight link bandwidth. The transport load values for the two SCTP flows are also close to the transport loads of the two TCP flows, showing that SCTP and TCP protocol overheads are similar for the configurations we have in the simulations. The link and system utilities are high (⩾94% and ⩾96%, respectively) proving that SCTP congestion control algorithms causing the network to operate at high capacity without any threat of congestion collapse.
- *One SCTP and one TCP flows start together*: From Fig. 4, we observe that irrespective of the increase in the tight link bandwidth, on average SCTP gets 35–41% larger share of the bandwidth compared to TCPS (or TCPNR). However, the link and the system utility values are still high showing a stable network (see Table 2). We looked further into how the throughput of SCTP and TCPS changes over 1, 10, and 100 RTT intervals – Fig. 5. We picked the *worst case* simulation run where SCTP throughput is largest[27] compared to TCPS among all 30 runs. Fig. 5 validates that although SCTP is able to achieve higher throughput than TCPS, even *in the worst case*, SCTP responds to TCP traffic by increasing and decreasing its throughput. That is, even in the most aggressive and imbalanced case of 30 runs, SCTP does not simply take as much bandwidth as it can get; rather, over time SCTP gives and takes in a sharing with TCP. Therefore, the figure helps arguing for SCTP being TCP-friendly.

Table 2
Fairness index, link utilization, and system utilization when *both flows start at the same time*.

Scheme	FI	LinkUtil	SysUtil
TCPS-TCPS	0.996	0.937	0.961
TCPNR-TCPNR	0.993	0.937	0.961
SCTP-SCTP	0.999	0.941	0.966
SCTP-TCPS	0.972	0.939	0.964
SCTP-TCPNR	0.977	0.939	0.964

Table 3
Fairness index, link utilization, and system utilization when *one flow is followed by (fb) another flow*.

Scheme	FI	LinkUtil	SysUtil
TCPS fb TCPS	0.998	0.938	0.961
TCPNR fb TCPNR	0.998	0.937	0.961
SCTP fb SCTP	1.0	0.941	0.966
SCTP fb TCPS	0.975	0.939	0.964
SCTP fb TCPNR	0.976	0.939	0.964
TCPS fb SCTP	0.976	0.939	0.964
TCPNR fb SCTP	0.974	0.939	0.964

4.4.2. Latter flow starts after earlier flow is at steady-state

Results for Case II, where one flow starts earlier and the latter flow starts after the earlier flow is at steady state, are depicted in Figs. 6–8 and Table 3.

[27] In this particular run SCTP gets for about 62% more bandwidth than TCPS.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

- *One TCP flow followed by another TCP flow*: By examining Fig. 6, we first observe that as expected, the throughput of the first TCP flow drops after the second TCP flow is introduced to (more or less) half of its previous value (for both TCPS followed by (fb) TCPS and TCPNR followed by TCPNR cases). The fairness index of the system is also high for both TCPS fb TCPS and TCPNR fb TCPNR (more than 0.99) cases, suggesting that even when the TCP flows start at different times, they still share the link fairly.
- *One SCTP flow followed by another SCTP flow*: By examining Fig. 6 and comparing the results with those of the TCP flows, we observe that the earlier SCTP flow gives way to the latter SCTP flow, and both share the link fairly (see fairness index values in Table 3).
- *One SCTP flow followed by a TCP flow*: In contrast, SCTP does not give a way in an "equally-shared" manner to a later TCP flow (Fig. 6). Although, the SCTP flow's throughput drops after a TCP flow is introduced, on average SCTP ends up getting a 36–39% larger share of the bandwidth than TCPS or TCPNR. We believe that the proposed TCP improvements that have been added[28] into the SCTP's congestion control mechanism are responsible for a (faster and) better loss recovery in SCTP and hence SCTP's larger share of the throughput compared to TCP [34,43–45]. Fig. 7 shows the evolution of throughputs when an SCTP flow is followed by a TCPS flow. For this figure, we plotted the *worst* simulation result out of 30 runs where SCTP vs TCP throughput was most imbalanced[29]. The graph helps argue for SCTP being TCP-friendly as although SCTP gets higher throughput compared to TCP, SCTP still gives way to TCP and shares the bandwidth with the newly introduced TCPS in a give-and-take manner.
- *One TCP flow followed by an SCTP flow*: When it comes to SCTP getting its share of bandwidth from an already stabilized TCP flow (Fig. 6), SCTP again achieves higher throughput than an existing TCP flow. Moreover, the bandwidth obtained is put into useful work by SCTP, as the high system utility values in Table 3 suggest. The evolution of throughput for TCPS and SCTP flows for different time intervals for the most *imbalanced* run out of 30 runs where SCTP achieves 61% more bandwidth than TCPS is depicted in Fig. 8. The figure shows how TCPS gives way to SCTP and how SCTP shares the bandwidth with TCPS in give-and-take manner. The figure again helps argue for SCTP being TCP-friendly.

In summary of all the results in this Subsection (4.4), we discovered that although SCTP's congestion control mechanisms were intended to be "similar" to TCP's, being a newer protocol, SCTP has some of the proposed TCP enhancements already incorporated which results in SCTP performing better than TCP. Therefore, SCTP can obtain larger share of the bandwidth when competing with a TCP flavor that does not have similar enhancements (as in the case of QualNet's TCP implementation). We conclude that SCTP is TCP-friendly but achieves higher throughput than TCP, due to SCTP's better loss recovery mechanisms[30] just as TCP-SACK or TCP-Reno perform better than TCP-Tahoe. Note that, TCP-Reno enhances the fast recovery mechanism of TCP-Tahoe by sending additional data for the continuing duplicate ACKs following a fast retransmission of the data that is assumed lost. The rationale behind this enhancement is that duplicate ACKs show that the subsequent data packets still get across the network and reach to the receiver. As a result of this enhancement, TCP-Reno performs better than TCP-Tahoe but is still considered TCP-friendly[31].

