OSIA: Out-of-order Scheduling for In-order Arriving in concurrent multi-path transfer

Jingyu Wang\textsuperscript{a}, Jianxin Liao\textsuperscript{a}, Tonghong Li\textsuperscript{b}

\textsuperscript{a} State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, PR China
\textsuperscript{b} Technical University of Madrid, Madrid 28660, Spain

Abstract

One major problem of concurrent multi-path transfer (CMT) scheme in multi-homed mobile networks is that the utilization of different paths with diverse delays may cause packet reordering among packets of the same flow. In the case of TCP-like, the reordering exacerbates the problem by bringing more timeouts and unnecessary retransmissions, which eventually degrades the throughput of connections considerably. To address this issue, we first propose an Out-of-order Scheduling for In-order Arriving (OSIA), which exploits the sending time discrepancy to preserve the in-order packet arrival. Then, we formulate the optimal traffic scheduling as a constrained optimization problem and derive its closed-form solution by our proposed progressive water-filling solution. We also present an implementation to enforce the optimal scheduling scheme using cascaded leaky buckets with multiple faucets, which provides simple guidelines on maximizing the utilization of aggregate bandwidth while decreasing the probability of triggering 3 dupACKs. Compared with previous work, the proposed scheme has lower computation complexity and can also provide the possibility for dynamic network adaptability and finer-grain load balancing. Simulation results show that our scheme significantly alleviates reordering and enhances transmission performance.

1. Introduction

Recent studies have suggested that concurrent multi-path transfer (CMT) (Phatak and Goff, 2002; Hsieh and Sivakumar, 2005; Lyengar et al., 2006; Liao et al., 2008) is an effective traffic engineering technique to improve throughput by aggregation of bandwidth, especially for the high-bandwidth applications such as video sharing/download/streaming. Other benefits include increased service reliability, latency reduction, and fault tolerance by sending redundant data over different paths, and enhanced mobility when combining coverage areas of multiple mobile and wireless access networks. Three key issues in the use of multiple paths in multi-homed mobile networks are: (1) how to maximize the aggregate throughput, (2) how to alleviate the reordering caused by the delay difference between any two paths, and (3) how to adjust to the variability of the throughput and delay, especially in wireless networks. In packet-switched networks such as the Internet, a packet is the smallest unit of data that can be transmitted over a network. In a multi-path network, a packet flow can be split over multiple paths between a source and a destination. Although multi-path communication can bring many benefits, it is also faced with the problem of “reordering”. That is, packets injected into the network later arrive at the destination before the packets injected into the network earlier. Unfortunately, current TCP protocol do not deal with this issue, leaving it instead to be misinterpreted as packet losses by the protocol stacks of these networks.

CMT needs schemes that can split traffic across multiple paths, while relieving the problem of “reordering”. Current traffic scheduling schemes, however, exhibit a tussle between the splitting granularity and the ability to avoid packet reordering. Packet-based scheduling quickly assigns the desired load share to each path. When paths differ in delay, however, a large number of packets can become out-of-order. TCP confuses this reordering as a sign of congestion, resulting in degraded performance of the applications (Laor and Gendel, 2002). Even for some UDP-based real time applications, such as video streaming or Voice-over-IP (VoIP), packet reordering may cause apparent loss of data and aggravate the buffer requirements at the receiver. Flow-level splitting, on the other hand, pins each flow to a specific path and thus packet reordering is avoided. But, flows differ widely in terms of their sizes and rates, and once assigned, a flow persists on the path throughout its lifetime (Papagiannaki et al., 2004; Zhang et al., 2002). Consequently, flow-level splitting may assign inaccurate amounts of traffic to each path or fail to quickly...
re-balance the load in the face of changing demands. The inability to quickly react to traffic spikes may congest links and reduce network effective throughput.

This paper shows that one can obtain the accuracy and responsiveness of flow-based splitting and still alleviate packet reordering. We introduce Out-of-order Scheduling for In-order Arriving (OSIA), a traffic scheduling mechanism. OSIA exploits a simple observation as shown in Fig. 1. Consider load balancing traffic over 3 paths in which path 1 is the slowest, and path 2 is the fastest. Here it is worth noting that this slow and fast refer to the measure in terms of delay (s) rather than bandwidth (bits/s). If the packets are scheduled and transmitted according to their original orders at the sender, the packets via path 1 will not arrive in the receiver on time (e.g., packet 2). As a result, the reordering takes place. To ensure the packets arrive at the intrinsic order, one can adjust the sending sequence of packets in such a way that packets at the slower path are transmitted earlier, while packets at the faster path are transmitted later. For example, in Fig. 1, packet 2 should be transmitted first via path 1, and then packet 3 is transmitted via path 3, finally the packet 1 is transmitted via path 2. Thus, the sending time discrepancy can bridge the gap between the faster and slower path. The faster path makes use of the packet queuing time to compensate the path propagation time in order to achieve the consistent arrival time.

Different from the traditional packet scheduling, OSIA disorders the original packet sequence. To be specific, the ready-to-transmit packets are out of order. During each interval, OSIA estimates the delays on these paths and partitions the flow into several chunks one-time, based on the delay difference between the parallel paths under consideration. Then the probabilistic scheduling is used to assign packets in each chunk to different paths and transmit them in parallel. The small size of packets enables OSIA to spilt traffic dynamically and accurately, ensuring that the resources of each path are fully utilized.

