TKN Telecommunication Networks Group

Technical University Berlin

Telecommunication Networks Group

A Splitter/Combiner Architecture for TCP over Multiple Paths

Tacettin Ayar, Berthold Rathke
and
Łukasz Budzisz

{ayar,rathke,budzisz}@tkn.tu-berlin.de

Berlin, February 2012

TKN Technical Report TKN-12-001

TKN Technical Reports Series

Editor: Prof. Dr.-Ing. Adam Wolisz

Contents

List of Figures

List of Tables

Chapter 1

Introduction

TCP is the prevalent reliable transport layer protocol used in the Internet to carry user data. Usually, a single path between TCP sender and receiver is used to transport TCP traffic. In such a case, TCP throughput is limited to the capacity of the bottleneck link on that path. Instead, if multiple paths are used simultaneously to aggregate the bandwidth the TCP performance may increase.

It has been already shown using TCP fluid model [24, 27] that the potential of the multipath solution lies not only in providing robustness but also, in conjunction with an appropriate congestion controller, in providing means to balance the Internet congestion in a stable way.

Despite these potential benefits and a large body of work [15, 18, 19, 22, 23, 31, 33, 34, 38, 41], several issues concerning multipath transport of TCP remain to be addressed before it can be succesfully deployed. These include (i)easy activation/deactivation of the solution on the end-hosts, (ii)opportunity to place the solution on network elements in-between the TCP sender and receiver (i.e., deployment of the solution must not be limited to the end-hosts), and (iii)independency of operation from the operating systems (OSs) used on end-hosts.

First issue is to develop a solution that does not change TCP/IP protocol stack implementation of the end hosts. To expect OS vendors to change their TCP/IP protocol stack implementations to include the solution is for us a little bit unrealistic. Indeed, TCP variants proposed till today shows the infeasibilty of that expectation. Instead, we offer a solution that is pluggable to the host. When users want to benefit from the solution, they may install the solution and use it.

Second of the mentioned limitations of the TCP over multiple paths solutions is the freedom of its use from the TCP end points. Deployment and adoption of the solutions that are based on the end-host support is usually a big problem [34, 38]. Indeed, to the best of our knowledge, there is no TCP over multiple paths solution without end-host support (i.e., TCP sender and receiver sides are not aware of its presence).

Last point in achieving high deployment of the multipath TCP solutions is to test the implementation with real hosts running different OSs. Thus, end users using different OSs may benefit from the proposed solution.

Chapter 2

State of The Art: TCP Problems over Mulitiple Paths and The Approaches To Handle Them

TCP design assumes that packets of a TCP flow follows a single path. When TCP data packets are distributed over multiple paths, a couple of issues must be considered. In this section, we list these issues and the solutions offered by the researchers.

2.1 Preventing/Handling Out-of-Order Packet Receptions

Since multiple paths may have different delays, packets scheduled for shorter delay paths may arrive at the receiver before the packets scheduled to the paths with longer delays.

TCP performance suffers from out-of-order packets in different ways, as described in [28]. Among these, the most important one is that they will cause generation of duplicate acknowledgements (DUPACKs). When TCP receiver gets a packet out-of-order, it generates a DUPACK. Reception of three DUPACKs cause TCP sender to unnecessairly trigger the fast retransmission/recovery algorithms [36].

In order to minimize the impact of the out-of-order deliveries in multi-path networks, solutions proposed so far (Figure 2.1):

1. *use delay estimations of the paths to minimize the number of out-of-order packet receptions:*

    - BAG (Bandwidth Aggregation) proxy [22] uses PET (Packet Pair based Earliest-Delivery-Path-First (EDPF [13])) scheduling algorithm to distribute packets to multiple paths. Based on path delay estimations, EDPF algorithm selects the path that will earliest deliver the packet to the client.
    - Simula Proxy [34] uses path delay estimations to calculate packet arrival times and buffer the potentially out-of-order packets on the shorter paths, compensating the different path delays.

2. *provide receiver-side buffering and reordering:* Even if the solutions try to minimize out-of-order packet arrivals at the TCP receiver, it is most of the time inevitable since scheduling algorithms are based on the delay estimations, which may not be accurate because of the variations in the network. Thus, buffering and reordering of the out-of-order packets may be necessary before they reach the TCP receiver. A component on the receiver host is generally used in that case to buffer and reorder out-of-order packets before passing them to the TCP receiver entity [22, 33].

   How long packets will be buffered if there is a gap in packet sequence numbers received so far? The missing packet may be lost or on the way. BAG [22] receiver side component defines two buffer management policies (BMP) to answer that question:

   - *comparison-based BMP:* if a sequence number is missed and packets with higher sequence numbers are received from all the paths, then a packet loss is inferred. There is no need to wait for the missed packet.
   - *timer-based BMP*: A timer is started for each packet reception. If the timer expires before the packet is received, then the packet loss is inferred. Buffered packets are sent to the TCP receiver so that the DUPACK generation may be started. BAG uses comparison based BMP as the main algorithm and timer-based BMP as a backup with constant timer value of 500 ms.

     Horizon [33] uses the timer-based BMP with an adaptive timer. Since it is possible for TCP sender to timeout if a path has a delay very larger than the other paths, Horizon keeps its own estimate of one-way propagation delay. Mean skewed one-way propagation delay and its variance is estimated using the same weighted moving average algorithm as in TCP round-trip time (RTT) and retransmission timeout (RTO) estimation. If a sequence gap is not filled after this estimated delay, then a buffered packet is delivered to the TCP receiver so that it may generate a DUPACK, which prevents TCP sender from timeout.

3. *extend TCP receiver side sliding window mechanism to cover multiple paths:*

   - TCP-PARIS (PArallel download protocol for ReplIcaS) [23] uses its own TCP-PARIS receiver instead of standard TCP receiver. TCP-PARIS downloads different portions of a replicated file from different servers. In case a file is replicated on n servers, a TCP-PARIS receiver makes n sub-connections to all of these servers. A TCP-PARIS flow is defined as a combination of all these sub-connections.

     TCP-PARIS uses a receiver-side sliding window mechanism for flow control. In contrast to TCP, TCP-PARIS spans the sliding window over multiple TCP sub-connections. Minimum sequence number of the on-the-fly segments is the left boundary of the window and specifies the smallest missing sequence number (SMS). SMS is updated when the smallest missing segments of all connections are received. The right boundary of the window, largest assigned sequence number (LAS), is used for new segment-to-server assignments and assures that a segment is not assigned to more than one server.

- MPTCP uses two sequence numbers for the packets: (1)connection level data sequence numbers (DSNs) and (2)subflow[1] level sequence numbers. Each subflow has its own sequence number space that is mapped to DSNs. If a packet is retransmitted via another subflow, then it uses the same DSN mapped to a different subflow level sequence number. MPTCP buffers and orders packets from different subflows based on their DSNs. pTCP [19] also uses a similar method to map sequence numbers of the connection data packets to sub-connection (i.e., TCP-v) sequence numbers.

4. *allow out-of-order packet arrivals, but eliminate its side effects:*

   - TCP uses the number of DUPACKs as a sign for congestion and therefore reduces its sending rate. In a multiple path environment DUPACKs may not be a sign of congestion; it may just indicate packet reordering caused by different delays of paths. Therefore, default TCP fast retransmission threshold (dupthresh) value (i.e., 3) may not be suitable for multiple path environments and leads to throughput degradation. By means of computer simulations, authors investigate a relation between the number of paths used and dupthresh value in [15]. They propose to increase dupthresh value logarithmically based on the number of paths used.
   - PRISM [31] tracks the state of the receiver buffer by processing SACK packets from the receiver to detect out-of-order packet receptions and packet losses. When a DUPACK is received, it is processed and buffered. By comparing following ACKs with the stored ACKs, PRISM reorders SACKs and modifies their cumulative ACK numbers to hide reordered packets to the TCP sender. If a new ACK is received which covers buffered ACKs, then the buffered ACKs are sent to the TCP sender. If a packet loss is detected, then PRISM sends the buffered SACKs immediately, without changing their cumulative ACK numbers.
   - DEF (DUPACK Estimation and Filtering, Chapter 5) uses path delays to estimate number of out-of-order packet arrivals at the receiver side. It filters the excessive DUPACKs to prevent TCP sender from the unnecessary fast retransmit/recovery.

5. *allow out-of-order packet arrivals without eliminating its side effects:* Multiple paths may be used for redundancy concerns. If the available paths have high packet loss rates, then the same copy of the packets may be sent via the default path as well as via the backup path(s). Some of the packets will be dropped and some will arrive at the receiver (possibly) out-of-order. As an example, MultiPath TCP [18] has been developed for mobile ad-hoc networks (MANETs) which have high error rates. MultiPath TCP uses two disjoint paths to send copies of the same TCP segment to create redundancy. It doesn't process the DUPACKs generated since the packet loss rate is high.

[1]MPTCP calls sub-connections as subflows

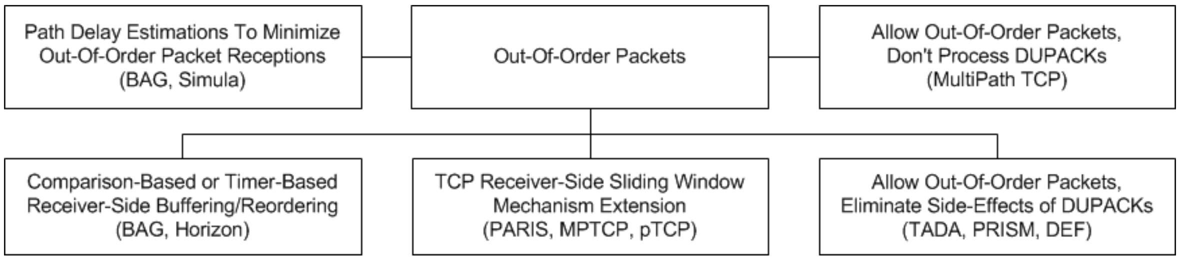

Path Delay Estimations To Minimize Out-Of-Order Packet Receptions (BAG, Simula)
Out-Of-Order Packets
Allow Out-Of-Order Packets, Don't Process DUPACKs (MultiPath TCP)
Comparison-Based or Timer-Based Receiver-Side Buffering/Reordering (BAG, Horizon)
TCP Receiver-Side Sliding Window Mechanism Extension (PARIS, MPTCP, pTCP)
Allow Out-Of-Order Packets, Eliminate Side-Effects of DUPACKs (TADA, PRISM, DEF)

Figure 2.1: Out-of-Order Packets Solutions

2.2 Management of The list of Multiple Paths

Path list management includes two steps (Figure 2.2):

1. *The list of usable multiple paths must be found during connection setup:* The following methods are used to detect the available multiple paths:

    - Active (i.e., connected to a network and may be used to send/receive data) interfaces of the multi-homed hosts are used (e.g., [19, 22, 31, 34, 41]): a packet forwarded to/from each interface follows a different path.
    - MultiPath TCP uses a multipath routing protocol (i.e., Split Multipath Routing (SMR) [10]) to get the list of routes that may be used to send data.
    - TCP-PARIS receiver is supplied with the addresses of the file servers where the file which will be downloaded is replicated. TCP-PARIS receiver establishes a TCP sub-connection with each of the servers and data flows via the paths between the TCP-PARIS receiver host and the server hosts.

2. *Path list must be updated during the connection time since some new paths may become available (e.g., by means of activation of an idle interface) or some paths may become unusable:*

    - MPTCP allows addition/deletion of paths during the connection by means of ADD_ADDR and REMOVE_ADDR TCP options.
    - pTCP uses if-up and if-down messages to indicate that an interface became active/passive.