4.5. Related work

Although SCTP has been standardized by the IETF in 2000, there is little work comparing the performance of (single-homed) SCTP with competing TCP flavors. Reference [35] used the *Opnet simulator* to perform initial simulation studies to find possible flaws in the early version of SCTP specification and implementation in 2001. Reference [37] looked into the throughput of competing SCTP and TCP connections in a shared link topology using *SCTP reference implementation* from 1999 on a test-bed (with Linux machines and NIST network emulator). Reference [37] focused on the impact of the message (A-PDU) size in comparing TCP and SCTP throughputs and showed that SCTP is *not more aggressive than TCP* when sharing a link. Reference [34] studied competing SCTP and TCP flows over a *satellite* (high bandwidth-delay product) link using *ns-2*. Reference [34] found out that the subtle enhancements in the SCTP congestion control mechanism help SCTP to recover faster after a loss and hence increase the throughput of SCTP compared to TCP. The results of [34], although for *higher bandwidth-delay product links*, align with our simulation results presented in this section.

5. TCP-friendliness of CMT

This section investigates the TCP-friendliness of CMT. In Section 3, we described the definition and the goals of the TCP-friendliness doctrine. Traditionally, the notion of TCP-friendliness was defined for end-to-end transport connections over a single path. Our goal is to understand and characterize the TCP-friendliness of SCTP-based CMT for transport connections over multiple paths.

The design goal of CMT was to achieve a performance similar to the aggregated performance of multiple, independent, single-homed SCTP associations (called AppStripe). Iyengar et al. showed that the throughput of CMT can be similar or greater[32] (especially when the receiver buffer is unconstrained and the paths are showing similar characteristics) than AppStripe [19]. They studied the performance of CMT under the assumption that the network paths which CMT subflows run over are *bottleneck-independent*. Since we are interested in the TCP-friendliness of CMT, we revise this assumption and investigate how CMT behaves

[28] Refer to Subsection 4.2.
[29] In this particular run, SCTP gets 53% more share of the bandwidth than TCPS.
[30] Reports in [34,43–45] also support our conclusion.
[31] Interested readers can see [11] for a comparison of the congestion control mechanisms of Tahoe, Reno, NewReno, and SACK TCP.
[32] Due to the sharing of the TSN space, CMT is more robust to ACK losses.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

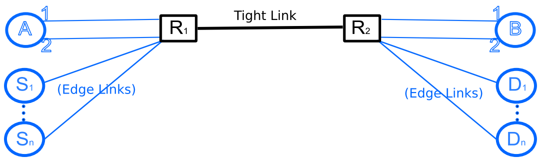


Fig. 9. Simulation topology for the TCP-friendliness of CMT.

when a *tight link*[33] is shared among the CMT subflows and together with other TCP flows.

In the following subsections we describe our experimental framework (Subsection 5.1), simulation results and analysis (Subsection 5.2), followed by the reviews of related work and the recent controversies over the TCP-friendliness doctrine (Subsection 5.3).

5.1. Experimental framework

In the following sub-sections, we describe different aspects of the experimental framework used this section. Experiments are simulated in QualNet[34].

5.1.1. Topology

We designed an experimental framework to explore the TCP-friendliness of CMT in a single shared tight link topology[35] as depicted in Fig. 9. The tight link has 100 Mbps bandwidth capacity and 2 ms one-way propagation delay. Each edge link has 100 Mbps bandwidth capacity and 14 ms one-way propagation delay. This way RTT of the paths in the network is 60 ms, similar to US coast-to-coast [41]. No artificial packet losses are introduced in the tight link or the edge links. Therefore, all the losses in the simulations are due to buffer overflows at routers R1 and R2 caused by network traffic. We used RED queues [47] in our simulations. Note that, when a group of TCP flows share a tight link with drop-tail queues, *global synchronization* and *phase effects* can cause the TCP flows not to get "equal" share of the bandwidth [47–51] (i.e., TCP becomes TCP-unfriendly). Introducing randomness to the network helps reducing the impact of global synchronization and phase effects [47]. We calibrated RED parameters in our simulations so that TCP flows show TCP-friendly behavior (i.e., all TCP flows in the network get "equal" share of the tight link). As recommended by references [47,52], min_{th} is set to 5 packets, max_{th} is set to three times min_{th}, w_q is set to 0.002, and max_p is set to 0.02 in our simulations. The buffer size at routers R1 and R2 is set to the bandwidth-delay product (BDP).

5.1.2. Network traffic

In our experimental topology (Fig. 9), nodes A and B are multihomed hosts while nodes S_i and D_i are single-homed hosts. We first run *n* TCP flows, from source nodes S_i to destination nodes D_i, $1 \leqslant i \leqslant n$. We then add one of the following traffic loads into the network.

- *Flows TCP1 and TCP2:* Two additional TCP flows running over the network paths A1-R1-R2-B1 and A2-R1-R2-B2, respectively.
- *Flows SCTP1 and SCTP2:* Two single-homed SCTP flows running over the network paths A1-R1-R2-B1 and A2-R1-R2-B2, respectively.
- *CMT flow:* a two-homed CMT flow from host A to host B, with two subflows *CMT-sub1* and *CMT-sub2*, running over the network paths A1-R1-R2-B1 and A2-R1-R2-B2, respectively.

For our experiments, *n* is varied as 8, 16, 32, 48, and 64 yielding a total of 10, 18, 34, 50, and 66 network flows[36]. All flows in the network are greedy (i.e., the applications at the sending hosts always have data to send). In addition, the receiving application at each host is always ready to consume whatever the transport layer protocol can deliver. Therefore, the sending rates of the traffic sources are not limited by the applications but by the network. The size of each application message (or A-PDU) is 1200 bytes. Similarly, TCP-MSS is set to 1212 bytes.

5.1.3. Transport protocol parameters

Single-homed SCTP associations and TCP[37] connections are using parameters similar to what has been described in Subsubsection 4.3.3 and Table 1. The CMT association uses DAC and RTX-CWND as RTX policy. Both sender and receiver buffers at the transport connections are unlimited.

5.1.4. The framework

We first started *n* TCP flows from nodes S_i to D_i randomly between the 0th and the 5th seconds of the simulation. Then, at the 10th second, we introduced either (i) the CMT flow, (ii) TCP1 and TCP2 flows, or (iii) SCTP1 and SCTP2 flows into the network. For each case, we measured the performance (mainly the sending rate) of the TCP flows from nodes S_i to D_i, and the performance of the newly introduced flow (s). Our goal in this framework is to see if CMT behaves more or less aggressively than the two independent TCP connections or SCTP associations. We explore answers to the following questions.