This paper makes the following contributions. (1) It introduces OSIA, showing that it is possible to transmit a TCP-like flow across multiple paths without causing packet reordering. In this paper, we take into consideration the characteristics of delay difference when designing load scheduling scheme. We propose an out-of-order scheduling scheme, which exploits the sending time discrepancy to preserve the in-order packet arrival. This scheme is suitable for the cases where the paths are dynamic, which is updated and executed periodically with the period interval.

(2) It formally analyses the flow splitting problem of OSIA. We assume that the source flow is regulated by a set of leaky buckets and uses a mixed strategy consisting of deterministic flow-level splitting and probabilistic packet-level scheduling to achieve a finer-grain parallelism while alleviate the out-of-ordering. We formulate a constrained optimization problem of maximizing total end-to-end aggregate throughput. We derive a closed-form solution by our proposed progressive water-filling (PWF) algorithm.

(3) It also presents an implementation to enforce the OSIA scheme using cascaded leaky buckets with multiple faucets, each bucket for each chunk and each faucet for each path, which provides simple guidelines on maximizing utilization of aggregate bandwidth and helps to guarantee the packets from multiple paths can arrive at the receiver in-order.

The rest of this paper is organized as follows. For ease of presentation, we first survey the related works in Section 2. Then, we discuss the analytical model, solution and process of OSIA in Section 3. In Section 4, we discuss implementation-related issues. Simulation results are shown in Section 5, and Section 6 concludes this paper.

2. Related work

Packet reordering (Mogul, 1992; Leung et al., 2007) is a phenomenon in which packets with higher sequence numbers are received earlier than those with smaller sequence numbers. It can be caused by a myriad of reasons. In Leung et al. (2007), five major causes are listed: packet-level multi-path routing, route fluttering, inherent parallelism in modern high-speed routers, link-layer retransmission, and router forwarding lulls. This paper focuses on alleviating the packet reordering caused by another new reason of the multi-path heterogeneity when performing CMT.

When packets from a single flow travel on multiple paths with different RTT, they may arrive reordered at the destination. Traditional TCP design is based on the assumptions of nearly in-order packet delivery and error-free transmission channel. Following these assumptions, three or more dupACKs (Allman et al., 1999) caused by any out-of-order packet events are misinterpreted as packet losses. As a result, the fast retransmission algorithm (Jacobson et al., 1992) is activated frequently to retransmit packets that have not been lost (referred to as false fast retransmit), which keeps window size unnecessarily small, and results in undesirable under-utilization of network bandwidth. Besides, persistent spurious retransmission can exacerbate network congestion, lead to classical congestion collapse (Allman et al., 1999), and reduce the TCP performance. Even for UDP-based application, this reordering can result in more demanded buffer space and extra application delays.

In this section, we survey the solutions proposed to date for packet reordering. We categorize the reordering solutions into three different classes, namely, (i) the class of approaches for distinguishing the events of packet reordering; (ii) the class of approaches for adjusting the triggering threshold; and (iii) the class of approaches for avoiding the occurrence of packet reordering.

The first class is a collection of methods that process the ordering information of segments and ACKs received, and then infer and generate more appropriate congestion response when the reordering events are detected. The TCP source finds out which segment or ACK has been reordered. It then reacts, say, by recovering previous congestion responses and/or disabling future congestion responses for a time period. The Eifel algorithm (Ludwig and Katz, 2000), DSACK TCP (Floyd et al., 2000), and TCP-DOOR (Wang and Zhang, 2002), belong to this class.

The second class is a group of techniques that avoid or delay triggering spurious congestion responses by deferring them for a time period. During the time period, the response will be canceled whenever it is inferred not to be caused by congestion. The response will be carried out only when the corresponding timer expires. The TCP source searches for an appropriate dupACK threshold to proactively avoid triggering a spurious fast retransmission and fast recovery as well as a retransmission timeout at

![Fig. 1. Load balancing traffic over three parallel paths in which the path 1 is the slowest and path 2 is the fastest. If the packets are scheduled and transmitted as their original orders, the packets via path 1 will arrive later at the receiver.](image-url)
the same time. The Leung-Ma Algorithm (Leung and Ma, 2005), Lee-Park-Choi algorithm (Lee et al., 2002), TCP-DCR (Bhandarkar and Reddy, 2004), and TCP-PR (Bohacek et al., 2006), belong to this class.

In contrast to the above two approaches just aiming to decrease the negative impact caused by packet reordering, the third class of approaches focuses on reducing the occurrence of packet reordering essentially. Currently, packet-based scheduling (Leung and Li, 2003) and packet resequencing (Lane and Nakao, 2010) are the most commonly used schemes to prevent packet reordering caused by the inherent multi-path routing. But they require additional techniques to preserve packet order, which has great impact on system performance (Arthur et al., 2007). Flow-level splitting allocates each flow to a specific path and avoids packet reordering, however it cannot assign accurate amount of traffic to each path. Furthermore, a sub-flow level (flowlet) scheduling has been proposed (Kandula et al., 2007) to tackle this problem with granularity, but it is not fine enough to quickly assign the desired load share to each path. In contrast, Kaspar et al. (2009) address and quantify the impact of packet reordering due to the multi-path heterogeneity when performing CMT, but their solution does not make full use of the network resources and cannot be suitable to the dynamic mobile networks.