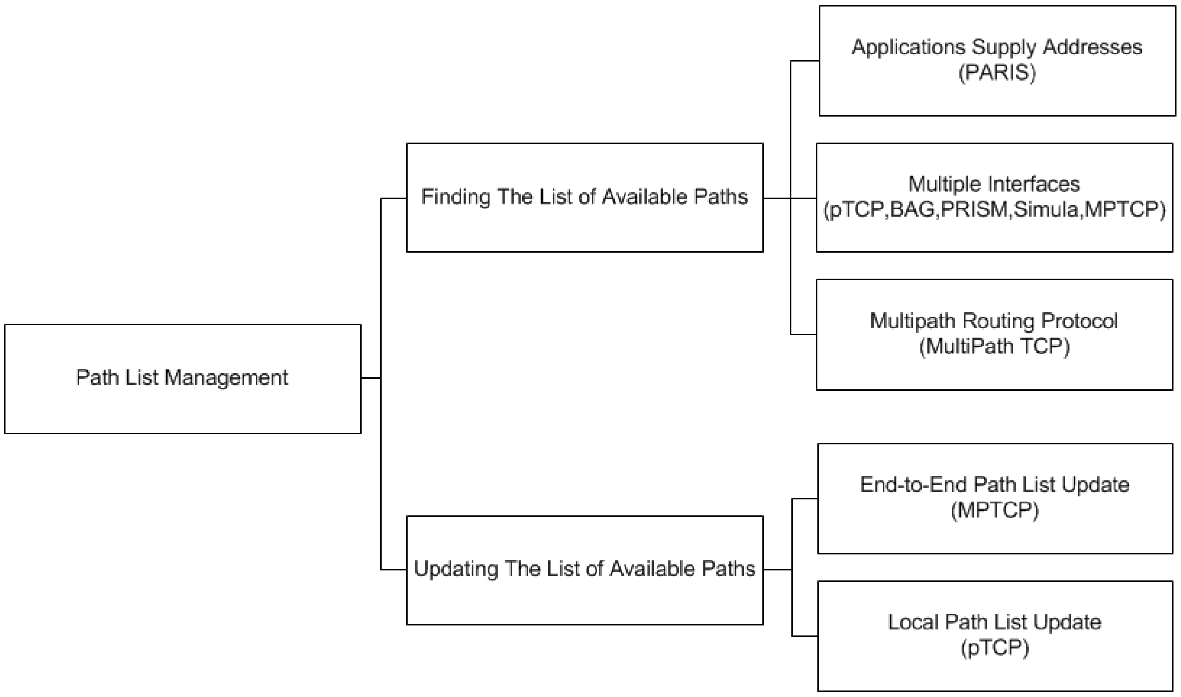

Path List Management
Finding The List of Available Paths
Updating The List of Available Paths
Applications Supply Addresses (PARIS)
Multiple Interfaces (pTCP,BAG,PRISM,Simula,MPTCP)
Multipath Routing Protocol (MultiPath TCP)
End-to-End Path List Update (MPTCP)
Local Path List Update (pTCP)

Figure 2.2: Path List Management Solutions

2.3 Force TCP Packets to Follow A Specified Path

The methods outlined in [32] may be used to direct packets via multiple paths in the Internet. Packets may be sent as they are received, may be (re)directed, or may be encapsulated before they are sent (Figure 2.3):

1. MultiPath TCP [18] uses source routing to specify the path that the packet will follow in the network.

2. BAG proxy captures TCP data packets sent by the TCP sender and uses IP-in-IP encapsulation to re-direct them to multiple interfaces of the multi-homed TCP receiver.

3. PRISM proxy captures TCP data packets sent by the TCP sender and uses *Generic Routing Encapsulation (GRE)* to re-direct them to neighbors of the TCP receiver. Then, neighbors decapsulate the packets and pass them to the TCP receiver which is within the same ad-hoc mode WLAN.

4. Simula Proxy captures TCP data packets sent by the TCP sender and uses network address translation (NAT) to re-direct them to multiple client interfaces. Simula proxy changes the destination address of the packets to direct it to targeted MH interface.

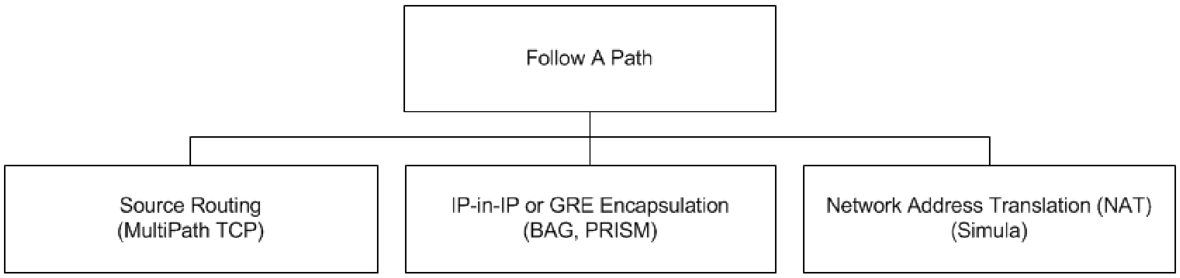

Follow A Path
Source Routing (MultiPath TCP)
IP-in-IP or GRE Encapsulation (BAG, PRISM)
Network Address Translation (NAT) (Simula)

Figure 2.3: Specifying The Path Solutions

2.4 Scheduling The Packets to The Paths and Finding The Path Characteristics

Path characteristics (e.g., bandwidth, delay, packet loss rate, utilization) may be used in packet scheduling decisions. For example, aggregation of path bandwidth is an important benefit of using multiple paths. If the path bandwidths are known, then the packets can be scheduled to the paths so that they don't cause congestion on the paths. The paths may be used to send data packets as efficiently as possible without exceeding their capacities.

The following methods are used to schedule packets to the paths (Figure 2.4):

1. A very basic approach to schedule packets to available paths is to forward packets to the neighbors in round-robin (RR) or random manner [15].
2. Horizon selects one of the neighbors as the next hop based on a cost function. Each node has a price which is proportional to the maximum amount of packets queued at the node as well as the back-pressure of the TCP flow via its neighbors. A node selects the next hop among the neighbors with the minimum cost.
3. Packet-pair based path capacity and delay estimation is used by BAG and Simula Proxies. They use estimated path capacities to determine how many packets may be scheduled to a path. As we stated in Section 2.1, path delays are used along with the path bandwidth to minimize out-of-order packet receptions in packet-to-path scheduling decisions.
4. The TCP-like path capacity probing is used by [39]. Each sub-connection has a cwnd which is increased/decreased similar to TCP cwnd.
5. PRISM tracks how many packets are sent via each path and then processes ACK packets to collect information about the path utilization and delay. PRISM selects the least utilized path to schedule a packet. If multiple paths have the same least utilization, then the path with lower delay is selected.
6. TCP-PARIS receiver has knowledge of the current cwnd value of the TCP-PARIS senders. Therefore, the receiver knows the servers that are able to transmit the next packets and decides on which server should transmit the following packets. A *Partition*

Rule (prule) for each server informs servers which segments to transmit. TCP options are used for the cwnd-prule information exchange. TCP-PARIS sender of each sub-connection piggybacks its current cwnd value on every data segment (as a TCP option of 8 bytes) to TCP-PARIS receiver. Similarly, prule is piggybacked (as a TCP option of up to 40 bytes) on ACK segments to the TCP-PARIS sender.

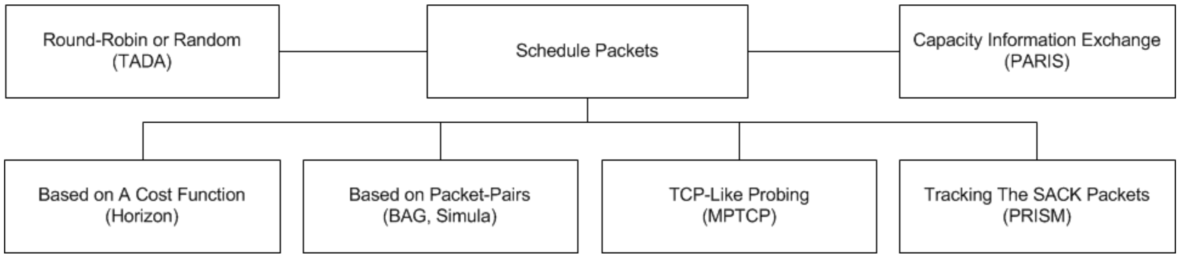

Round-Robin or Random (TADA)
Schedule Packets
Capacity Information Exchange (PARIS)
Based on A Cost Function (Horizon)
Based on Packet-Pairs (BAG, Simula)
TCP-Like Probing (MPTCP)
Tracking The SACK Packets (PRISM)

Figure 2.4: Packet Scheduling Solutions

2.5 Isolate Losses on One Path From The Other Paths

When a packet loss occurs on a path, TCP will reduce its sending rate by half. However, this may not be proper in some cases. Assume that there is a low- and a high- bandwidth path and a packet is lost on the low-bandwidth path. Since the TCP sending rate will be halved, high-bandwidth path will be under-utilized. The phenomena of a loss on a path not affecting the sending rates for the other paths is called as the *loss isolation* in [22].

Calculating the bandwidth of each path separately and decreasing the bandwidth of the path after detection of a packet loss on the path is the common solution to that problem. For example,

1. On a packet loss on a path, PRISM sends the path's bandwidth value to its TCP sender-side component, TCP-PRISM, in negative acknowledgments (NACKs). TCP-PRISM uses bandwidth information in setting its new cwnd value. Additive increase proportional decrease (AIPD) algorithm is used to set new cwnd proportional to the congested link's bandwidth over total bandwidth.

2. When a packet loss is detected on a sub-connection, [39] decreases only the cwnd of the sub-connection by half. Cwnd values of other sub-flows are not changed.

3. DEF looks for retransmissions by the TCP sender. If there is no retransmission, then DEF filters DUPACKs. Otherwise, it doesn't filter DUPACKs so that TCP sender can estimate the number of inflight TCP packets.

2.6 Handling Incorrect RTT/RTO Estimations

Since packets will arrive at the receiver out-of-order or via paths with different delays, TCP sender's RTT estimation may not be accurate. This may cause premature timeouts on the long paths and unnecessary waiting times for the lost packets on the short paths.

To the best of our knowledge, there is no solution which deals explicitly with the instable RTT estimations. Indeed, receiver side buffering of the out-of-order packets (See Section 2.1) implicitly handles this problem. Assume that we have a short path and a long path. Since out-of-order packets will arrive from the short path, they will be buffered till the packets come from the long path. They will be released after the arrival of these packets. Thus, TCP receiver will generate ACKs paced with the RTT of the long path.

2.7 Deployment of The Solutions

The solutions may be divided into two classes from the deployment point of view: (1)end-host based and (2)proxy-aided solutions (Figure 2.5):

1. End-host based solutions have components only on the TCP end- points. TCP sender and/or receiver TCP/IP stack implementation is modified to handle multiple-path related issues.
2. In addition to end-host changes, proxy-aided solutions have some components on a network element (i.e., a proxy) in-between the TCP sender and receiver. The proxy helps in handling the multiple-path related issues.

As we stated at the start of the chapter, deployment of the available solutions requires some form of support from the end hosts.

The proxy-based solutions [22, 31, 34] needs to deploy some components of their solutions on a proxy in between the TCP sender and the receiver:

1. BAG clients send their interface addresses to the BAG proxy.
2. Simula proxy must be configured to re-direct TCP packets to the interfaces of the Simula client.
3. PRISM proxy has to know addresses of WWAN interfaces of the neighbors of the TCP receiver to send TCP data packets via GRE encapsulation.

End-host based solutions require changes on the TCP-sender and/or the receiver:

1. *socket calls are modified:* TCP-PARIS defines a SOCK_PARIS socket option and gets addresses of the servers on which the file is replicated via modified connect() socket call. Similarly, MPTCP [41] uses socket options [43] in its implementation.
2. *TCP sender is changed:* PRISM, MPTCP, and TCP-PARIS replace TCP sender with their senders. TCP dupthresh value is changed by [15].
3. *TCP receiver is changed:* MPTCP and TCP-PARIS replaces TCP receiver with their receivers. TCP delayed ACK algorithm is changed in [15].

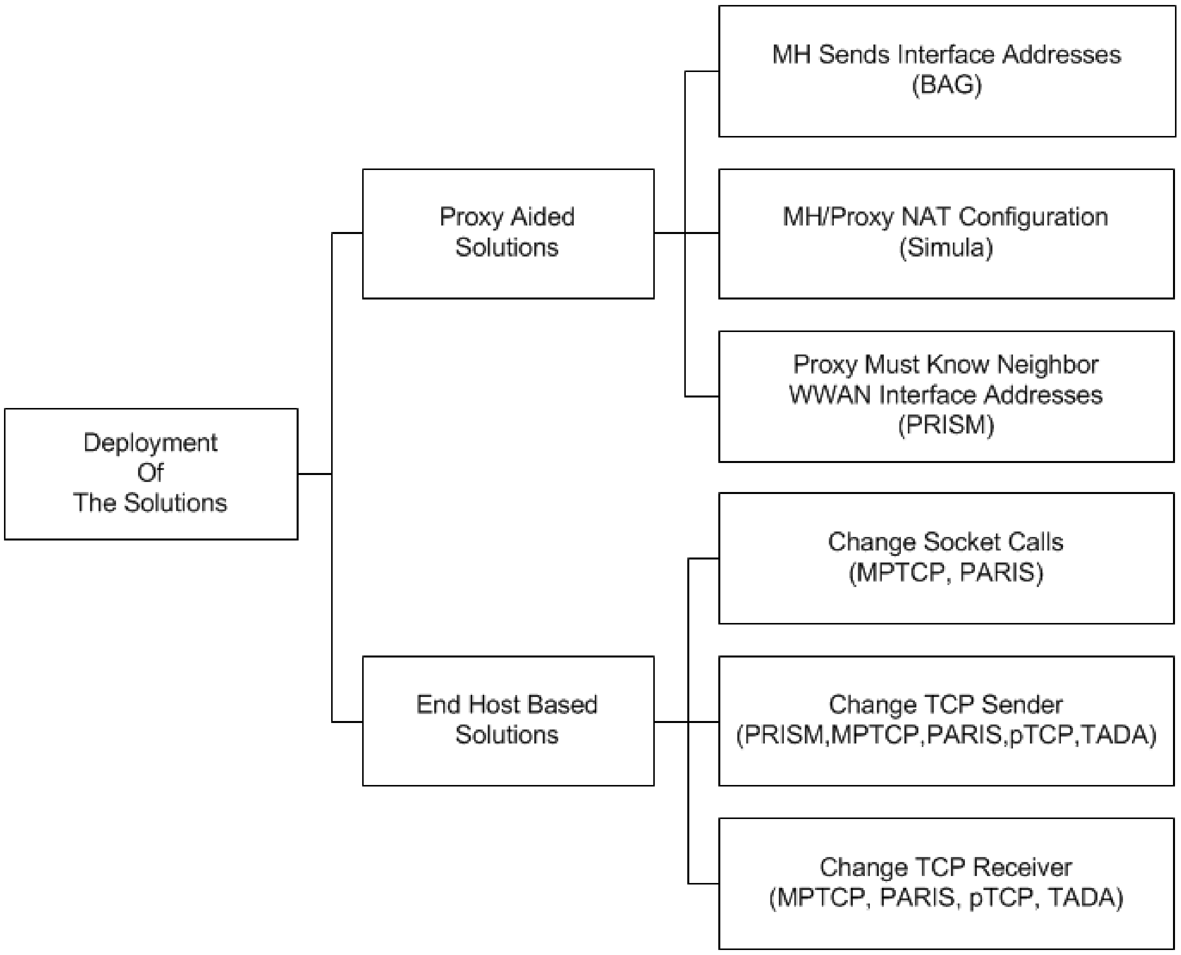

Deployment Of The Solutions
Proxy Aided Solutions
End Host Based Solutions
MH Sends Interface Addresses (BAG)
MH/Proxy NAT Configuration (Simula)
Proxy Must Know Neighbor WWAN Interface Addresses (PRISM)
Change Socket Calls (MPTCP, PARIS)
Change TCP Sender (PRISM,MPTCP,PARIS,pTCP,TADA)
Change TCP Receiver (MPTCP, PARIS, pTCP, TADA)

Figure 2.5: Deployment of The Solutions

Most of the TCP over multiple paths research (and the solutions presented in this paper) focused on the simultaneous use of interfaces of multi-homed hosts. Since these solutions generally require changes on the TCP senders and/or receivers, deployment and adoption of such solutions is a big problem [34, 38].