- TCP's congestion control algorithms aim to achieve an "equal" share of the tight link bandwidth. How much of the bandwidth sharing an CMT flow could achieve compared to two independent TCP or SCTP flows?
- What is the cost of introducing one CMT flow into the other network flows compared to introducing two independent TCP or SCTP flows?

5.1.5. Metrics

We measured the following metrics in our simulations.

- *Per flow throughput* – similar to the definition given in Subsubsection 4.3.5, we defined throughput (sending rate) of a transport flow as the amount of transport layer data (including the original, the fast-retransmit-

[33] We prefer to use the term *tight link* [32] rather than *bottleneck*, in this paper.
[34] In this section, simulations run with svn revision 10 of the SCTP module in QualNet 4.5.1.
[35] Our topology is similar to the access link scenario in [46].
[36] We counted each CMT subflow as one flow.
[37] All the TCP flows in this section are TCP-SACK connections.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

ted, and the re-transmitted data, that may potentially be lost) put on the wire by the transport protocol sender per unit time.

- *Average (avg.) flow throughput* – is calculated using Eq. (5), where λ_i is the throughput (sending rate) of transport flow $i \in [1..(n+2)]$. While calculating the avg. flow throughput, we counted each CMT subflow as one flow. We expected the avg. flow throughput to be close to the equal share of the available bandwidth in the network.

$$avg.\ flow\ throughput = \frac{\sum_{i=1}^{n+2} \lambda_i}{n+2} \tag{5}$$

- *Fairness Index* – as defined by Eq. (2) in Subsubsection 4.3.5. We measured the fairness index of all the flows in the network (each CMT subflow is counted as one flow) to understand how equally flows in the network actually share the available bandwidth.

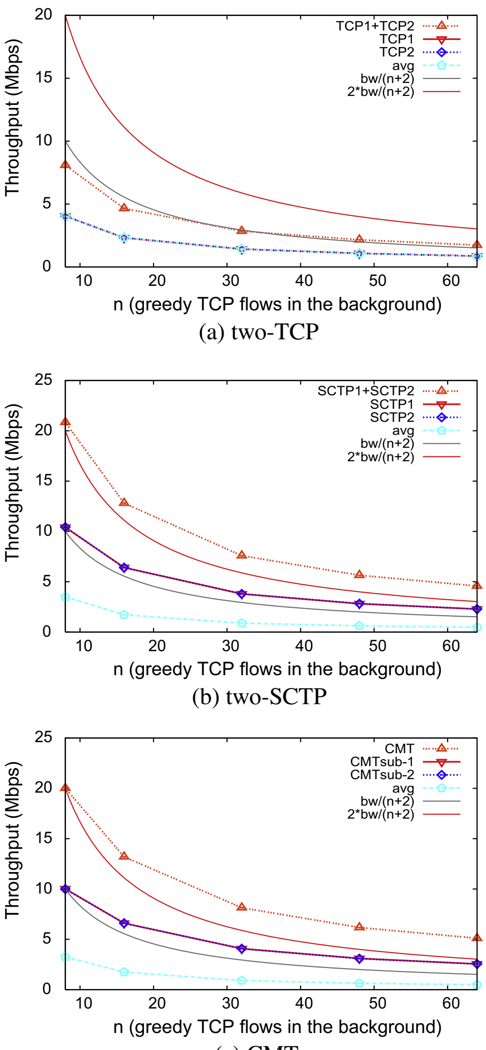


Fig. 10. Throughputs of (a) two-TCP, (b) two-SCTP, and (c) CMT flow together with the avg. flow throughputs.

We run simulations for 36 min. The result for each configuration is averaged over 30 runs with a 95% confidence interval. We measured the metrics between the 3rd and 33rd minutes.

5.2. Simulation results and analysis

Before analyzing the simulation results, we have the following hypotheses for each case.

- *Introducing two TCP flows:* After TCP1 and TCP2 have been introduced into the network, we expect all the flows (including the newly introduced TCP flows) to get an equal share of the available bandwidth.
- *Introducing two SCTP flows:* After SCTP1 and SCTP2 have been introduced into the network, we expect the SCTP flows to get similar or higher throughput than the existing TCP flows in the network. As elaborated in Section 4, the proposed TCP improvements that have been incorporated into the SCTP's congestion control mechanism facilitate better loss recovery and hence improved throughput of SCTP compared to TCP [34,43–45].
- *Introducing the CMT flow:* In a tight-link-independent topology (with drop-tail queues), CMT achieves higher throughput than the independent SCTP flows (especially when the receiver buffer is unconstrained and the paths have similar characteristics), as CMT shares the TSN space and hence is more resilient to losses [19]. Similarly, in a tight-link-dependent topology (with RED queues) as in Fig. 9, we expect CMT to obtain higher throughput (i.e., higher share of the tight link bandwidth) compared to two TCP or two SCTP flows.

The simulated results are depicted in Figs. 10 and 11. We observed the following from the figures.

- *two-TCP case:* From Fig. 10(a), TCP shows TCP-friendly behavior, where TCP1, TCP2 and an average TCP flow in the network all get "equal" throughput, which is less than the ideal bandwidth share, $\frac{bw}{(n+2)}$. The high fairness index values (close to 1) in Fig. 11 for the two-TCP case also confirm the equal sharing of the bandwidth among the TCP flows. We also checked the throughput of all the individual TCP flows in the network for all the n values and again confirmed that all the TCP flows in the network obtained "equal" throughputs (results not shown here due to space constraints).
- *two-SCTP case:* From Fig. 10(b), SCTP1 and SCTP2 get "equal" throughput, and the achieved throughput is higher than both the throughput of an average TCP flow and the ideal share of bandwidth. The low fairness index values in Fig. 11 for the two-SCTP case result from SCTP flows obtaining higher throughput than the TCP flows in the network. However, as we investigated further, adding the SCTP flows into the network does not "starve" any of TCP flows in the network (note that, we have obtained graphs showing the individual flow throughputs, but did not include them in this paper due to space constraints). Such behavior is due to the fact that SCTP implements a congestion control mechanism, and a sender does not frivolously send as much as