In one of our earlier papers (Wang et al., 2008), we present a preliminary fragmentation strategy to avoid unnecessary retransmission timeout events, and focus on a one-time scheduling subproblem, as opposed to periodic scheduling. Besides, that strategy is imperfect and simulation is also insufficient. Consequently, it is desirable to design a more refined scheduling scheme to alleviate the inherent packet reordering caused by the multi-path heterogeneity when performing CMT, so that the packets can reach the destination almost in order. In addition, the traffic scheduling algorithm should have a low computation complexity and high network adaptability since mobile network conditions may change quickly.

3. OSIA scheduling

In this section, we first present our OSIA scheduling model to provide CMT, and formulate this scheduling problem as a constrained optimization problem of maximizing the aggregate throughput while contributing to that the packets are received in order with a high probability. To solve this problem, we then introduce a progressive water-filling algorithm and give out a two-step scheduling process.

3.1. Transmission and scheduling model

CMT uses the host’s multiple-interface feature to simultaneously transfer data across multiple end-to-end paths between the sender and the receiver. These independent network interfaces can be used effectively to transmit (or receive) packets independently, thus multiple paths are allowed to have packets “in flight” at all times over all interfaces. In most of CMT solutions (Iynghar et al., 2006; Liao et al., 2008), the traffic scheduling is based on the association (means the whole connection), and transmit–receive is performed on a per-path basis. Suppose that there exist three available paths 1, 2, and 3, which are depicted in Fig. 2, while path 1 is the faster one and path 3 is the slowest one. We use sub-flow i to indicate the sub-flow transmitted over path i. It is worth noting that in this model all sending (receiving) times are measured based on the exit (entrance) of packets from (to) the transport layer.

Based on the above model, if we split the flow into multiple chunks and schedule them in normal order over multiple paths with different delays, reordering is bound to occur as long as the path delay difference exists. Under the precondition of maximizing the utilization of the transmission resource of each path, the most promising solution to mitigate packet reordering is to adjust the sending order of packets, which is motivated by two fundamental observations: (i) packets transmitted via the faster paths will arrive sooner at the destination; and (ii) packets received simultaneously from different paths are bound to be transmitted at different times.

On the basis of above observations, instead of scheduling packets in-order, we propose an out-of-order scheduling scheme OSIA in which the packets injected into the faster path are to be transmitted later and the slower path earlier. OSIA exploits the sending time discrepancy to remove the path delay difference. We firstly use deterministic flow-level splitting to partition the traffic into several chunks. Then the packets of each chunk are assigned to different paths using probabilistic packet-level scheduling to achieve a finer-grain parallelism. Finally, the packets assigned to the same path are connected together to form a continuous sub-flow. Operating in this manner allows us to schedule packets among multiple paths before the actual transmission, which ensures that the congestion control of each path is not disturbed by the scheduling process.

As the network is time-varying, the CMT process of a connection in our model is divided into some variable-size periods, in each of which network performance can be relatively constant. Scheduling period x lasts t_x seconds, which is decided by the varying frequency of the path delay. With completion of period x, the period x+1 starts with updated transmission policy again.

3.2. Problem formulation

For any period x, we model the average available bandwidth of each path i as rate b_i. The contribution of all links and the propagation delay are aggregated into the path delay d_i. The ready traffic to be transmitted in [0, t_x] via path i is W_i = b_i t_x, which can arrive in [d_i, t_x + d_i] at the receiver. For most of the applications such as video sharing/download/streaming, video data are buffered at the local disk of the sender, which helps to guarantee that enough amounts of data can be obtained in advance and applied to mass scheduling and transmission.

For any set of paths S_{P_i}, with parameters \{b_i, d_i\}, i = 1, \ldots, M, we first do the following preprocessor.

1. Sort and relabel the paths according to their fixed delays d_i in non-decreasing order.
2. If there are k (k > 1) paths \{p_i, p_{i+1}, \ldots, p_{i+k-1}\}, with the same fixed delay, which is accurate to 0.01 s, i.e., d_i = d_{i+1} = \ldots = d_{i+k-1}, we can just aggregate these paths into a new path i with d_i = d_i and b_i = b_i + b_{i+1} + \ldots + b_{i+k-1} temporarily.
Relabel the paths, and then we get a new set of paths SP, with parameters \( \{d_i, b_i\}, i=1, \ldots, N \), and \( b_1 < b_2 < \cdots < b_N \).

In the following, we first determine the optimal partitioning scheme for the paths \( P_i, i=1, \ldots, N \). For each path \( P_i \) consisting of \( k \) paths with same delays, we can then partition its assigned chunk into \( k \) parts according to each path's average available bandwidth.

Assume there is no delay between the time when a packet is received and the time when the corresponding ACK is sent. From the perspective of receiver as shown in Fig. 3, the scheduling starts at time 0, we can observe that: (1) during period \([0, d_1]\), no packet can be received; (2) during period \([d_1, d_2]\), only packets from path 1 transmitted at \([0, d_2-d_1]\) period can be received; (3) during period \([d_2, d_3]\), the receiver can receive packets from path 1 transmitted at period \([d_2-d_1, d_3-d_1]\) and packets from path 2 transmitted at period \([0, d_3-d_2]\); and (4) during period \([d_h, d_{h+1}]\) \((1 < h < N)\), the receiver can receive packets from path \( i \) \((1 \leq i \leq h)\), which can be traced back to the sender transmitted at period \([d_h-d_i, d_{h+1}-d_i]\).