3 requirements are mentioned in [38] for a solution that requires changes at the both ends of the connection: (1)the solution must be implemented by the operating system, (2)at least one of the end-hosts must have simultaneous Internet access accross different network interfaces, and (3)both sides must be capable of using the solution.

Thus, solutions that require changes on the end-points (i.e., user premises) have a little chance of being adopted and deployed. It is necessary to design TCP over multiple paths solutions that are independent from the changes on end-hosts.

Chapter 3

TCP Splitter/Combiner Architecture (SCA)

In [48], we defined basic requirements for a generic TCP Splitter/Combiner Architecture (SCA) that enables development of highly deployable TCP over multiple paths solutions. In this technical report, we give the detailed description for the SCA and one of its applications.

SCA may distribute TCP data packets of a single connection to multiple paths (i.e., *splitter*) or may aggregate TCP data packets of a single connection from multiple paths (i.e., *combiner*).

SCA captures the TCP packets and processes their headers[1] to detect TCP connections and to process data/ACK packets of detected TCP connections to increase TCP performance. SCA aims at high deployment possibilities and is designed by a consideration of the protocol stack and end-to-end transparency:

- **Protocol Stack Transparency:** TCP multiple path related issues are handled in SCA which is placed beneath the transport layer. SCA design is isolated from the other layers. SCA may be implemented as a module or a thin layer.

  To the best of our knowledge, using a thin layer below the TCP is first introduced by ATCP [11]. Although ATCP uses its layer to remedy TCP problems in ad-hoc networks (i.e., frequent route changes and network partitionings), we design our layer to shield multiple-path use from the TCP.

- **End-to-End Transparency:** SCA works on *Split/Join Point(s)* which may be located on the TCP sender/receiver hosts as well as in-between the TCP sender and TCP receiver. Neither TCP sender nor TCP receiver must be aware of the presence of the SCA.

[1]We assume that TCP headers are readable by SCA.

3.1 SCA Protocol Stack Transparency

SCA is designed to be protocol stack transparent in order to alleviate the problem of deployment (Section 2.7). That is, SCA instances work standalone and their implementation does not need any changes on the TCP/IP implementation of the end-host operating systems (OSs). We call running SCA instances as SCA Proxies (SCAPs).

SCAPs may be located either on end-hosts or interim network elements (e.g., APs, routers, ... etc). SCA is placed below the transport layer (if exists) and above the forwarding layer (i.e., network or data link layer) that supplies multiple paths as shown in Figure 3.1. All the TCP packets must pass through the SCA.

Figure 3.1 shows SCA on the TCP end-points (e.g., multi-homed hosts). In that case, transport layer instance (i.e., TCP end-point entity) sends outgoing TCP packets to the lower layer. Similarly, forwarding layer instance (e.g., IP module) passes incoming TCP data/ACK packets to the upper layer. In both cases, SCAP captures packets and processes them. Neither TCP nor forwarding layer entity knows about the intervention of the SCAP.

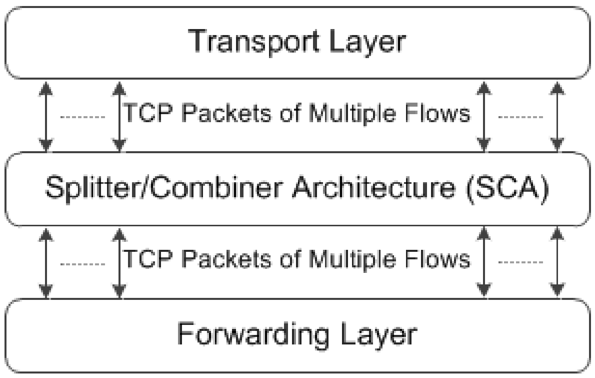

Transport Layer
TCP Packets of Multiple Flows
Splitter/Combiner Architecture (SCA)
TCP Packets of Multiple Flows
Forwarding Layer

Figure 3.1: SCA Position in a Network Protocol Stack

Figure 3.2 shows SCA on a network element other than TCP end-points (i.e., network elements without any TCP entity). In that case, forwarding layer instance (e.g., data link layer module) sends TCP packets to its upper layer entity which is the SCAP. The SCAP processes the packet and sends it back to forwarding layer instance or drop the packet.

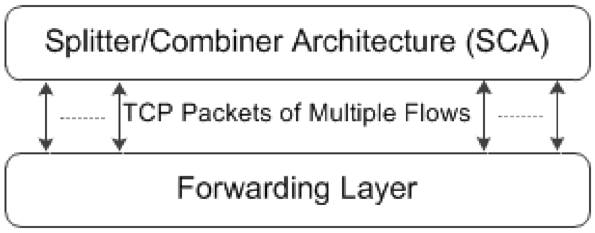

Splitter/Combiner Architecture (SCA)
TCP Packets of Multiple Flows
Forwarding Layer

Figure 3.2: SCA Position on A Network Element's Protocol Stack

3.2 Pipes: SCA Abstraction for Multiple Paths

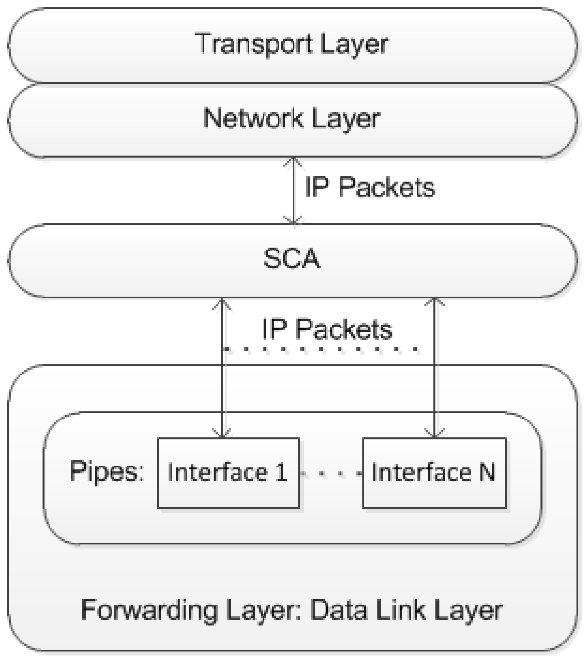
SCA requires that the functionality of splitting TCP traffic via multiple ”paths” is provided by the forwarding layer. Multiple paths may be provided by means of:

- IP source routing [1]: TCP packets may be requested to follow a specific path. IP Loose/Strict Source and Record Route option may be used to specify the path they must follow.
- IP-in-IP encapsulation [5]: The source and destination SCAP IP addresses may be specified in the outer IP header to create tunnels between a SCAP peer.
- IP-in-IP tunneling [4]: SCAPs may use special headers to exchange signaling information. Tunnel Headers in the IP-in-IP tunneling may be used to carry SCA signaling information.
- Multi-path routing protocols: Packets may be sent via routes find by a multipath routing protocol (e.g., SMR [10], YAMR [35], etc.).
- Multiple interfaces of a multi-homed host: Packets may be sent via multiple access networks when interfaces are connected to different access networks. Alternatively, direct links between the cross-connected multiple-interface neighbors may be used to send packets.
- Multiple channels on the same physical link (e.g., more capacity supplied via different WDM wavelengths).

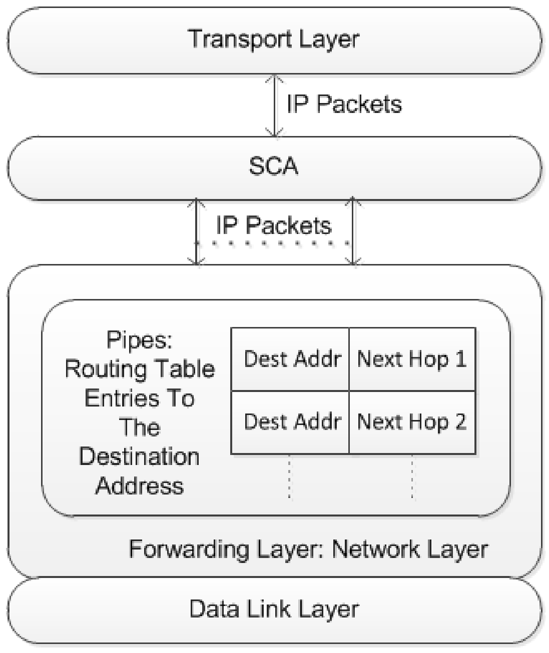
Figure 3.3a shows an example of a SCAP that works in between the network layer and data link layer of a host with multiple interfaces. The interfaces of the multi-homed-host are used as the pipes. TCP packets are sent via each interface to the receiver.

Figure 3.3b shows an example of a SCAP that works in between the network layer and transport layer of a host. Multiple routes to the destination in the routing table are used as the pipes. If there is only one route, a multipath routing protocol (e.g., SMR [10], YAMR [35]) may be initiated to find the multiple paths.

In order to be independent of the multiple path implementation, we refer to the term *pipe* instead of path in the remainder of this report. We assume that pipes may be accessed/used by SCAPs to send TCP data/ACK packets via them.



(a) SCA Uses Interfaces of a Multi-Homed Host as Pipes



(b) SCA Uses Routing Table Entries as Pipes

Figure 3.3: Example Pipes

3.3 Overview of The SCA Components

SCA includes components related to the management of TCP connection records, processing of TCP data and ACK packets, and SCA signaling packets (Figure 3.4):

- *Packet Classifier* captures TCP packets from the lower/upper layer and classifies them based on the header information. It passes TCP packets to the related SCA component (e.g., FIN packets showing a connection release are passed to the *Connection Handler*, whereas Data/ACK packets must be processed by the *Data/ACK Processor*).
- *Multiple Pipes Adapter* provides a common interface to use pipes in the underlying network.
- *Connection Handler* reacts to new TCP connections by creating records of the new TCP connection and discarding connection information in case of a TCP connection release.
- *Data/ACK Processors* process the data/ACK packets of the TCP connections.
- *Signaling Unit* processes the SCA signaling packets.
- *Configuration and Management Unit* is used to configure parameters used by the components.

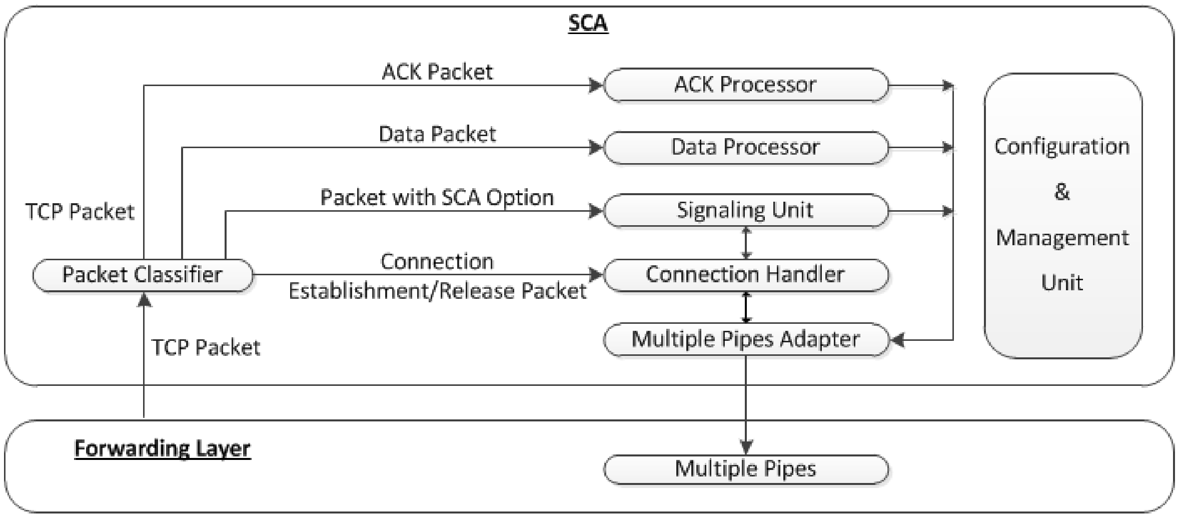


Figure 3.4: SCA Components

3.4 Packet Classifier

Packet Classifier is responsible for looking at the TCP header to determine the packet type. An admission control policy may be applied to packets (Section 3.9) before accepting them into the SCAP. The packets that are not admitted will be passed to the lower/upper layer without SCAP processing.