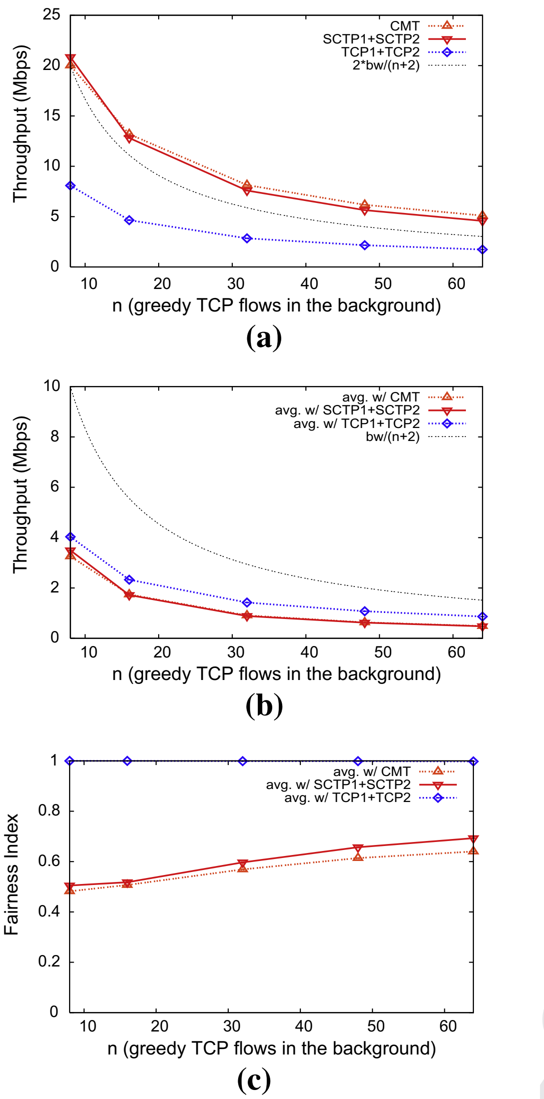


Fig. 11. (a) Throughput of two-TCP vs. two-SCTP vs. CMT flows, (b) Average flow throughputs, and (c) Fairness index of all the flows in the network.

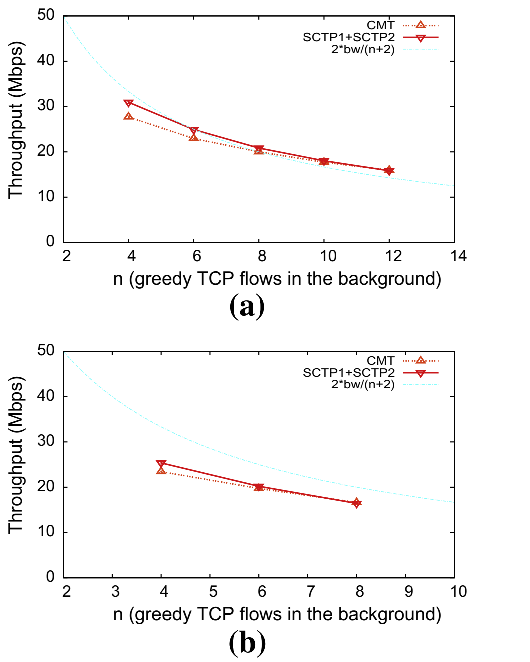


Fig. 12. Throughputs of two-SCTP and CMT flows for smaller n values (a) $w_q = 0.002$ (b) $w_q = 0.001$.

it can. As a result, we see other TCP flows co-existing (without being starved) with two SCTP flows in the network (although SCTP's throughput is higher). Referring back to the definition of TCP-friendliness in Section 3, we conclude that SCTP is TCP-friendly but achieves higher throughput than TCP, due to SCTP's better loss recovery mechanisms [34,43–45], just as TCP-SACK or TCP-Reno performs better than TCP-Tahoe.

- *the CMT case:* From Fig. 10(c), each CMT subflow obtains "equal" throughput, but the achieved throughput is higher than the throughput of an average TCP flow in the network. We also checked the individual subflow throughput and confirmed that CMT subflows perform better than the TCP flows in the network (results not shown in this paper due to space constraints). As depicted in Fig. 11(a), CMT performance is better than the total performance of TCP1 and TCP2. We had also expected CMT to perform better than two SCTP flows. However, CMT is actually showing similar or worse (for $n = 8$) performance than two SCTP flows, which contradicts our earlier hypothesis. To further investigate this issue, we run another set of experiments with n values set to 4, 6, 8, 10, and 12. We observed that the performance of CMT gets worse than two SCTP flows as n gets smaller, as depicted in Fig. 12(a). To investigate the worse performance of CMT compared to two SCTP flows as n gets smaller, we have the following hypothesis.
*hypothesis**: CMT subflows share the same TSN space and ACK information, unlike independent SCTP flows. Therefore, one ACK can simultaneously trigger all the CMT subflows to send data to the network. Consequently, one CMT flow (containing two CMT subflows) can create *burstier* data traffic compared to two SCTP flows. The burstiness causes more packets of the CMT subflows to be marked by the RED queues. Therefore, the CMT flow does not perform as well as we expected (i.e., better than two SCTP flows).
To validate hypothesis*, we examined the RED parameter w_q that can be adjusted to alter RED's responses to burstiness. The rationale is that if we can make RED to react burstiness less aggressively, then we should observe CMT not performing as bad compared to two SCTP flows.
As suggested by [47,48], we changed w_q to be 0.001 (instead of 0.002 used in other simulations in this section), in order for making RED queue to react less aggressively to bursty traffic. The simulation results

for $w_q = 0.001$ are depicted in Fig. 12(b). Comparing Figs.[38] 12(a) and (b), in the latter figure, CMT still performs similarly or worse than SCTP for small n's, but not as badly as in the former figure. Therefore, we have reason to believe that hypothesis* holds. One last question on the comparative performance of CMT and two SCTP flows is why CMT performs worse *for smaller n values*? Intuitively, as n gets smaller the marking probability at the bottleneck RED queue for each flow in the network increases, and hence burstiness affects each flow more.