The following theorem establishes the essential condition to ensure the in-order arrival.

**Theorem 1.** The probabilistic scheduling ensures that the packets can arrive in order at receiver, if their sending times meet the following conditions: if the \( i \)-th path starts to send packets at time \( 0 \), then the \( i+1 \)-th path starts to send packets at \( d_i-d_i-1 \), the \( i+2 \)-th path at \( d_{i+1}-d_{i+1-2} \), and so on, and the 1st path begins at \( d_1-d_1 \). This can be represented as follows:

\[
ST_i = ST_{i-1} - (d_i-d_{i-1}) = ST_{i-2} - (d_i-d_{i-2}) = \cdots = ST_1 - (d_i-d_1) \tag{1}
\]

where \( ST_i \) denotes the starting time of \( i \)-th path.

However, if the packets are scheduled with the above probabilistic scheduling scheme via all \( N \) paths uniformly, the overall performance will be decreased down by the \( N \)-th path, which is the slowest one. In this case, a plentiful network bandwidth of faster paths is wasted, because the first packet can only be received at \( d_1 \) time. Fortunately, for most actual applications, the sender always has sufficient number of buffered packets. Thus, we can partition the buffered packets into \( N-1 \) chunks and send them in parallel to utilize the bandwidth of the faster paths. We use a probabilistic scheduling to evenly assign the packets of the \( h \)-th chunk to \( h \) paths \((1 \leq h \leq N-1)\). In the mean time, we stipulate that the assigned packets to the path \( i \) \((1 \leq i \leq h)\) be transmitted during period \([d_i-d_i, d_{i+1}-d_i]\), which ensures that the packets of the \( h \)-th chunk arrive at the receiver during the period \([d_h, d_{h+1}]\). In this way, the packets of all \( N-1 \) chunks are more likely to arrive in order at the receiver. For the remaining flow, the packet can be transmitted on all \( N \) paths with a probabilistic scheduling, in which the \( N \)-th path starts to send packets at time 0 and the \( i \)-th path starts to send packets at time \( d_i-d_i \) that is \( ST_i = (d_i-d_i) \).

Focusing on each path \( i \) solely, the packets in the \( h \)-th chunk should be transmitted \( d_h-d_i \) later than these in the \( h+1 \)-th chunk, which can be represented as follows:

\[
ST_i^h = ST_i^{h+1} - (d_h-d_i) \tag{2}
\]

where \( ST_i^h \) denotes the starting time of \( h \)-th chunk through \( i \)-th path.

**Corollary 1.** According to Theorem 1, we can infer a discipline that the time when the \( i \)-th path starts to send packets of the \( h \)-th chunk is the same as the time when the \( i+1 \)-th path starts to send packets of the \( h+1 \)-th chunk, that is:

\[
ST_i^h = ST_{i+1}^{h+1} \tag{3}
\]

Under the constraints as (1)-(3) of starting time, we can formulate an optimization problem of maximizing the aggregate throughput through adjusting the amount of data transmitted on each path with respect to each chunk.

For scheduling the \( h \)-th chunk, the path \( i \) can be assigned \( n_i \) packets each of which has the same size \( s_i \). The transmission duration of path \( i \) is \( n_i \cdot s_i \). The size of total transmitted traffic \( W \) is

\[
W = \sum_{i=1}^{N} \sum_{h=1}^{N} n_{ih} s_i \tag{4}
\]

Noting that the fraction of the size of the \( h \)-th chunk to \( W \) is

\[
f_i = \frac{\sum_{h=1}^{N} n_{ih} s_i}{W}. \tag{5}
\]

We can formulate the following linear constraint optimization problem for path set \( SP \), maximizing end-to-end aggregate throughput (AT) while preserving packets in-order (denoted as AT(\( SP \), \( t_s \)) as

\[
\text{Maximize : } AT = \frac{W}{t_s}
\]

subject to:

\[
\begin{align*}
ST_i^h &= ST_{i+1}^{h+1} - (d_h-d_i) \quad &d_0 = 0 \\
ST_i^h &= ST_{i+1}^{h+1} \quad &ST_h^h = 0 \\
\sum_{i=1}^{N} f_i &= 1 \\
8h + g_{2h} + \cdots + g_{gh} &= 1 \\
g_{2h} + \cdots + g_{gh} &= 1 \\
g_h &> 0 \\
& \quad h = 1, 2, \ldots, N-1 \\
& \quad i = 1, 2, \ldots, h
\end{align*}
\]

3.3. Progressive water filling solution

With the aid of an intuitive “water-filling” model (Mao et al., 2005), we propose a “Progressive Water-Filling” (PWF) algorithm for solving the above optimization problem AT(\( SP \), \( t_s \)) directly. A multi-path network (\( N \) paths) and an amount of packets are described in Fig. 4, we model each path \( i \) as a bucket with an area of cross section \( b_i \). In addition, each bucket \( i \) is pre-loaded with content \( b_id_i \) to a level \( d_i \). Assume each bucket has a finite depth \( L^* > d_i \) which is the highest pre-loaded level of the \( N \) buckets. The most units of fluid held by the \( N \) buckets is just the value of \( W \) in the problem AT(\( SP \), \( t_s \)) where \( t_s = L^* \).