SCAP-accepted TCP packets will be passed to the related SCA component:

- SCA signaling packets are used to detect and coordinate SCAPs on the pipe between the TCP sender and receiver. All the signaling packets are handled by *Signaling Unit*.
- TCP connection establishment packets (i.e., packets with SYN flag set) are passed to *Connection Handler* to construct/select entries of multiple paths.
- Data/ACK packets are passed to *Data/ACK Processor*.
- TCP connection release packets are passed to *Connection Handler* to remove entries of multiple pipes assigned for the flow.

3.5 Connection Handler

Connection Handler manages the records of the TCP connections. A new connection request is detected by means of TCP segments which have SYN flag is set and ACK flag is not set.

Connection identifiers (*conn_ids*) are assigned to the connections based on their TCP sender and receiver end points (i.e., sender and receiver IP addresses and port numbers).

After a connection request is detected, Connection Handler assigns pipes for the connection. The list of available pipes are obtained from the Multiple Pipes Adapter by means of find_pipes() API call (Section 3.6.1).

Multiple Pipes Adapter finds the list of available pipes to the receiver, assigns a pipe identifier (*pipe_id*) to each pipe and returns back the list of pipes to the Connection Handler. In this phase, each pipe may be identified with the (conn_id, pipe_id) pair. The list of pipes is passed to the Pipe Selection Policy to get a subset of these pipes. New connection is marked as a pending connection (three-way handshake is not complete for the connection establishment).

When a TCP segment with SYN and ACK flags are set has arrived for a pending connection, then connection establishment is confirmed. This also shows that SCA is on the reverse path of the connection. That is, we may expect ACK packets pass through the SCA in addition to the data packets. Connection is marked as an active connection.

When TCP connection release packets (i.e., packets with FIN and RST flags set) are received, the TCP connection release procedure is followed. When the connection tear-down is complete, the connection is marked as closed and connection related records are deleted (e.g., withdraw_pipes() (Section 3.6.4) is called from the Multiple Pipes Adapter).

3.6 Multiple Pipes Adapter

SCA must be used on network elements which support multiple pipes that are within the forwarding layer. Since the pipes may be available in different forms (e.g., multiple next hops for the same destination in a routing table, multiple interfaces of a multi-homed host to different networks) or must be constructed by means of different algorithms (e.g., by means of using a multi-path routing algorithm), Multiple Pipes Adapter is defined as a component which interacts via specified APIs to other components of the SCA and handles interaction with the forwarding layer.

Multiple Pipes Adapter interacts with the forwarding layer to access and use the available multiple pipes. Other SCA components use the Multiple Pipes Adapter APIs to get information about the pipes that may be used and to send data via them.

3.6.1 Get The List of Multiple Pipes

PIPE-LIST find_pipes(conn_id, destination_address)

API is used to find pipes that may be used to send TCP data packets to the destination_address and returns the list of them, if any.

tcp_conn_id is used to identify the connection that needs the pipes to be assigned. Multiple Pipes Adapter may access information about the connection (e.g., source/destination TCP end-point) by using the tcp_conn_id. The API also associates the list of pipes found with the connection.

The PIPE-ID-LIST contains pipe_ids which are assigned by the Multiple Pipes Adapter to the available pipes (i.e., each pipe has a unique pipe_id).

Pipe implementation is shielded from the other components by means of pipe_ids: other components only know pipe_ids and Multiple Pipe Adapter handles pipes (Section 2.2).

3.6.2 Probe For Pipes Characteristics

PIPE-PROPERTIES get_pipe_properties(tcp_conn_id, pipe_id)

API is used to probe for characteristics of a pipe. PIPE-PROPERTIES includes the pipe parameters probed by the SCAP (e.g., bandwidth, delay, utilization, packet loss rate, etc.).

In order to get pipe parameters, following methods may be used: packet-pair based estimation of the pipe delays and capacities [22, 34], tracking the TCP timestamps [3], SACK [6] or D-SACK [9, 21] to estimate packet arrivals at the receiver, or probing pipe capacities with TCP-like cwnd increase and decrease [39, 42].

In addition, the local MIB information (e.g., [8, 20, 26, 29]) may be used to set the initial values just after the pipes are found by Multiple Pipes Adapter (e.g., packet loss rate of an interface, number of active TCP connections, etc.). These values may be used as the initial values. Multiple Pipes Adapter or Signaling Unit is responsible for monitoring the pipes during the connection (e.g., does the pipe still usable? What are the current estimations for the pipe capacity/delay/loss rate?, etc.).

If tcp_conn_id is NULL, then the total pipe characteristics are returned. Otherwise, tcp_conn_id is used to get pipe characteristics relevant to a TCP connection. For example, the bandwidth used by a TCP connection may be asked as well as the total bandwidth of the pipe.

3.6.3 Employ Newly Available Pipes

PIPE-ID-LIST employ_pipes(tcp_conn_id, pipe_id_list)

API is used by Multiple Pipes Adapter to associate pipes which become available after the connection establishment. The API may be called when an interface becomes active or a host associates with an AP/router after the pipes were assigned to the TCP connection by use of the find_pipes() API.

Pipes in the pipe_id_list are associated with the connection with tcp_conn_id.

3.6.4 Delete Pipes

PIPE-ID-LIST withdraw_pipes(tcp_conn_id, pipe_id_list)

API is used when a connection terminates or connection doesn't need some pre-assigned pipes. Pipes in the pipe_id_list are disassociated from the connection with tcp_conn_id.

The API must be called after the connection termination and may be called during the connection to cancel the use of some pipes which are previously assigned to the connection.

In addition, Multiple Pipes Adapter may use this API to disassociate some pipes which become unavailable during the connection. For example, an interface becomes inactive or a host looses its connection to an AP/router.

3.6.5 Select A Pipe Subset

PIPE-ID-LIST select_pipes(tcp_conn_id, pipe_id_list)

API is used to select a subset from a set of pipes based on a pipe selection policy (Section 3.9).

tcp_conn_id is used to determine the pipe selection policy for the connection. The withdraw_pipes() API is used to disassociate filtered pipes by the pipe selection policy and remaining pipes are returned back.

3.6.6 Send A Packet via one of The Multiple Pipes

send_mp_packet(tcp_packet, tcp_conn_id, pipe_id)

API is used to send a packet via one of the multiple pipes.

Since other SCA components only know the pipe_ids and do not know what the real pipes are, they use this API to send a tcp_packet which belongs to the connection with tcp_conn_id by using the pipe with pipe_id.

tcp_conn_id is supplied to enable update for pipe characteristics related with the TCP connections. For example, if a pipe is assigned to more than one TCP connection, what proportion of the pipe is used by the assigned TCP connections.

3.6.7 Hide The Use of Encapsulation

TCP-PACKET check_encapsulation(tcp_packet)

API is used to check whether tcp_packet is encapsulated or not. As stated in Section 3.2, packets may be sent/received encapsulated. Since other components are shielded from the use of encapsulation, only the Multiple Pipes Adapter knows about the encapsulation.

send_mp_packet() API encapsulates packets before sending them when encapsulation is necessary.

When packets are received, check_encapsulation() is used to get decapsulated packets, if encapsulation is used for the connection. It returns the packet immediately if encapsulation is not used.

3.7 Data/ACK Processor

Data/ACK Processor is responsible for handling data/ACK packets. They deal with the problems of preventing/handling out-of-order packet receptions (Section 2.1) and scheduling of packets to the pipes (Section 2.4).

It may apply to a TCP data/ACK packet one of the following operations (*4-D*):

1. *duplicate:* TCP packets may be (one or more times) duplicated (e.g., [18]) and scheduled to multiple pipes to create robustness against packet errors.

2. *delay:* TCP packets may be buffered and delivered later, based on a timer or a condition. For example, some packets of a TCP flow may be intentionally delayed to shape the TCP traffic (e.g., to reduce number of out-of-order packet arrivals at the receiver side [34], or to seperate sent packets by a given interval [18, 22]). Another example is to buffer out-of-order packets at the receiver before passing them to the TCP receiver [22].

3. *deliver:* TCP packets may be released immediately after their capture, or after the delay or duplicate operation. Delivery may be done locally for incoming packets or over any available pipe to outgoing packets. For outgoing packets, one of the available pipes must be selected to efficiently utilize the pipes.

4. *drop:* TCP packets may be dropped by SCA. For example, assume that a packet is buffered for early retransmission purposes. However, the ACK for that packet arrived and the buffered packet is not needed to be kept any more, and thus, may be dropped.

3.8 Signaling Unit

SCAPs on different SCAP devices may exchange signaling information by means of the following methods:

1. *SCA in-band signalling* may use TCP options to carry signalling information. The SCAP that wants to send a signaling information will generate an SCA option and add it to the TCP packet. The peer SCAP will use the content of the SCA control option and remove it from the packet.

   The main drawback of the in-band signaling is that its size is limited by the TCP option space allowed. 16-21 bytes of space is left for the SCA options when the most common TCP options are encountered [41].

2. *SCA out-of-band signaling* may use signaling channels established between the SCAPs.

   SCAPs may use a well-known port number to exchange signaling packets (e.g, RIP routing processes use UDP port 520 [7] or BGP systems use TCP port 179 [25]).

   Signaling TCP packets will be generated to pass control information from one SCAP to the other. Using TCP packets along with the signaling ports gives freedom in using as much space as need by means of data portion of the TCP packets.

3. *SCA hybrid signaling* may use in-band signaling to carry out-of-band signaling channel end-point information between SCAPs. First a connection between the signaling channel end-points will be established. Then, out-of-band signaling over the connection may be used to exchange the signaling data.

3.9 Configuration and Management Unit

Configuration and Management Unit is used to set parameters for the SCA components. The configuration parameters may be defined for each component:

- *Packet Classifier* may accept packets into the SCAP or reject them based on an admission control policy.

  Based on the TCP end-points, some applications or hosts may not benefit from the SCA:

  - Splitting the traffic may not be necessary for some applications. For example, short-lived flows like web traffic may be carried over only one path instead of multiple paths as they will most likely end within a couple of packets anyway. Thus, they may not be accepted for the SCA processing.
  - Splitting the traffic may not be necessary for some applications. For example, short-lived flows like web traffic may be carried over only one path instead of multiple paths as they will most likely end within a couple of packets anyway. Thus, they may not be accepted for the SCA processing.

- *Connection Handler* may configure the number of acceptable connections based on the current number of connections or the total number of pipes used by the connections.

- *Multiple Pipes Adapter* may use a pipe selection policy to select a subset of the pipes from the set of available pipes. Using different policies in pipe selection gives additional flexibility to the SCA.

  Pipe selection policy may be based on the pipe parameters (e.g., delay, loss rate, throughput, etc.) to decide which pipes will be selected. For example, only the pipes that have close RTT values may be selected or some pipes with high BERs may be excluded.

  In addition, pipe selection policy may be based on the TCP end- points. For example, some privileged flows (e.g., flows originated from some IP addresses) may be assigned more pipes and some other flows may be penalized and forced to use only a single pipe.

Chapter 4

SCA Deployment

SCA deployment will take some time. We envision that SCAPs may work in three modes based on their deployment level:

1. *Standalone SCAP:* If there is only one SCAP on the path between the TCP sender and the receiver, SCAP must work alone

2. *SCAP Pair:* If there are two SCAPs on the path, then they may work as a peer

3. *SCAP Chain:* If there are multiple SCAPs on the path, then they may be used as cascaded chains

We will use the network architecture in Fig. 4.1 to discuss how SCAPs may be used in different network scenarios.