After this detour to explain the performance difference between CMT and two SCTP flows, let's get back to the discussion of TCP-friendliness. As stated earlier, one CMT flow (with two subflows) has better throughput than two TCP flows and each CMT subflow has better throughput than TCP1 and TCP2, because of the better loss recovery mechanisms implemented in CMT (note that, since CMT is based on SCTP, CMT inherits all the built-in TCP enhancements in SCTP such as appropriate byte counting – ABC) and CMT being more resilient to losses due to sharing of the TSN space and ACK information. We perceive this situation to be similar to two TCP-Reno flows outperforming two TCP-Tahoe flows. CMT also incorporates a TCP-like (AIMD-based) congestion control mechanism and TCP flows can co-exist with CMT in the network (though CMT throughput is higher). Therefore, we conclude a two-homed CMT to be TCP-friendly.

5.3. Related work

As explained in Section 3, the notion of TCP-friendliness emerged in the late 1990s as a reaction to the increase of non-TCP traffic in the Internet. Given the enormous economic, social, and political importance of the Internet, it is easy to justify a conservative approach, such as TCP-friendliness, to address the increasing non-TCP traffic in order not to upset the health and stability of the Internet. However, a lot has changed since the late 1990s. Newer developments (one of them being CMT) demand that we re-consider the notion of TCP-friendliness. In the following sub-sections, we discuss proposals similar to CMT and criticisms against TCP-friendliness.

5.3.1. Other CMT-like schemes

Seminal documents related to the notion of TCP-friendliness [10,30,53] discuss the appropriate level of *granularity* for a "flow" (i.e., end-to-end connection subject to congestion control). Although the IETF allows an HTTP application to open *up to two* TCP connections [54], applications opening multiple TCP connections or splitting one TCP connection into multiple TCP connections (i.e., a flow granularity other than one source IP address-destination IP address pair) have been frowned upon as they get more aggressive share of the bandwidth compared to a single TCP connection. Running between a *set* of source and a *set* of destination IP addresses, and over *multiple* paths, clearly, a CMT flow does not conform to the suggested granularity of a flow in the context of TCP-friendliness.

However, other proposals, similar to CMT, such as CP [55], MulTFRC [56], mulTCP [57], MPAT [58], and PA-MulTCP [59], also aim to achieve aggregated flow rates (i.e., rates similar to aggregated rate of a group of TCP connections). Some of these proposals are based on a window-based AIMD[39] mechanism, while some are based on the TCP-Friendly Rate Control (TFRC) [60,61] protocol. TFRC uses Padhye's TCP-friendly equation [31] to adjust its sending rate. AIMD responds to every congestion indication (packet drop) by multiplicatively decreasing its sending rate. On the other hand, TFRC does not respond to a single packet loss but instead responds to the (measured) average loss rate – or *loss events* that can include one or multiple packets losses. Therefore, TFRC aims to achieve a smoother sending rate compared to window-based TCP, making TRFC more suitable for streaming or multimedia applications.

- CP (Coordination Protocol) defines "flowshare" as the rate of a *single* TCP flow and aims to obtain *multiple* flowshares. CP uses TFRC and estimates a single flowshare (i.e., the available bandwidth for a single TCP flow). Then, CP multiplies the estimated flowshare bandwidth with N to emulate an aggregated flow rate similar to N flowshares.
- Similar to CP, MulTFRC aims to emulate the behavior of N TFRC protocols for providing smoother aggregated sending rate. Unlike CP, instead of naively multiplying the TFRC rate by N, MulTFRC implicitly estimates the loss event per flow in the aggregate flow. MulTFRC extends the TCP-friendly equation to support multiple TCP flows and uses estimated *per aggregate-flow loss rate* in the equation [62]. It is shown that MulTFRC produces smoother sending rates than CP [56].
- MPAT is based on mulTCP [57] which in turn is based on AIMD. MulTCP takes N as an input parameter and aims to behave like N TCP flows. Standard TCP uses the AIMD ($a = 1$, $b = 1/2$) algorithm. That is, if there is a sign of congestion, the congestion window (cwnd) is decreased by $b = 1/2$ of the current congestion window value, while cwnd is increased by $a = 1$ in every RTT if there is no congestion (during steady-state). MulTCP assigns AIMD ($a = N$, $b = 1/2$ N) to the aggregate flow to emulate N TCP flows. However, it is shown in [58] that the loss rate experienced by mulTCP ends up being smaller than the loss rate experienced by N TCP flows. This makes mulTCP more aggressive than N TCP flows, especially as N grows. MPAT is proposed to provided better fairness than mulTCP. MPAT maintains congestion control states as many as the number of flows it manages. Therefore, as N grows, the overhead of MPAT increases.
- Like MPAT, PA-mulTCP is also based on mulTCP. However, unlike MPAT, PA-mulTCP maintains a single congestion window state for the entire aggregate flow (which reduces the overhead) and yet achieves fairer aggregated flow than mulTCP. PA-mulTCP adds an addi-

[38] Each data point in both figures is an average of six runs, where the error bars are almost invisible.

[39] We explained AIMD earlier in Section 3.

tional probe window to detect the sending rate of a real TCP connection and uses this rate to adjust the rate of the aggregated flow.

In addition to the proposals above, in the *tsv working group*, IETF has started Multipath-TCP (MPTCP) [63]. Similar to SCTP and SCTP-based CMT, MPTCP aims to achieve TCP connections over multiple paths (IP addresses), an ability to move data traffic to a less congested path when needed, and to use multiple paths simultaneously to utilize available capacity in the network.

5.3.2. Criticism against TCP-friendliness

The most explicit and blunt criticism on TCP-friendliness came from B. Briscoe starting in his controversial and seminal paper "Flow-Rate Fairness: Dismantling a Religion" [64] in 2007. Briscoe, referring TCP-friendliness as *flow-rate fairness*, instead proposed what he called *cost fairness*. Considering social, real-world, and economic examples, cost fairness (which takes its roots from Kelly's work on weighted proportional fairness [65]) is a more realistic measure of fairness than flow-rate fairness. Briscoe refuses the dogma that *equal flow rates are fair*. Instead, in a system where cost fairness is established, each "economic entity" would be accountable for the costs they caused to others. Cost fairness allocates cost to *bits* instead of flows; hence, cost fairness is immune to the problems such as splitting flow identifiers or opening multiple connections as flow-rate fairness is. Representing the viewpoint of flow-rate fairness, a "rebuttal" [66] not only states the usefulness of flow-rate fairness but also accepts the limitations of flow-rate fairness. Following Briscoe, M. Mathis published an Internet draft [67] arguing that we have to *rethink* the notion TCP-friendliness to keep up with an evolving Internet.