We try to “fill” all the buckets, until all buckets reach their capacity limits. If path \( i \) is assigned with a portion \( p_i \), this is equivalent to filling \( p_i \) of \( W \) units of fluid into bucket \( i \). With this model, the optimization problem is equivalent to maximizing \( W \), the total units of fluid filled into the \( N \) buckets.

In this case, let each bucket have a finite depth as shown in Fig. 4. In order to maximize \( W \), the WP units of fluid should be distributed to the \( N \) buckets in such a manner that all the buckets have the same rated fluid level \( L_0 \). This means

\[
L_1 = L_2 = \cdots = L_N = L^* \tag{6}
\]
The fraction of the $h$th chunk filled into $i$ buckets, is

$$f_h = \left\{ \begin{array}{ll}
\frac{\sum_{k=0}^{n_h} b_i (d_{i+h} - d_k)}{\sum_{k=0}^{n_h} b_i (d_{i+h} - d_k)} & 1 \leq h \leq N-1
\end{array} \right. \quad (13)$$

In other words, for each chunk, the probability that a packet is transmitted along a given path corresponds to the ratio of its bandwidth to the total available bandwidth of all involved paths, which might be simply validated intuitively. The traditional scheduling scheme is always preferential to use the path with a higher bandwidth and a lower fixed delay, but it is impossible to order the paths consistently in many cases. A brute force optimization evaluation of all feasible path combinations would have exponential complexity (Tsirigos and Haas, 2004). Using our approach, the scheduling scheme can be easily performed with $O(N^2)$ complexity, where $N$ is the number of paths available. As a consequence, this scheme can achieve a good balance of workload and high system throughput. The implementation of traffic scheduling will be discussed in Section 4.

3.4. Scheduling process

The OSIA scheduling scheme is designed as a mixed strategy, which consists of two steps: a deterministic splitting step, based on flow level striping; followed by a probabilistic assigning step, based on packet level scheduling.

1. The first step is the key to solving the reordering problem, which uses the PWF algorithm to partition the original data flow into multiple chunks. To ensure in-order delivery, OSIA exploits the sending time discrepancy to remove the path delay difference. The original flow is systematically partitioned into several chunks in this step. An example of three paths is shown by Fig. 5. The first two marking points at "actual data" axis represent the partitioned borders, whose positions are deterministically decided by the PWF algorithm, thus the first and the second chunk can be constructed in advance. However, the last marking point cannot be determined beforehand, which represents the ending position of the flow to be transmitted in the current period. In fact, the third chunk consists of the remaining part of the flow to be transmitted in this period.

2. The second step assigns the packets of each chunk to the target paths in terms of their probabilistic portions in parallel. As for each chunk, we refer to a bandwidth-aware probabilistic scheme to schedule the packets across multiple paths with different delays. This packet-level scheduling scheme achieves a finer-grain parallelism and tries to alleviate the packet reordering. In Fig. 5, the mapping of "actual data" axis to "transmitted" axis illustrates this assigning process. Figure 5(a)-(c) stand for the assignments of chunks 1-3, respectively. Moreover, the corresponding relation between "transmitted" axis and "received" axis shows that the CMT process preserves the in-order arrival of packets at the receiver.

In summary, this mixed scheme can alleviate the inherent packet reordering caused by the multi-path heterogeneity when performing CMT. It also has a low computation complexity. As the TCP RTO estimator is relatively conservative, the CMT using this mixed scheduling scheme can also decrease the unnecessary timeout retransmission. We have the following theorem.

**Theorem 2.** If (14) is satisfied, there is no timeout caused by the path delay difference.

$$\sum_{m=1}^{h} \varepsilon_d (\tau_m - r_m) < K \sum_{m=1}^{h} \varepsilon_d (\tau_m - r_m) \quad (14)$$
Then, the remaining flow is redirected to the second leaky bucket with its ratio computed as \(g_{hi} = b_1(d_1 - d_i)/\sum_{i=1}^{h-1} b_i(d_N - d_i)\). Then, the remaining flow is redirected to the second leaky bucket with its ratio computed as \(f_2 = b_2(d_2 - d_1)/\sum_{i=1}^{h-1} b_i(d_N - d_i)\), while the probability that a packet is assigned to path 2 is \(g_{22} = b_1/(b_1 + b_2)\) and \(g_{22} = b_2/(b_1 + b_2)\), respectively. The remaining flow is redirected to the 3rd,... \(h\)th ..., \(N\)th leaky bucket successively, in which the \(h\)th leaky bucket has \(h\) faucets. At the \(N\)th bucket, there is no need for splitting the flow and the probabilistic scheduling is used to leak the subsequent flow. It is worth noting that there always exist tokens for the incoming traffic. Consequently, the above deterministic partitioning scheme does not introduce additional loss or delay to the application data. The varying path conditions could trigger the updating of the leaky bucket parameters.

4.2. Path parameter estimation

The proposed scheme works best when some QoS supports are available in the network. For example, with the support of the resource reservation protocol (RSVP) (Zhang et al., 1993), a source can reserve the required bandwidth along each path, and a router or a switch can use the generalized processor sharing (GPS) scheduling to guarantee the reserved bandwidth (Zhang, 1997). If such QoS provisioning mechanisms are not available, the receiver could estimate the path parameters, i.e., \(b_i\) and \(d_i\), for a snapshot of the network and send the estimates back to the source, if the path conditions vary at a relatively large time scale.