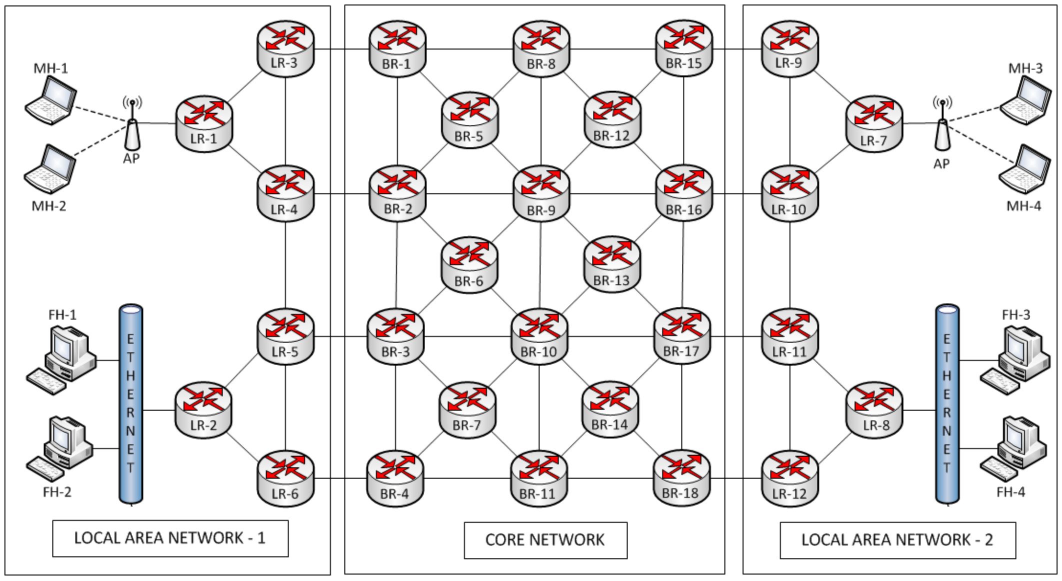


Figure 4.1: Common Network Architecture

Our example network consists of two local networks (LANs) connected to each other via a core network (CN). LAN clients may have wired or wireless (e.g., Femtocells or LTE) interfaces. LANs are connected to CN via multiple gateway (GW) links (e.g., for redundancy or since GW links have low capacities)..

Fig. 4.1 shows CN as a mesh network of routers. The routers may be connected via wired as well as wireless links (e.g., a wireless mesh network backbone). In addition, LAN GW links may be connected to separate CNs (e.g., different ISP networks).

4.1 SCAP on a Single Network Element

A possible scenario that SCAPs will be used includes wireless access networks (e.g., WLANs or WMNs) that are connected to the backbone via low-capacity GW links. Thus, the first SCAP deployments will be on some access networks. In that case, SCAPs used on an access network may not find a peer SCAP to collaborate.

Fig. 4.2 shows SCAPs located only on the LAN-1 routers, LR-1 and LR-2. In that case, SCAPs may act as a splitter and combiner for the uploaded data since there are more than 1 path for the upload data traffic. For the downloads, they have to forward packets via 1 path since there are no other paths.

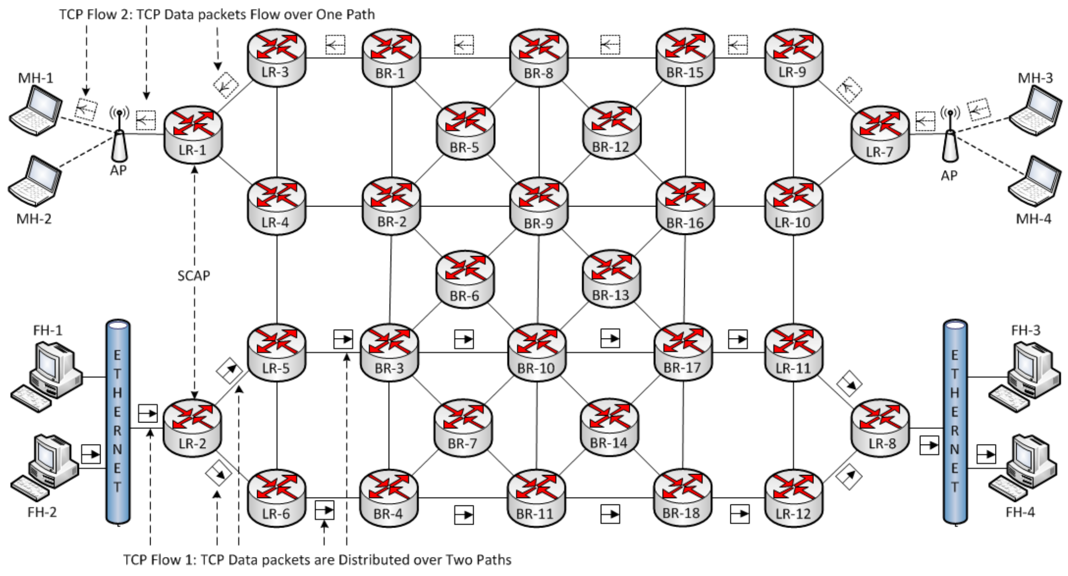


Figure 4.2: Standalone SCAP

Fig. 4.2 shows two TCP flows: data transfer from FH-2 to FH-4 and from MH-3 to MH-1. Since there is no peer for the SCAPs, they have to work alone in both cases. In the flow from FH-2 to FH-4, SCAP on LR-2 detects the TCP connection and distributes data packets to its neighbors (i.e., LR-5 and LR-6). If GW links have low capacities (e.g, 1.544 Mbps T1 links or DSL lines), then the TCP connection may benefit from the aggregated capacities of the both GW links. In addition, the SCAP may process the incoming ACKs to prevent TCP sender from the side-effects of data packet distribution. An example of this kind of SCAP is presented in Chapter 5.

Contrary, in the flow from MH-3 to MH-1 SCAP on LR-1 detects the TCP connection but can't distribute the data packets because of the data packet flow direction.

4.2 SCAP Pairs

When SCAPs are adopted, they will have high deployment on LANs. It will be easy for a SCAP to find a peer to collaborate: SCAPs that are located on LANs of the TCP sender and receiver may work as a peer, as shown in Fig. 4.3.

In that case, one SCAP may operate as the splitter and the other one as the combiner. Splitter may distribute data packets to multiple pipes and the combiner may collect and reorder the distributed packets. The receiver may only get the packets in-order. In Figure 4.3, LR-7 and LR-2 work as splitters, while LR-1 and LR-8 work as combiners.

Signaling between the SCAP pairs may be provided, for example, by means of pTCP headers [19] or IETF MPTCP options. MPTCP TCP options or pTCP headers may be added by splitter SCAP and removed by the combiner SCAP. Then, the hosts with single interfaces may benefit from the multiple paths within the core network.

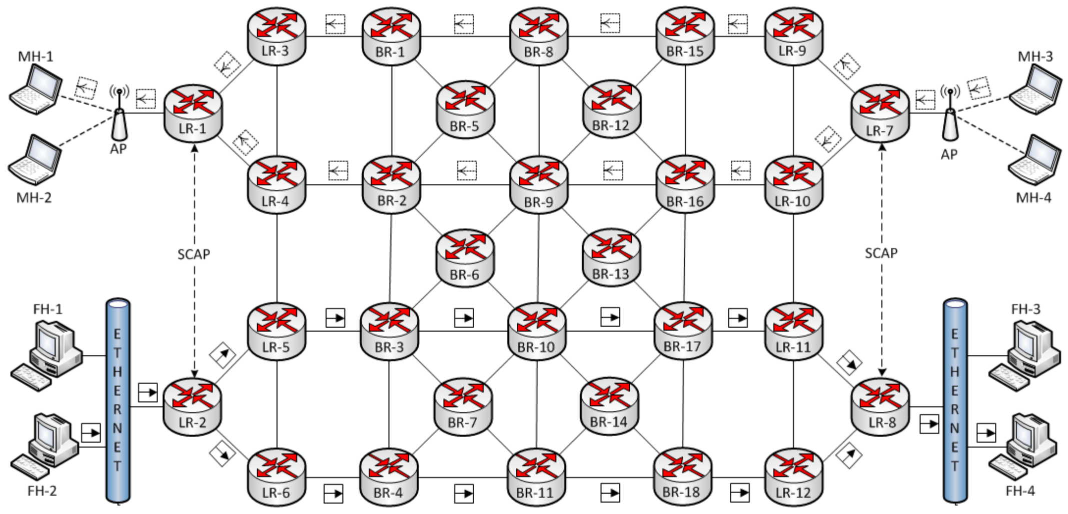


Figure 4.3: SCAP Pairs

4.3 SCAP Chains

After high deployment of SCAPs on LANs, we expect them also to appear in CNs. CNs may use SCAPs for load balancing. When SCAPs are deployed on CNs, we expect to see SCAPs as chains, as shown in Fig. 4.4.

In this scenario, we show SCAPs on CN routers. BR-4 works as a splitter. It captures packets from the TCP sender (FH-1) and distributes them via disjoint paths to the first combiner (BR-10). BR-10 works as both a combiner and a splitter. It combines packets distributed by BR-4 and distributes them via disjoint paths to the next level combiner (BR-17). BR-17 combines packets and sends them in-order to the TCP receiver (FH-3).

Since (BR-4,BR-10) and (BR-10, BR-17) constitute SCAP peers, they may use SCA signaling packets in this design as mentioned above, in Section 4.2.

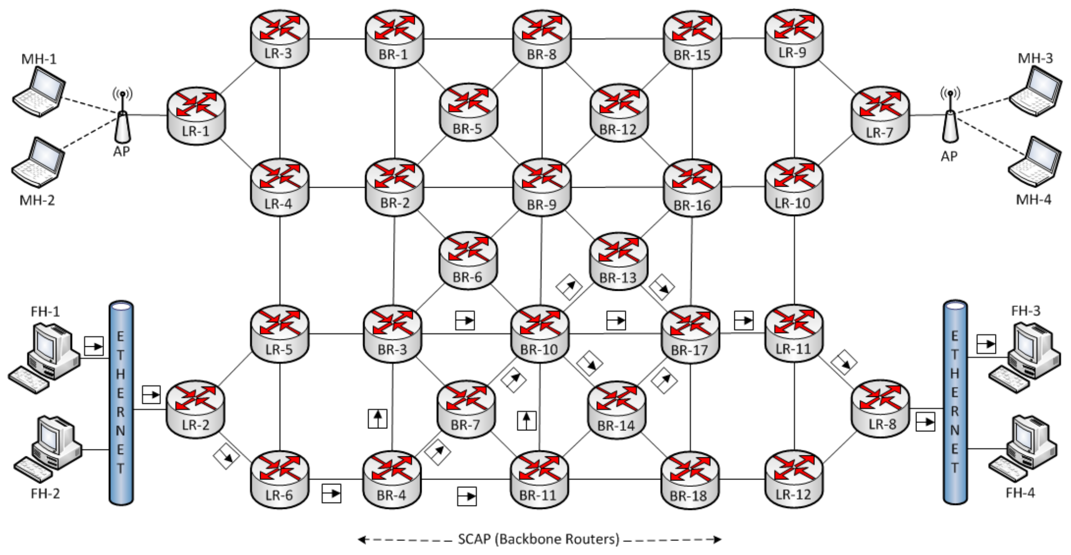


Figure 4.4: SCAP Chain

Chapter 5

A Standalone SCAP

We started our work with the case that there is only one SCAP on the path between the TCP sender and the receiver. Therefore, our standalone SCAP may work on a network scenario as shown in Figure 4.2.

Our solution defines an algorithm to improve TCP performance with *RTT based DUPACK Estimation and Filtering (DEF)*. It mainly deals with out-of-order packet receptions problem mentioned in Section 2.1.

5.1 DEF Assumptions

DEF may be used in any network which satisfies the following assumptions:

1. There are multiple paths between the split point and the TCP receiver.
2. TCP sender uses only one interface to send/receive data.
3. There is only one path between the TCP sender and the split point.
4. Path RTTs between the SCAP and the receiver are stable (i.e, path RTT variance is low).
5. TCP sender and receiver have enough buffer spaces to keep out-of-order packets.

Figure 5.1 shows an example network for the DEF: TCP sender uses one interface to send the TCP packets. DEF is placed on a router (i.e., split point) in between the TCP sender and receiver. It works under the network layer and above the data link layer. There is only one pipe between the TCP sender and the split point. There are multiple pipes between the split point and the TCP receiver.

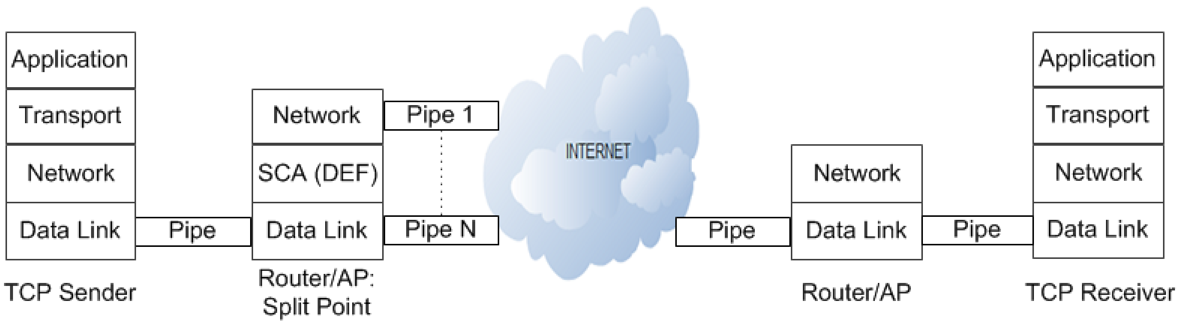


Figure 5.1: DEF Position on the Network

5.2 DEF States

As shown in Fig. 5.2, DEF uses either all the available paths of the connection based on the round robin (RR) data scheduling (NO_LOSS state) or only the shortest path (RETRANSMISSION state). All the packets are scheduled to the shortest path in RETRANSMISSION state to minimize further DUPACK generation.