The views from both sides, one clinging onto the flow-rate fairness and the other asking flow-rate fairness to be dismantled, are now being heavily discussed in the IETF mailing lists such as end2end-interest, iccrg, tsvwg, and tmrg. In addition, workshops such as [68] discuss the compelling reasons to replace or renew TCP and its congestion control algorithms. Moreover, bigger research activities and agendas that will change and redesign the entire Internet architecture are underway, [69–72]. We are going towards a world where TCP and TCP-friendliness might not set the standards any longer. However, the authors believe that it will be at least a decade or more before any other view becomes an alternative or displaces TCP-friendliness.

6. Summary of conclusions and future work

TCP-friendliness in the Internet has been traditionally studied in the context of single-path or single-homed transport connections. We designed an experimental framework to investigate TCP-friendliness of CMT, which, unlike standard TCP, uses multiple paths simultaneously. In our experimental framework, we first explored TCP-friendliness of single-homed SCTP (Section 4). We showed that although SCTP's congestion control mechanisms are intended to "be similar to" TCP's, being a newer protocol, SCTP has already incorporated several TCP's enhancements. Therefore, SCTP obtains higher share of the bandwidth when competing with TCP that does not have similar enhancements. We conclude that SCTP is TCP-friendly but achieves higher throughput than TCP, due to SCTP's better loss recovery mechanisms [34,43–45], just as TCP-SACK and TCP-Reno outperform TCP-Tahoe.

In Section 5, we investigated the TCP-friendliness of CMT. We measured the sending rate of one two-homed CMT flow and two SCTP flows, and also the impact of CMT and two SCTP flows on the other TCP flows in the network while sharing a tight link. We found that while sharing a tight link with other TCP flows, CMT's performance is similar or worse than two SCTP flows mainly because of the burstier data traffic that CMT creates compared to two SCTP flows. We also discovered that one two-homed CMT flow obtains higher share of the tight link bandwidth compared to two TCP flows, because of better loss recovery mechanisms in CMT (as CMT inherits built-in TCP enhancements in SCTP). In addition, sharing of ACK information makes CMT more resilient to losses [19]. Although CMT obtains higher throughput than two independent TCP flows, CMT's AIMD-based congestion control mechanism allows other TCP flows to co-exist in the network. We conclude that CMT to be TCP-friendly, just as two TCP-Reno flows are TCP-friendly compared to two TCP-Tahoe flows.

The experimental framework designed in this paper can be extended to have more rigorous study of TCP-friendliness of both single-homed SCTP and CMT. We expect to obtain more insights by investigating (i) an increase in the number of SCTP and CMT flows in the network, (ii) an increase in the number of CMT subflows (hence, concurrency of one CMT flow), (iii) the impact of asymmetric RTTs and edge links, and (iv) the existence of unresponsive flows (similar to UDP) and short-lived TCP flows in the background traffic similar to the testing suite in [46]. In addition to simulations, it will be worth developing the experimental framework in a network emulator to work with SCTP and CMT kernel implementations.

Our final word on TCP-friendliness of CMT is that although this paper investigates the TCP-friendliness of CMT in accordance with the current TCP-friendliness doctrine, we witness hot debates in the IETF that questioned the very foundation of the TCP-friendly Internet. We argue that multihoming and CMT are two of the developments that support a research agenda to pursue alternative fairness criteria for the Internet.

References

[1] R. Braden, Requirements for Internet Hosts – Communication Layers, RFC 1122, Internet Engineering Task Force, October 1989.
[2] DARPA's Wireless Network After Next (WNAN) Program. http://www.darpa.mil/sto/solicitations/WNaN.
[3] BBN's PIRANA project. www.ir.bbn.com/ ramanath.
[4] Y. Liu, Y. Xiong, Y. Yang, P. Xu, Q. Zhang, An Experimental Study on Multi-channel Multi-radio Multi-hop Wireless Networks, in: IEEE Globecom, 2005.
[5] J. Robinson, K. Papagiannaki, C. Diot, X. Guo, L. Krishnamurthy, Experimenting with a multi-radio mesh networking testbed, in: Testbed Workshop on Wireless Network Measurements (Winmee), 2005.
[6] R. Stewart, Stream control transmission protocol, RFC 4960, Internet Engineering Task Force, September 2007.

Please cite this article in press as: I. Aydin et al., Evaluating TCP-friendliness in light of Concurrent Multipath Transfer, Comput. Netw. (2012), doi:10.1016/j.comnet.2012.01.010