Estimating path parameters based on end-to-end measurements has been an active research area for many years. Many effective techniques (Jain and Dovrolis, 2003; Kapoor et al., 2004; Gurewitz and Sidi, 2001) can be applied to estimate the path parameters in our approach. For example, the SLoPS (Jain and Dovrolis, 2003) and CapProbe (Kapoor et al., 2004) can be used to estimate the...
Fig. 7. Number of 3 dupACKs events triggered by various scheduling with respect to the delay (i.e. D2/D1) and bandwidth (i.e. BW2/BW1) ratio. (a) Round-Robin scheduling, (b) CWND-aware scheduling, (c) Flow-level scheduling, (d) OSIA scheduling.

end-to-end available bandwidth (or bottleneck bandwidth) of a path. If the source and the receiver are synchronized, the minimum one-way packet delay measured in the last time window would be a good approximation of the fixed delay $d$ on that path. Otherwise, the approach presented in Gurewitz and Sidi (2001) can be used to estimate the one-way delay from cyclic-path delay measurements. Therefore the OSIA depends upon the assumption of the availability of path bandwidth and delay, at least which can be obtained. Moreover, this OSIA has a certain tolerance for the inaccurate measurement result of bandwidth and delay according to Theorem 2. For the sake of convenience, our following simulation experiments are all arranged in controlled environments where all metrics can be preset, after all the technology of network measurement does not belong to the innovation of this paper.

After obtaining the path parameters, the Stream Control Transport Protocol (SCTP) (Stewart, 2007) and its CMT extension (Iyengar et al., 2006; Liao et al., 2008) can be used for delivering the parameters to the sender via the SACKs of each path. The sender then computes the optimal partition and updates the parameters of the leaky buckets periodically. Note that path conditions could change because of path failure, rerouting, etc. Furthermore, the variation of cross traffic load using the same paths may cause variations of the estimated path parameters and trigger updating of the leaky bucket parameters. If the congestion occurs at a relative large time scale, the proposed traffic partitioning scheme can adapt to the congestion as well.

5. Evaluation and numerical results

This section presents the simulation model used to evaluate the performance of the proposed method and then provides the experimental results.

5.1. Simulation model

For the purpose of simulation, we implemented the cmpSCTP (Liao et al., 2008) protocol to support the CMT in OPNET 10.0.A (OPNET simulator, 2005). In our simulation, we use a simple topology with two or more parallel paths between sender and receiver. A real traffic trace (the "Star Wars" movie) is used as a source of FTP foreground traffic. To simulate paths with different available bandwidths in the dynamic network, cross-traffic is introduced. Our cross-traffic in each path is generated according to a Pareto process with an on-off period that takes value in the range [10 ms, 1 s]. The bit error rate on a link dynamically changes within the range between $1 \times 10^{-3}$ to $1 \times 10^{-5}$ (with the average of $2 \times 10^{-4}$), and all links have the same bit error rate. The path Maximum Transmission Unit (MTU) at each path is 1 Kbytes, and the packet size is 1 kb.

We compare our proposed OSIA scheduling scheme against three different methods in our experiments: (i) traditional Round-Robin scheduling scheme (Hahne and Gallager, 1986); (ii) CWND-aware scheduling scheme (Saadawi and Lee, 2004); (iii) flow-level scheduling scheme. Clearly, the Round Robin approach suffers
from the excessive packet reordering and is not recommended in practice. The CWND-aware scheduling is based on bandwidth-delay product, in which the outgoing traffic is divided over the multiple paths according to the ratio of their CWNDs (i.e. in each point of the time, the possibility of the path $i$ serving the packet is $\frac{\text{CWND}_i}{\sum \text{CWND}_i}$). The flow-level scheduling stripes the flow adaptively based on the path bandwidth. The scheduling periods of all schemes are set as 1 s. We use the number of fast retransmission events and the term "goodput" as the performance metric of evaluating these methods, in which the "goodput" is the application level throughput and measures the rate of the packets reaching their destination successfully, excluding retransmitted packets.

5.2. Number of 3 dupACKs

To evaluate the impact of various network parameters on the performance of our approach, we consider a simple topology with only two parallel paths, and vary the following network parameters in the simulations: (i) the delay of the paths; and (ii) the bandwidth of the paths. For the sake of simplicity, we assume that the average delay and the bandwidth of path 1 are fixed to 50 ms and 1 Mbps, respectively. The relative delay and bandwidth of path 2 with respect of path 1 are varied in this experiment.

Figure 7(a) shows the number of 3 dupACKs with the Round-Robin scheduling, in which multiple paths are served in a Round-Robin manner with one packet transmitted to one available path in a service round. Although round robin data is simple, it causes a lot of 3 dupACKs events when the paths have different characteristics, while the number of 3 dupACKs events is nearly independent of the delays of the paths. The number of 3 dupACKs with CWND-aware scheduling is illustrated in Fig. 7(b). As can be observed, the CWND-aware scheduling can lessen the occurring of 3 dupACKs, which exploits the knowledge of the CWND. Finally, comparing the simulation results of Fig. 7(c) and (d), we can see that the performance of flow-level and OSIA scheme are comparable. Clearly, the packet reordering events are considerably prevented by our OSIA scheduling scheme, which fully exploits the available resources.