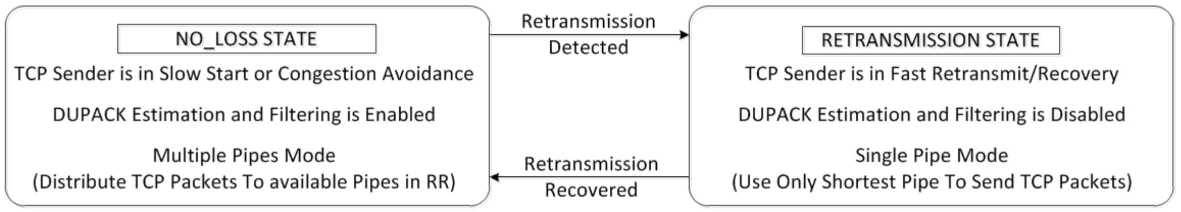

NO_LOSS STATE
TCP Sender is in Slow Start or Congestion Avoidance
DUPACK Estimation and Filtering is Enabled
Multiple Pipes Mode
(Distribute TCP Packets To available Pipes in RR)
Retransmission Detected
Retransmission Recovered
RETRANSMISSION STATE
TCP Sender is in Fast Retransmit/Recovery
DUPACK Estimation and Filtering is Disabled
Single Pipe Mode
(Use Only Shortest Pipe To Send TCP Packets)

Figure 5.2: DEF State Transitions

5.2.1 NO_LOSS State

When data transmission of a connection is started, DEF enters NO_LOSS state till a retransmission is observed. In order to detect retransmissions, DEF keeps track of the highest data sequence number sent by the sender so far. When a packet is received from the sender, this variable is updated. If a data packet is received with smaller sequence number than this variable, then a retransmission is inferred and RETRANSMISSION state is entered.

5.2.2 RETRANSMISSION State

DEF stays in RETRANSMISSION state till the TCP sender recovers from the packet retransmissions. We defined three requirements for DEF to exit RETRANSMISSION state:

1. *A new ACK must be received:* TCP sender recovery phase may not be completed if there are still DUPACK receptions.

2. *All the data packets scheduled for the paths before entering the RETRANSMISSION state must reach to the receiver:* so that packets scheduled for longer paths before entering the RETRANSMISSION state doesn't generate unexpected DUPACKs.

3. *All the packets before entering the RETRANSMISSION state must be recovered:* so that DEF doesn't switch between RETRANSMISSION and NO_LOSS states unnecessarily. When TCP sender receives partial ACKs, it may retransmit some segments which may cause DEF to enter RETRANSMISSION state again just after it enters NO_LOSS state.

5.3 DUPACK Estimation and Filtering Algorithm

As shown in Fig. 5.2, DEF algorithm is enabled only in NO_LOSS state in which multiple paths are used. When DEF data processor schedules a packet to a path, it estimates whether the packet arrival to the receiver will generate a DUPACK or not. In order to decide, DEF keeps track of following values for each path:

- *path_rtt* is the estimated RTT of the path.
- *highest_data_seq* is the highest data sequence number scheduled for the path.
- *highest_data_time* is the estimated arrival time for the highest-sequenced data packet scheduled for the path. Since ACK packets follow the same route, independent of the path the data packet is sent to, RTTs, instead of one-way delays, are used to estimate data packet arrival times.

DUPACK estimation works as follows. When a packet with sequence number seq_i is scheduled for the path p_i at time t_i, its arrival time is estimated as $\mathrm{arr}_i = \mathrm{t}_i + \mathrm{p}_i.\mathrm{path\_rtt}$. DEF checks all paths to find the path which has ($\mathrm{seq}_i > \mathrm{p}_j.\mathrm{highest\_data\_seq}$) and ($\mathrm{arr}_i < \mathrm{p}_j.\mathrm{highest\_data\_time}$). If such a path p_j exists, then the segment will reach to the receiver out-of-order. In that case, an*estimated_dupacks* counter is incremented by 1.

DEF ACK processor keeps track of highest ACK number received so far for the connection. It uses this value to check whether a received ACK is a DUPACK or not. If a DUPACK is received, then the estimated_dupacks counter is checked. If it is non-zero, then the ACK is an expected RTT-difference-based DUPACK. DEF ACK processor simply filters (i.e., drops) the DUPACK and decrements the estimated_dupacks counter by 1. In that way, the TCP sender is prevented from reception of unnecessary DUPACKs.

5.4 Retransmission Detection and Response Algorithm

DEF uses the highest sequence number of data packets sent so far to decide on retransmission events. In the NO_LOSS state, if a packet is received with sequence number lower than this value, then a retransmission is inferred.

DUPACKs are harmful when the TCP sender is in the NO_LOSS state (i.e., either in slow-start or in congestion avoidance phase) with increasing sequence numbers. However, when

TCP sender is in the RETRANSMISSION state (i.e., in fast retransmit/recovery), DUPACKs are useful since they provide the information about the number of in-flight packets and may give an opportunity to increase the TCP congestion window (cwnd). Thus, before entering the RETRANSMISSION state, DEF data processor sets value of estimated_dupacks counter to 0, disabling the DUPACK filtering.

DEF data processor sets the following variables of the connection before entering the RETRANSMISSION state to fulfill the criteria specified in Section 5.2:

- *checkpoint_data_time* is set as the highest estimated data arrival time when the RETRANSMISSION state is entered (Criteria 2).
- *checkpoint_data_seq* is set as the value of highest_data_seq variable when the RETRANSMISSION state is entered (Criteria 3).

DEF ACK processor checks these variables upon reception of a new ACK (Criteria 1). If all three criteria are satisfied, then DEF leaves the RETRANSMISSION state and enters the NO_LOSS state again.

Chapter 6

The Standalone SCAP: Prototyping and Performance Evaluation

We implemented DEF as a pluggable netfilter [44] module in Linux. NF_INET_POST_ROUTING hook was used to capture and process TCP packets after the packets are processed by the network layer code of the Linux.

We used NCTUns 6.0 [45] simulation and emulation tool to evaluate performance of the DEF algorithm. NCTUns uses real Linux kernel TCP/IP implementation by means of a kernel re-entering methodology when simulating networks [16]. It uses tunnels to direct real TCP packets to the related simulation entity. Thus, TCP packets pass through the real Linux kernel TCP/IP protocol stack several times till it reaches to its destination entity in the simulation environment.

Our testbed consists of two Pentium dual-core E5300 2.60GHz CPU machines with 2 GB memory as shown in Fig. 6.1. Two machines are connected to the same local network via a switch. One of the machines is used to run NCTUns simulations. Since we want to test DEF with different OSs, we used the second machine as a Windows XP, Ubuntu 11.04, and PC-BSD 8.2 TCP sender. We used the emulator capability of NCTUns [37] for this purpose. We defined TCP sender as an "external" host that is included within the simulation scenario (Figure 6.2).

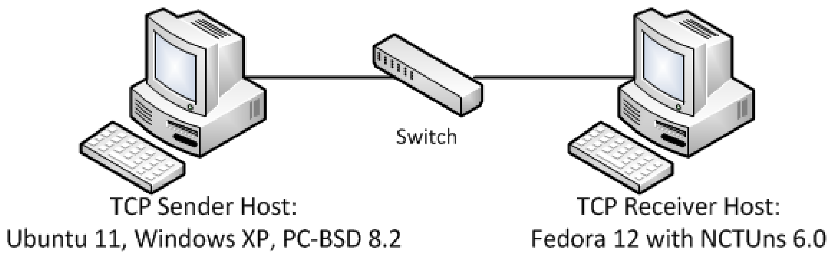


Figure 6.1: Test-bed for DEF Performance Evaluation

Our scenario (Fig. 6.2) consists of an external TCP sender host (node1), 4 routers within the local backbone (nodes 2 to 5), 3 GW links, each with the capacity of 1.544 Mbps (as in T1), 3 paths within the Internet with 10 Mbps capacities and different delays (D1, D2, and D3, respectively), and a simulated TCP receiver host (node10).

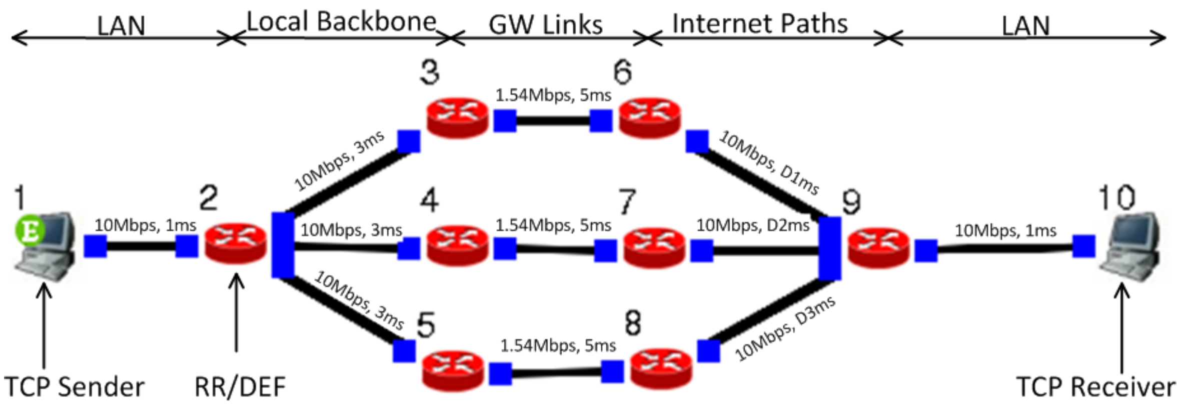


Figure 6.2: NCTUns Emulation Scenario for DEF Performance Evaluation

As shown in Table 6.1, the paths delays are set as follows: D1 is constant and has a value of 10 ms in each direction. Two (D1 and D2) and three path (D1, D2 and D3) cases are considered, with delay differences between the paths ranging from 0 ms (all the paths have the same delay) to 160 ms (consequtive paths have 160 ms delay difference) in 40 ms steps.

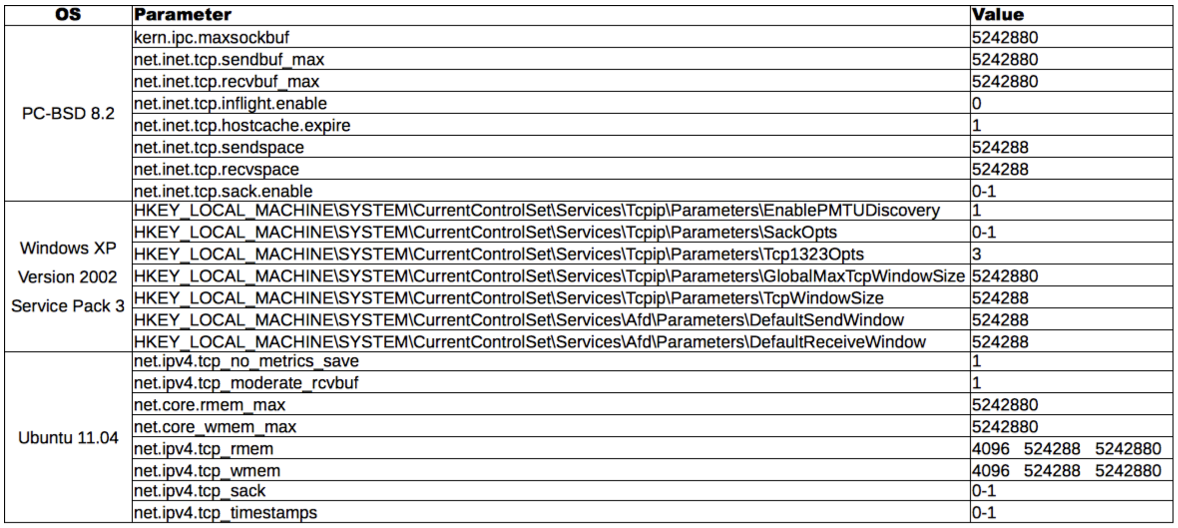
Table 6.1: Path Delays and RTTs used in NCTUns Emulations

D1/RTT1 (ms)	D2/RTT2 (ms)	D3/RTT3 (ms)	Delay/RTT Difference (ms)
10/40	10/40	10/40	0
10/40	50/80	90/120	40
10/40	90/120	170/200	80
10/40	130/160	250/280	120
10/40	170/200	330/360	160

Different OSs may use different variants of TCP by default or support different TCP variants which may be set by the end-users [40]. Since DEF is aimed at high deployment, we used an external machine as the TCP sender host to test how DEF performs with different OS TCP implementations. We selected representatives from the different OS families: Microsoft Windows XP Professional Version 2002 Service Pack 3 (Windows), Ubuntu 11.04 (Linux), and PC-BSD 8.2 (BSD).

Default values of TCP related OS parameters (e.g., send/receive buffer sizes, TCP window size. . .) may not allow TCP to achieve high data transfer rates. Since data packets must be buffered by the receiver before the in-order packets come from the longest path, TCP receiver must have enough buffer space. Similarly, TCP sender must keep these packets in its send buffer till it receives ACKs for the packets. Thus, TCP sender must also have enough buffer space. In addition, TCP window size must be large enough to allow high volumes of data transfers. We executed our experiments following the suggestions in [30, 46, 47] to configure OS parameters related to the TCP performance. Table 6.2 shows the configured system parameters of the OSs.