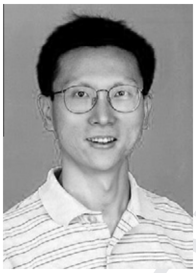
[7] R. Stewart, I. Arias-Rodriguez, K. Poon, A. Caro, M. Tuexen, Stream Control Transmission Protocol (SCTP) Specification Errata and Issues, RFC 4460, Internet Engineering Task Force, April 2006.
[8] V. Jacobson, Congestion Avoidance and Control, in: ACM SIGCOMM, Stanford, CA, 1988, pp. 314–329.
[9] J. Mahdavi, S. Floyd, TCP-friendly unicast rate-based flow control, Techical note sent to end2end-interest mailing list, January 8, 1997.
[10] S. Floyd, K. Fall, Promoting the use of end-to-end congestion control in the internet, IEEE/ACM Transactions on Networking 7 (1999) 458–472.
[11] K. Fall, S. Floyd, Simulation-based comparisons of Tahoe, Reno, and SACK TCP, Computer Communications Review 26 (1996) 5–21.
[12] Scalable Network Technologies, Inc. http://www.scalable-networks.com.
[13] I. Aydin, sCTP QualNet Simulation Module. degas.cis.udel.edu/SCTP/.
[14] A. Caro, J. Iyengar, sCTP ns-2 Simulation Module. pel.cis.udel.edu.
[15] I. Aydin, C.-C. Shen, SCTP QualNet Simulation Module: Details and a Comparison with SCTP ns-2 Module, Technical Report, 2009/336, University of Delaware, Newark, DE, April 2009.
[16] R. Stewart, P.D. Amer, Why is SCTP needed given TCP and UDP are widely available?, ISOC MEMBER BRIEFING 17, www.isoc.org/briefings/017/index.shtml, June 2004.
[17] C. Casetti, W. Gaiotto, Westwood SCTP: load balancing over multipaths using bandwidth-aware source scheduling, in: IEEE VTC2004-Fall, vol. 4, 2004, pp. 3025–3029.
[18] A.A. El, T. Saadawi, M. Lee, LS-SCTP: a bandwidth aggregation technique for stream control transmission protocol, Computer Communications 27 (2004) 1012–1024.
[19] J. Iyengar, End-to-end Concurrent Multipath Transfer Using Transport Layer Multihoming, Ph.D. thesis, University of Delaware, Newark, DE, USA, April 2006.
[20] J. Liao, J. Wang, X. Zhu, cmpSCTP: An Extension of SCTP to Support Concurrent Multi-Path Transfer, in: IEEE ICC, 2008, pp. 5762–5766.
[21] F. Perotto, C. Casetti, G. Galante, SCTP-based Transport Protocols for Concurrent Multipath Transfer, in: IEEE WCNC 2007, Hong Kong, 2007, pp. 2969–2974.
[22] G. Ye, T. Saadawi, M. Lee, IPCC-SCTP: an enhancement to the standard SCTP to support multi-homing efficiently, in: IEEE International Conference on Performance, Computing, and Communications, 2004, pp. 523–530.
[23] J. Iyengar, P. Amer, R. Stewart, Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths, IEEE/ACM Transactions on Networking 14 (5) (2006) 951–964.
[24] J. Nagle, Congestion Control in TCP/IP internetworks, RFC 896, Internet Engineering Task Force, January 1984.
[25] J. Postel, Transmission Control Protocol, RFC 793, Internet Engineering Task Force, September 1981.
[26] D.-M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computer Networks and ISDN Systems 17 (1) (1989) 1–14.
[27] M. Allman, V. Paxson, W. Stevens, TCP Congestion Control, RFC 2581, Internet Engineering Task Force, April 1999.
[28] S. Floyd, T. Henderson, The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 2582, Internet Engineering Task Force, April 1999.
[29] S. Floyd, T. Henderson, A. Gurtov, The NewReno Modification to TCP's Fast Recovery Algorithm, RFC 3782, Internet Engineering Task Force, April 2004.
[30] B. Braden, D. Clark, J. Crowcroft, S. Deering, D. Estrin, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, RFC 2309, Internet Engineering Task Force, April 1998.
[31] J. Padhye, V. Firoiu, D. Towsley, J. Kurose, Modeling TCP Throughput: A Simple Model and its Empirical Validation, ACM SIGCOMM (1998) 303–314.
[32] M. Jain, C. Dovrolis, End-to-End Available Bandwidth: Measurement methodology, Dynamics, and Relation with TCP Throughput, IEEE/ACM Transactions on Networking 11 (4).
[33] R. Alamgir, M. Atiquzzaman, W. Ivancic, Impact of retransmission mechanisms on the performance of SCTP and TCP, in: AMCOS'05: Proceedings of the 4th WSEAS International Conference on Applied Mathematics and Computer Science, World Scientific and Engineering Academy and Society (WSEAS), Rio de Janeiro, Brazil, 2005.
[34] R. Alamgir, M. Atiquzzaman, W. Ivancic, Effect of Congestion Control on the Performance of TCP and SCTP over Satellite Networks, in: In NASA Earth Science Technology Conference, 2002.
[35] R. Brennan, T. Curran, SCTP congestion control: Initial simulation studies, in: International Teletraffic Congress (ITC 17), Brazil, 2001.
[36] M. Mathis, J.Mahdavi, S.Floyd, A.Romano, TCP Selective Acknowledgment Options (SACK), RFC 2018, Internet Engineering Task Force, October 1996.
[37] A. Jungmaier, M. Schopp, M. Tuxen, Performance Evaluation of the Stream Control Transmission Protocol, in: IEEE Conference on High Performance Switching and Routing (HPSR), Germany, pp. 141–148, 2000.
[38] V. Jacobson, R. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, Internet Engineering Task Force, May 1992.
[39] M. Allman, TCP Congestion Control with Appropriate Byte Counting (ABC), RFC 3465, Internet Engineering Task Force, February 2003.
[40] M. Allman, S. Floyd, C. Partridge, Increasing TCP's Initial Window, RFC 3390, Internet Engineering Task Force, October 2002.
[41] S. Shakkottai, R. Srikant, N. Brownlee, A. Broido, K. Claffy, The RTT distribution of TCP flows in the Internet and its impact on TCP-based flow control, Tech. Rep. 2004-02, CADIA, 2004.
[42] R. Jain, W. Hawe, D. Chiu, A Quantitative measure of fairness and discrimination for resource allocation in Shared Computer Systems, Tech. Rep. 301, DEC, September 1984.
[43] P. Natarajan, Leveraging Innovative Transport Layer Services for Improved Application Performance, Ph.D. thesis, University of Delaware, Newark, DE, USA, February 2009.
[44] P. Natarajan, P.D. Amer, R. Stewart, Corrections on: Improving file transfers using SCTP multistreaming, Originally published in the proceedings of IEEE IPCCC 2004 but the corrected version is available at www.cis.udel.edu/ amer/PEL/poc/.
[45] P. Natarajan, P.D. Amer, R. Stewart, Multistreamed Web Transport for Developing Regions, in: ACM SIGCOMM Workshop on Networked Systems for Developing Regions (NSDR), Seattle, 2008.
[46] L. Andrew, C. Marcondes, S. Floyd, L. Dunn, R. Guillier, W. Gang, L. Eggert, S. Ha, I. Rhee, Towards a Common TCP Evaluation Suite, in: Proceedings of the International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Manchester, United Kingdom, 2008.
[47] S. Floyd, V. Jacobson, Random Early Detection gateways for Congestion Avoidance, IEEE/ACM Transactions on Networking 1 (4) (1993) 397–413.
[48] S. Floyd, V. Jacobson, On Traffic Phase Effects in Packet-Switched Gateways, Internetworking: Research and Experience 3 (1992) 115–156.
[49] I. Psaras, V. Tsaoussidis, Why TCP Timers (still) Don't Work Well, Computer Networks Journal (COMNET), Elsevier Science 51 (2007) 2033–2048.
[50] L. Qiu, Y. Zhang, S. Keshav, Understanding the performance of many tcp flows, Computer Networks 37 (2001) 277–306.
[51] S. Shenker, L. Zhang, D.D. Clark, Some observations on the dynamics of a congestion control algorithm, ACM CCR (1990) 30–39.
[52] S. Floyd, RED: Discussions of Setting Parameters, www.icir.org/floyd/REDparameters.txt (November 2007).
[53] S. Floyd, Congestion Control Principles, RFC 2914, Internet Engineering Task Force (September 2000).
[54] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, Internet Engineering Task Force (June 1999).
[55] D. Ott, T. Sparks, K. Mayer-Patel, Aggregate Congestion Control for Distributed Multimedia Applications, in: IEEE INFOCOM, 2004.
[56] D. Damjanovic, M. Welzl, MulTFRC: Providing Weighted Fairness for Multimedia Applications (and others too!), SIGCOMM Comput. Commun. Rev. 319 (2009) 5–12.
[57] J. Crowcroft, P. Oechslin, Differentiated end-to-end Internet Services using a Weighted Proportional Fair Sharing TCP, SIGCOMM Comput. Commun. Rev. 28 (3) (1998) 53–69.
[58] M. Singh, P. Pradhan, P. Francis, MPAT: Aggregate TCP Congestion Management as a Building Block for Internet QoS, in: IEEE International Conference on Network Protocols, 2004, pp. 129–138.
[59] F.-C. Kuo, X. Fu, Probe-Aided MulTCP: an Aggregate Congestion Control Mechanism, SIGCOMM Comput. Commun. Rev. 38 (1) (2008) 17–28.
[60] S. Floyd, M. Handley, J. Padhye, J. Widmer, TCP Friendly Rate Control (TFRC): Protocol Specification, RFC 5348, Internet Engineering Task Force (September 2008).
[61] S. Floyd, M. Handley, J. Padhye, J. Widmer, Equation-Based Congestion Control for Unicast Applications, in: ACM SIGCOMM, 2000.
[62] D. Damjanovic, M. Welzl, M. Telek, W. Heiss, Extending the TCP Steady-State Throughput Equation for Parallel TCP Flows, Tech. Rep. DPS NSG Technical Report 2, University of Innsbruck, Institute of Computer Science, August 2008.
[63] Multipath TCP (MPTCP), trac.tools.ietf.org/area/tsv/trac/wiki/MultipathTcp.