5.3. Achieved goodput

We use the above simulation topology and configuration, and only change the number of the packets to simulate the scenario where the sender transmits 500 packets through two paths. Figure 8 shows the goodput obtained by different methods for transferring a file of size 500 kb. According to Fig. 8(a), the aggregate goodput obtained by Round-Robin scheduling scheme (Saadawi and Lee, 2004) increases much slow, due to a mass number of unnecessary fast retransmissions. From Fig. 8(b), CWND-aware scheduling can increase the goodput, with the increase of CWND of the auxiliary path 2. However, it reduces the end-to-end goodput when the paths have different delays.

Figure 8(c) shows that the poor accuracy of flow splitting in the flow-level scheduling can hurt the network goodput. Flow-
level scheduling cannot divert the ongoing flows away from the congested path. This can congest the overloaded path, resulting in a high drop rate and a low overall goodput. Our scheme shown in Fig. 8(d), on the other hand, reacts quickly and diverts the ongoing flows away from the congested path. It is worth noting here that our OSIA scheduling scheme succeeds in splitting a TCP-like flow among multiple paths with different available bandwidth and delay.

5.4. Effect of path numbers

To validate the effect of number of paths on the performance of our approach, we run a set of simulations with two nodes connected by two, three, four, and five paths. The paths configuration is described in Table 1. In all the considered configurations, the overall bandwidth, summing the bandwidths of all paths, is equal to 4 Mbps. In the following, each result is an average of multiple simulation runs under the same set of parameters.

Our simulation results, detailed below, show that: OSIA interacts benignly with path delay and bandwidth difference. In particular, splitting a TCP-like flow across multiple paths using OSIA has no negative impact on the aggregate goodput. On the other hand, packet-based splitting reduces the end-to-end goodput when the multiple paths have different delays, whereas flow-level scheduling reduces the end-to-end goodput when the network condition is varied.

Figure 9 shows the effect of reordering on goodput as the number of paths increases. According to Fig. 9, the aggregate goodput obtained by Round-Robin scheduling scheme is the smallest. With the increase of number of paths, the goodput gained by CWND-aware and Flow-level scheduling gets reduced, due to a mass number of unnecessary fast retransmissions and the inefficient congestion adjustment strategies. Moreover, we observe that the goodput of PWF algorithm degrades gracefully when the difference of paths increases.

Two conclusions can be drawn from Fig. 9. First, out-of-ordering caused by packet-based splitting can hurt the goodput significantly. Second, PWF algorithms can split the same flow among multiple paths whose one-way delays are different by as much as 80 ms, without hurting the overall goodput. This is because it tends to exploit the delay difference of these paths and hence, the carefully pre-arranged out-of-order sending sequence ends up with an in-order arrival at receiver.

6. Conclusion and future work

CMT allows utilizing multiple paths to transmit packet in parallel between a source and destination. However, how to schedule packets on these paths is an important issue to be solved. Transmitting packets from a single flow on multiple paths with different RTTs can lead to out-of-order packet arrival and hence TCP-like performance degradation. Our objective is to maximize the aggregate throughput while decreasing the probability of packet reordering caused by the delay difference.

This study aims at contributing to the reasonable traffic scheduling of CMT in multi-homed mobile networks. An analytical model of OSIA is presented firstly. Then, we present a novel approach to alleviate the potential reordering events due to inherent delay difference. We also provide a practical implementation to enforce the optimal scheduling on each path with negligible computation overhead. There are several issues, which are worthy to be further discussed: (1) the development of an integrated solution for other types of reordering; (2) the complete mechanism considering more network parameters such as losses, bandwidth fluctuations, delay jitters, etc.; (3) the development of an improved retransmission timeout estimator for CMT; and

<table>
<thead>
<tr>
<th>Number of paths</th>
<th>Path id</th>
<th>Bandwidth (Mbps)</th>
<th>Propagation delay (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>1.25</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.75</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1.25</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.75</td>
<td>80</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 9. Goodput with the number of paths to validate the different effects produced by various scheduling.

Table 1
Bandwidth/delay settings for multi-path scenario.
(4) the investigation about how the performance of our proposed scheme scales with network size.

Appendix A

Consider this situation: packets are sent out simultaneously via $N$ distinct paths with different RTTs, and packets are not lost in transmission. Here, we only consider the scenario where timeout happens only if the RTTs of two paths are substantially different, without consideration of the timeout caused by the packet sequence. If the difference of RTTs between two paths is big and the RTT of slow path is larger than the timeout value, unnecessary retransmission will become very common and transmission efficiency will be greatly degraded.

In popular versions of TCP (Jacobson et al., 1992), the retransmission timeout (RTO) for the $i$th packet is set as

$$RTO_i = R_i + KV_i$$

where $R_i$ is the current smoothed estimate of RTT, $V_i$ is the current smoothed estimate of the deviation in RTT, and $K$ is a constant factor (typically $K=4$), which adjusts the measured averaged retransmission timeout with respect to its variance (Phatak and Goff, 2002). $R_i$ and $V_i$ are defined by the following recursive equations:

$$R_i = \alpha R_{i-1} + (1-\alpha)RTT_i$$

$$V_i = \beta V_{i-1} + (1-\beta)|RTT_i - R_i|$$

where $RTT_i$ is the sampled RTT of the $i$th packet. Eqs. (16) and (17) can be acted as a low-pass filter on the sampled RTTs to smooth out the variations.