Table 6.2: Configured OS System Parameters



OS	Parameter	Value
PC-BSD 8.2	kern.ipc.maxsockbuf	5242880
	net.inet.tcp.sendbuf_max	5242880
	net.inet.tcp.recvbuf_max	5242880
	net.inet.tcp.inflight.enable	0
	net.inet.tcp.hostcache.expire	1
	net.inet.tcp.sendspace	524288
	net.inet.tcp.recvspace	524288
	net.inet.tcp.sack.enable	0-1
Windows XP Version 2002 Service Pack 3	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\EnablePMTUDiscovery	1
	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\SackOpts	0-1
	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Tcp1323Opts	3
	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\GlobalMaxTcpWindowSize	5242880
	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\TcpWindowSize	524288
	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Afd\Parameters\DefaultSendWindow	524288
	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Afd\Parameters\DefaultReceiveWindow	524288
Ubuntu 11.04	net.ipv4.tcp_no_metrics_save	1
	net.ipv4.tcp_moderate_rcvbuf	1
	net.core.rmem_max	5242880
	net.core_wmem_max	5242880
	net.ipv4.tcp_rmem	4096 524288 5242880
	net.ipv4.tcp_wmem	4096 524288 5242880
	net.ipv4.tcp_sack	0-1
	net.ipv4.tcp_timestamps	0-1

DEF filtering prevents DUPACKs from arriving to the sender. However, other information on the DUPACKs are also filtered out: SACK blocks and timestamps. Availability of these information may increase the performance of the TCP sender. Thus, we executed our tests when SACK/Timestamps are enabled/disabled to see whether DEF/RR performance is affected or not.We have presented results for SACK and timestamps are enabled (SACK+TS), SACK and timestamps are disabled so that only cumulative ACKs are used (CACK), only SACK is enabled (SACK), and only timestamps is enabled (CACK+TS).

DEF and RR modules are located at Node2 and TCP data packets are distributed to the multiple paths there. TCP data packets are handed by the RR scheduler to the neighbors of Node2 within the local backbone. We collected the results for DEF as well as RR scheduling of packets to the multiple paths.

As shown in Figure 6.3, TCP receiver downloads data from the TCP sender for 600 seconds. TCP goodput is calculated by dividing downloaded amount of data to the download time. Goodput samples are collected till 95% of the values are within the ±5%vicinity of the average goodput (at least 3 samples are collected).

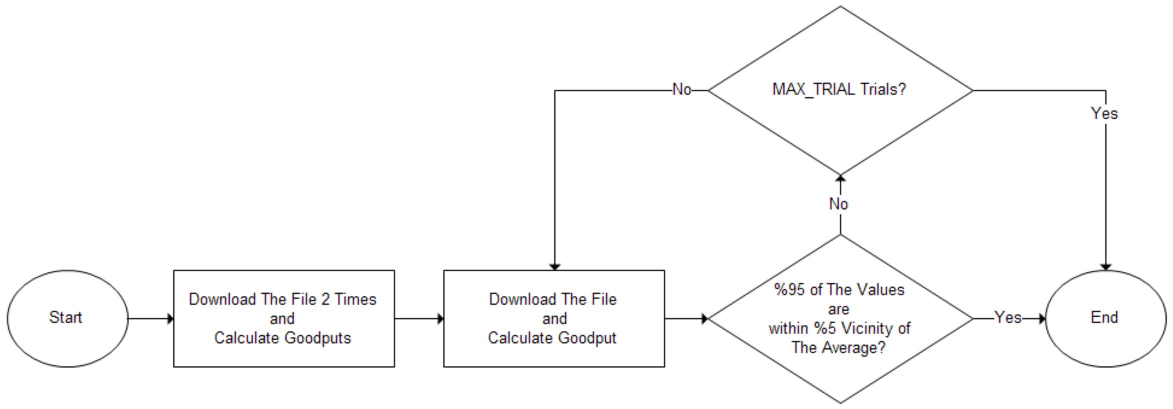

MAX_TRIAL Trials?
No
Yes
No
Start
Download The File 2 Times and Calculate Goodputs
Download The File and Calculate Goodput
%95 of The Values are within %5 Vicinity of The Average?
Yes
End

Figure 6.3: Average TCP Goodput Calculation for The Experiments

6.1 Experiment Results

We calculated TCP Goodput percentages with respect to single path (i.e., neither RR nor DEF is used) goodputs (≈1.40 Mbps for each OS) as:

((DEF|RR Goodput) / (Single Path Goodput)) * 100

We present results in Figure 6.4, 6.5, and 6.6. As expected [12], the Windows XP and PC-BSD TCP goodputs degrades sharply in RR with the increasing path RTT difference and it is worse than the single path case when there is even a small path RTT difference (e.g., 40 ms). On the other hand, DEF gets better TCP goodputs than single path use (i.e., TCP data transfer over one path, without DEF or RR) in all cases.

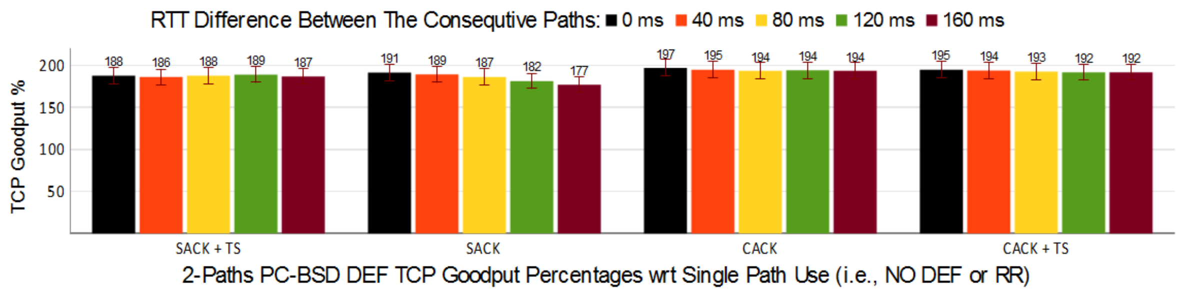
We observed 76% to 98% increase in the TCP goodput when 2 paths are used and 125% to 197% increase when 3 paths are used, as compared to the single path use. Because of the network dynamics, we see some fluctuations in DEF performance (not more than 10% when 2 paths are used and not more than 20% when 3 paths are used, between the best and the worst case for a given OS). It is caused by the buffering in the network which influences accuracy of the DEF DUPACK estimations, especially when number of used paths increases and equal-RTT paths are used.

The only exception of performance decrease in the RR regime is Linux OS. Ubuntu TCP performance is not affected since it implements algorithms to deal with out-of-order packet receptions as described in [14]. The algorithm mainly works as follows. Before a packet is re-transmitted, current ssthresh value is stored before it is changed. Timestamps and D-SACK are used by Linux TCP sender to detect spurious retransmissions:

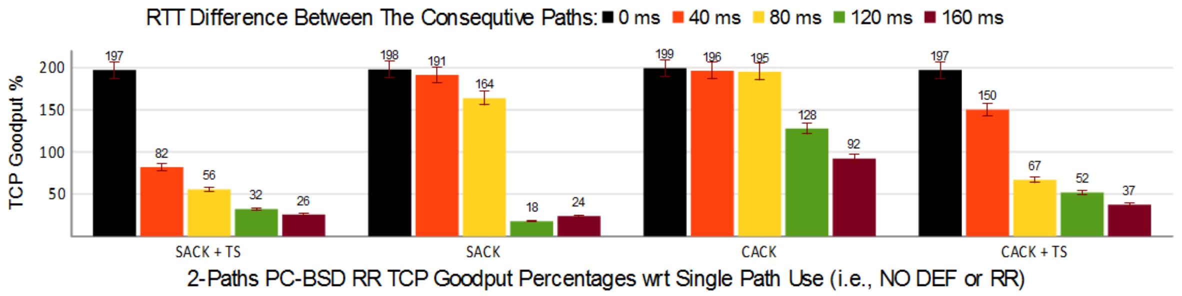
- D-SACK may report that original transmission of the retransmitted segment has already arrived at the receiver
- ACK carries timestamp of the first segment sent, instead of the retransmitted segment

If a spurious retransmission is detected, ssthresh is reverted to stored value. In addition, Linux reordering estimator adapts dupthresh value based on the number of spurious retransmissions (i.e., the threshold value is not always 3 and adjusted based on the reordering in the network).

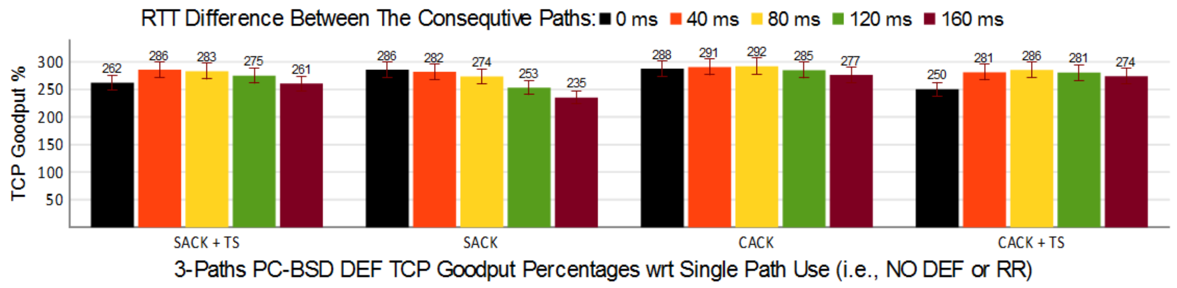
However. even though Linux has its own mechanisms to cope with the out-of-order packets, the results show that DEF did not harm the performance of Linux.