[64] B. Briscoe, Flow Rate Fairness: Dismantling a Religion, SIGCOMM Computer Communication Review 37 (2) (2007) 63–74.
[65] F. Kelly, Charging and Rate Control for Elastic Traffic, European Transactions on Telecommunications 8 (1997) 33–37.
[66] S. Floyd, M. Allman, Comments on the Usefulness of Simple Best-Effort Traffic, RFC 5290, Internet Engineering Task Force (July 2008).
[67] M. Mathis, Rethinking the TCP-friendly Paradigm, IETF Internet-Draft: draft-mathis-iccrg-unfriendly-pre00 (work in progress) (December 2008).
[68] The Future of TCP: Train-wreck or Evolution?, yuba.stanford.edu/trainwreck.
[69] Clean Slate Design for the Internet, cleanslate.stanford.edu.
[70] International Conference Re-Architecting the Internet (ReArch), conferences.sigcomm.org/co-next/2009/workshops/rearch.
[71] NSF's FIND (Future Internet Design) Program, www.nets-find.net.
[72] Trilogy Project – Architecturing the Future Internet, www.trilogy-project.org.

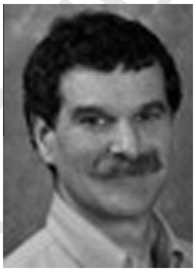
Ilknur Aydin is an Assistant Professor at the Computer Science department of SUNY Plattsburgh, NY, USA. Her research interests are in the area of wired, wireless, and mobile networking with a focus on transport layer issues. Dr. Aydin holds a Masters and Ph.D in Computer Science and Information Sciences from University of Delaware, Newark, DE, USA and received her B.S. in Computer Engineering from Marmara University, Istanbul, Turkey.

Janardhan Iyengar is an Assistant Professor at Franklin & Marshall College, an undergraduate liberal-arts college in Lancaster, Pennsylvania. Dr. Iyengar has worked in past on transport layer and end-to-end issues, and his current interests lie in Internet architecture, and in protocols and mechanisms to support increased flexibility and evolvability of the network.

Phillip T. Conrad is a faculty member in the Computer Science Department of the University of California, Santa Barbara, with a joint appointment in UCSB's College of Creative Studies. His job title is "Lecturer with Potential Security of Employment", a University of California designation that corresponds in rank to a tenure-track Assistant Professor, but with a focus on undergraduate education. He previously held faculty positions at Temple University and the University of Delaware. His research interests include computer science education, and computer networks, with a particular focus on transport protocols, and multimedia computing. Dr. Conrad's degrees include a Ph.D. in Computer Science from the University of Delaware, an M.S. in Computer Science from West Virginia University, and a B.S. in Computer Science from West Virginia Wesleyan College.



Chien-Chung Shen received his B.S. and M.S. degrees from National Chiao Tung University, Taiwan, and his Ph.D. degree from UCLA, all in computer science. He was a senior research scientist at Bellcore (now Telcordia) Applied Research working on control and management of broadband networks. He is now an associate professor in the Department of Computer and Information Sciences of the University of Delaware. His research interests include ad hoc and sensor networks, dynamic spectrum management, control and management of broadband networks, distributed object and peer-to-peer computing, and simulation. He is a recipient of NSF CAREER Award, and his research has been sponsored by NSF, NASA, Army Research Lab, RAND, and industrial companies. He is a member of both ACM and IEEE.



Paul D. Amer received the BS degree summa cum laude in Math from SUNY Albany in 1974, and the MS and PhD degrees in CIS in 1976 and 1979, respectively, from The Ohio State University. Since 1979, he has been at the University of Delaware where currently he is Alumni Distinguished Professor of Computer Science. Professor Amer's research focuses on innovative transport layer protocols such as SCTP, and data compression in multimedia.