According to the above discussion, the exact value of RTO depends on the sequence of RTT samples. In our multi-path scenario, it depends on the sequence of paths chosen to send packets. As an approximation, the average RTT and average deviation in RTT will be considered to compute RTO.

Logically, the time needed to receive a packet from an arbitrary path is a function of packet size, path bandwidth and delay of that path. More specifically, inspired from Demichelis and Chimento (2002), we use $d_i$ to denote the one-way delay of $i$th path. Based on this delay, the RTT of a packet (with size $s_i$) sent along path $i$ (with bandwidth $b_i$) can be computed by

$$r_i = \frac{s_i}{b_i} + 2d_i$$

The average RTT of any chunk $h$ is then

$$\bar{r} = \sum_{i=1}^{h} r_i g_i$$

in which $r_i$ is the smoothed estimate RTT for packets sent along path $i$ and $g_i$ is the portion of traffic sent over path $i$ as (14). Similarly, the average deviation in RTT is

$$\bar{v} = \sum_{i=1}^{h} p_i |r_i - \bar{r}| = \sum_{i=1}^{h} p_i \sum_{j=1}^{h} p_j (r_i - r_j)$$

From (15), no timeout will occur if

$$r_m < \bar{r} + KV_i \quad \forall m, \quad 1 \leq m \leq h$$

Considering (19) and (20), the inequality of (21) can be restated as

$$\frac{\sum_{i=1}^{h} g_i r_i}{\sum_{i=1}^{h} g_i} < K \quad \forall m, \quad 1 \leq m \leq h$$

Hence, if the TCP implementation is configurable, we can appropriately set the value of $K$ to satisfy the above inequality. Otherwise, the sender needs to carefully adjust the value of $g_i$ (i.e. the ratio of data sent over path $i$). As the above equation implicitly states, the condition for preventing timeout depends solely on the difference of paths' delays, rather than on their absolute values.

The timeout event cannot occur in the period of the first chunk due to only single path transmission. In the following, we analyze the periods of other chunks.

A.1. Period of the second chunk

Firstly let us consider the period of the second chunk in which merely two paths are exploited to carry the traffic, meaning that $h=2$. We assume that $r_1 < r_2$. (22) is always satisfied for $r_1 = r_2$. For $m=1$, (22) is equivalent to

$$\frac{g_{12}(r_1 - r_2)}{g_{12}(r_1 - r_2) + g_{22}(r_2 - r_1)} < K$$

which is always satisfied, because the nominator of that fraction should be negative.

For $m=2$ in (22), we have

$$\frac{g_{12}(r_2 - r_1)}{g_{12}(r_1 - r_2) + g_{22}(r_2 - r_1)} < K$$

By taking into account $r_2 - r_1 > 0$, (24) is shortened to

$$1 < g_{22} < \frac{1}{2K}$$

In standard TCP, the value of $K$ is typically set to 4 (Jacobson et al., 1992; Phatak and Goff, 2002). It forces us to set the value of $g_{22}$ greater than 0.125, that is $b_1 < 7b_2$. In other words, if only the bandwidth over the faster path is less than 7 times of the slower path, retransmission timeouts will not occur regardless of the delay of two paths.

A.2. Period of the third chunk

Secondly let us analyze the period of the third chunk, meaning that $h=3$. Remind that according to our model, we assume that $r_1 < r_2 < r_3$, and (21) is always satisfied for $r_1 = r_2 = r_3$. By setting $m=3$, (22) is equivalent to

$$\frac{g_{13}(r_3 - r_1) + g_{23}(r_3 - r_2)}{g_{13}(r_1 - r_2) + g_{23}(r_1 - r_3) + g_{33}(r_1 - r_3) + g_{33}(r_2 - r_3)} < K$$
If (26) holds true, timeout cannot occur in all the paths, since
$r_3$ is the delay in the slowest path.
If $g_{13}(r_2-r_1)+g_{23}(r_2-r_3) > 0$, by taking into account

$$g_{13} + g_{23} = 1$$

(27)

is shortened to

$$
\frac{r_3 + (g_{13} r_2 + g_{23} r_3)}{g_{13} + g_{23} + g_{33}} < 2K
$$

(28)

which is also the same as the instance of $r_1 = r_2 = r_3$.

If $g_{13}(r_2-r_1)+g_{23}(r_2-r_3) < 0$, similar to above simplification, we can reduce (26) to

$$\frac{1}{2K} < g_{33} < 1$$

(29)

which is also the same with the instance of $r_1 = r_2 = r_3$.

Considering (13) and (15), the inequality of (14) can be restated as

$$g_{23} < (1 - g_{33})\frac{2Kg_{33}(r_2-r_1)-r_1}{(r_2-r_1)}$$

(30)

$$g_{13} > (1 - g_{33})\frac{r_3-r_2}{2Kg_{33}(r_2-r_1)+r_1}$$

(31)

Similar to the case of two paths, (29) shows that if the portion of traffic over the fastest path is greater than 0.125 in the case of the third chunk, and the value of $g_{23}$ or $g_{13}$ meets the constraints of (30) or (31), then no timeouts will occur.

References