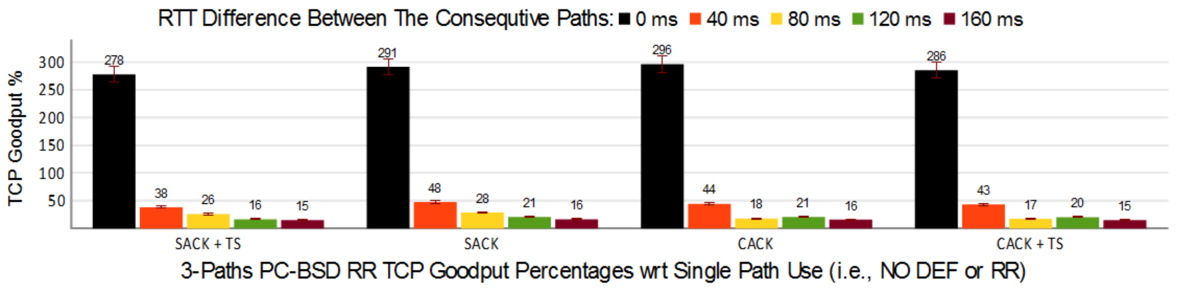
(a) PCBSD 2-Paths DEF Results



(b) PCBSD 2-Paths RR Results

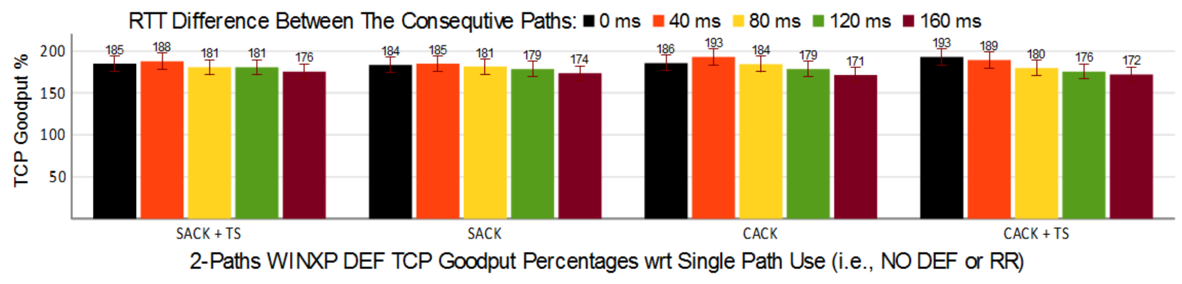


(c) PCBSD 3-Paths DEF Results

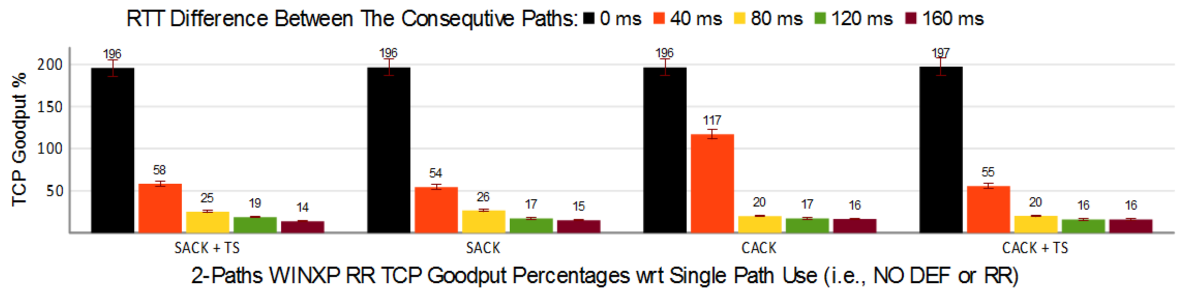


(d) PCBSD 3-Paths RR Results

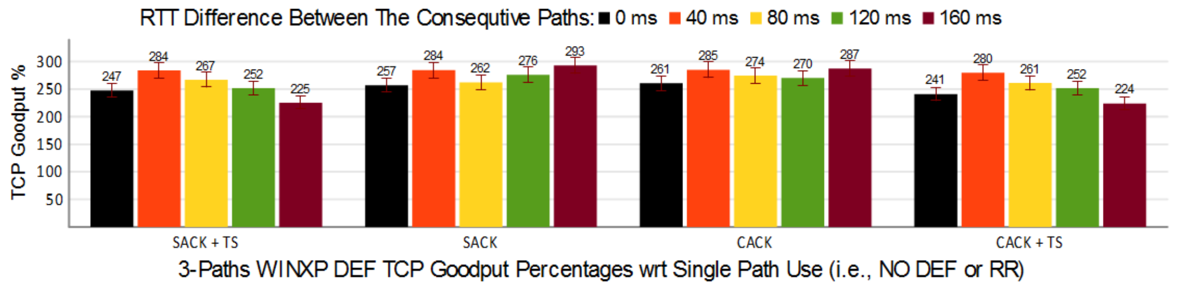
Figure 6.4: PC-BSD 8.2 Results



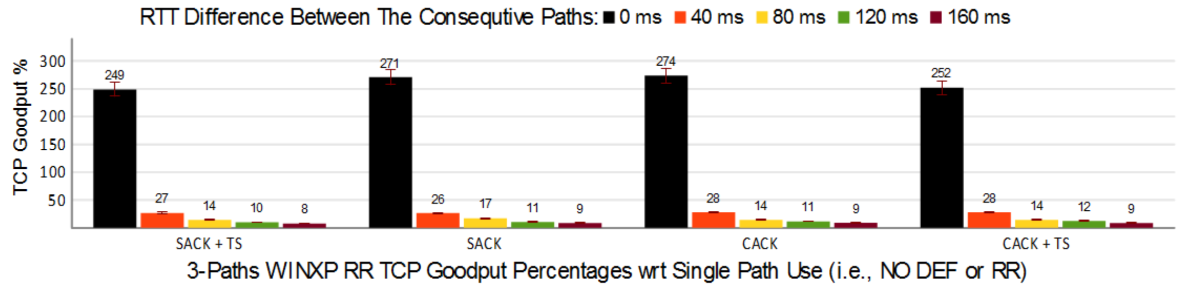
(a) Windows XP 2-Paths DEF Results



(b) Windows XP 2-Paths RR Results

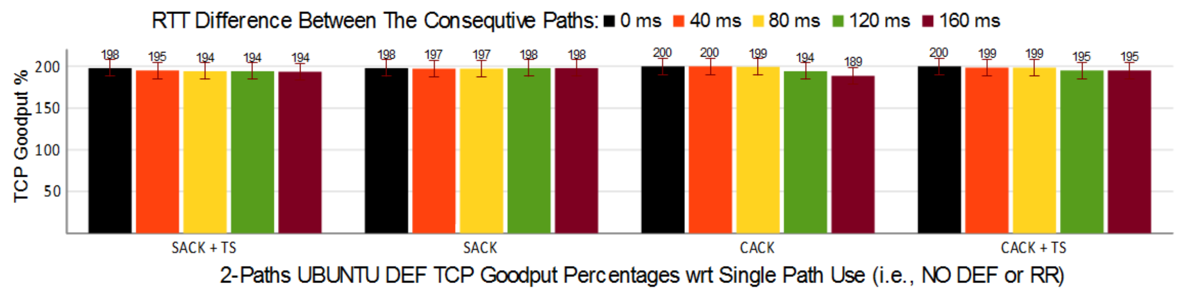


(c) Windows XP 3-Paths DEF Results

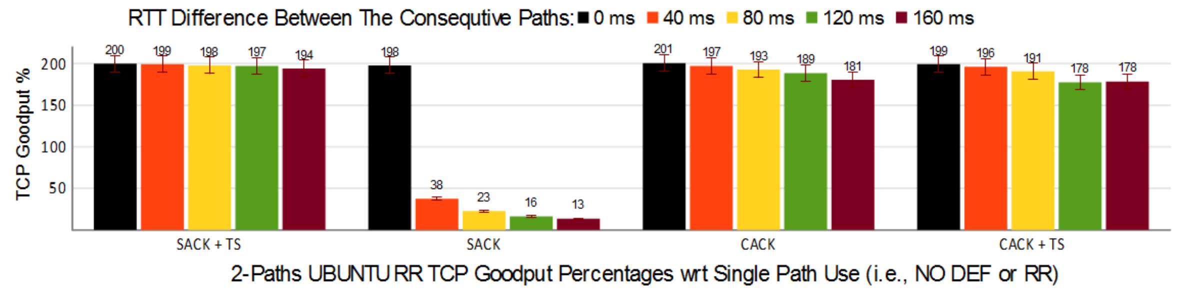


(d) Windows XP 3-Paths RR Results

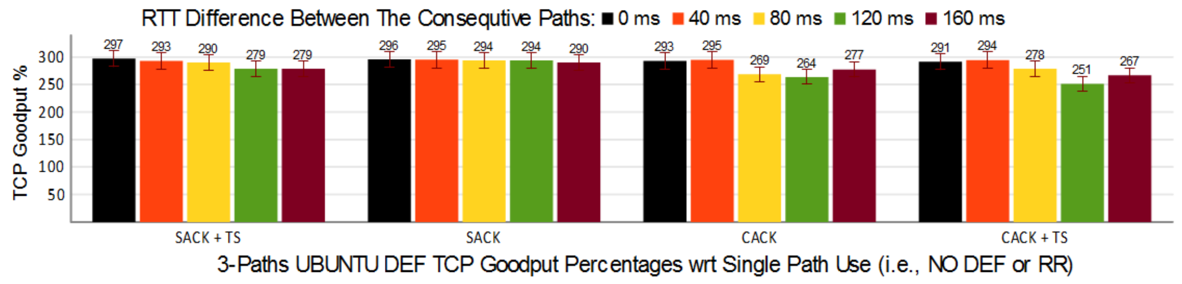
Figure 6.5: Windows XP Results



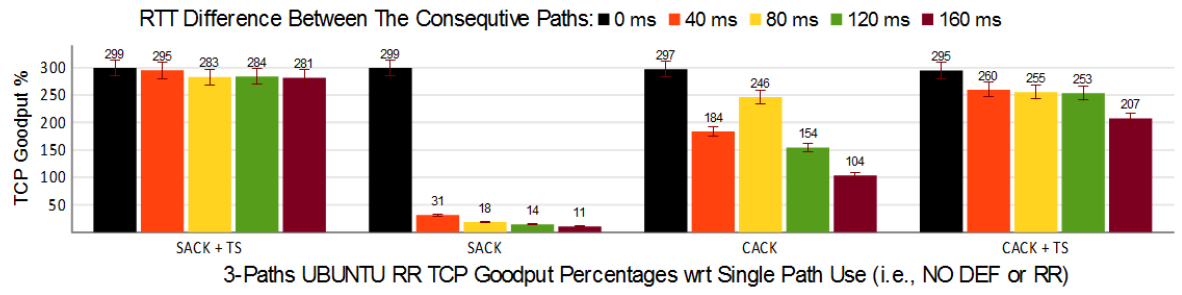
(a) Ubuntu 2-Paths DEF Results



(b) Ubuntu 2-Paths RR Results



(c) Ubuntu 3-Paths DEF Results



(d) Ubuntu 3-Paths RR Results

Figure 6.6: Ubuntu 11.04 Results

Chapter 7

Summary and Future Work

In this technical report, we investigated the possibility of distributing TCP flow packets over multiple paths.

Since TCP design assumes that its packets flow over a single-path within the underlying network, its performance is affected negatively in case of multi-path use. In Chapter 2, we investigated the problems that TCP faces with when its segments are split over multiple paths and presented the solutions of other researchers for these problems.

We defined an architecture, *splitter/combiner architecture (SCA)*, that may be used to develop agnostic TCP solutions over multiple paths in Chapter 3. The architecture is transparent to TCP end-points and protocol layers. Thus, no modification on end-systems is required to benefit from multiple path usage. The implementation of the architecture may be realized between every protocol layers below transport layer, even directly above the physical layer. Although SCA is not designed to be located on TCP end-points, its protocol transparency allows its instances, *SCA Proxies (SCAPs)*, to be used on the end-systems. In that case, it may integrate some mechanisms of the existing solutions or allow development of new solutions.

In Chapter 4, we discussed three stages for the deployment of SCAPs: (1)single, (2)paired, and (3)chained SCAP deployment. Since at the initial SCAP deployment stage it will be hard to find other SCAPs, there may be only one SCAP in between the TCP sender and receiver. In that case, SCAP must work standalone. At a later stage, it will be easier to find SCAP peers which allows them to benefit from the signaling information exchange. In future, we expect high deployment and SCAPs that work as cascaded.

Chapter 5 presents a sample single SCAP which increases TCP performance over multiple paths. The SCAP may work alone and demonstrates the applicability of our ideas. It differentiates between packet-loss and RTT-difference based DUPACKs and filters the RTT-difference-based DUPACKs.

In Chapter 6, simulation based experiments are executed and results are collected to test the SCAP. The results show the potential in using our architecture to develop TCP multipath solutions without end-system modifications, which is necessary for high deployments.

Based on the obtained results, we continue our work on single SCAP and plan to extend our scope to SCAP pairs and chains.

Bibliography

[1] J. Postel, "Internet Protocol," RFC 791, September 1981.

[2] R. Braden, "Requirements for Internet Hosts –Communication Layers," RFC 1122, October 1989.

[3] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323, May 1992.

[4] W. Simpson, "IP in IP Tunneling," RFC 1853, October 1995.

[5] C. Perkins, "IP Encapsulation within IP," RFC 2003, October 1996.

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, TCP Selective Acknowledgement Options, RFC 2018, October 1996.

[7] G. Malkin, "RIP Version 2," RFC 2453, November 1998.

[8] K. McCloghrie and F. Kastenholz, "The Interfaces Group MIB," RFC 2863, June 2000.

[9] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," RFC 2883, July 2000.

[10] S. J. Lee and M. Gerla, "Split Multipath Routing with Maximally Disjoint Paths in Ad Hoc Networks," In Proceedings of The IEEE International Conference on Communications 2001 (ICC '01), Volume 10, pp. 3201-3205, June 11-14 2001.

[11] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," IEEE Journal on Selected Areas in Communications, Volume 19, Number 7, pp. 1300-1315, July 2001.

[12] M.Gerla, S. S. Lee, and G. Pau, "TCP Westwood Performance Over Multiple Paths," UCLA CSD Technical Report #020009, January 2002. Available at: `http://nrlweb.cs.ucla.edu/publication/download/140/2002-tr-0.pdf`.

[13] K. Chebrolu and R. Rao, "Communication using Multiple Wireless Interfaces," In Proceedings of The IEEE Wireless Communications and Networking Conference 2002 (IEEE WCNC 2002), Volume 1, pp. 327-331, March 17-21 2002.

[14] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP," Proceedings of Usenix 2002/Freenix Track, pp. 49-62, Monterey, CA, USA, June 2002.

[15] Y. Lee, I. Park, and Y. Choi, "Improving TCP Performance in Multipath Packet Forwarding Networks," Journal of Communication and Networks (JCN), Volume 4, Number 2, pp. 148-157, June 2002.

[16] S.Y. Wang, C.L. Chou, C.H. Huang, C.C. Hwang, Z.M. Yang, C.C. Chiou, and C.C. Lin, "The Design and Implementation of the NCTUns 1.0 Network Simulator," Computer Networks, Volume 42, Issue 2, pp. 175-197, June 2003.

[17] K. Chebrolu, "Multi-Access Services in Heterogeneous Wireless Networks," PhD Thesis, ECE Department, U.C. San Diego, May 2004. Available at: `http://home.iitk.ac.in/~chebrolu/docs//thesis.pdf`.

[18] J. Chen, K. Xu, and M. Gerla, "Multipath TCP in Lossy Wireless Environment," In Proceedings of The Third Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net 2004), Bodrum, Turkey, pp. 263-270, June 27-30 2004.

[19] H.-Y. Hsieh and R. Sivakumar, "A Transport Layer Approach for Achieving Aggregate Bandwidths on Multi-homed Mobile Hosts," ACM/Springer Wireless Networks Journal, Volume 11, Number 1-2, pp. 99-114, January 2005.

[20] R. Raghunarayan, "Management Information Base For The Transmission Control Protocol (TCP)," RFC 4022, March 2005.

[21] K.-H. Kim and K. G. Shin, "Improving TCP Performance over Wireless Networks with Collaborative Multi-homed Mobile Hosts," In Proceedings of USENIX/ACM International Conference on Mobile Systems, Applications, and Services (MobiSys), Seattle, WA, USA, June 2005.

[22] K. Chebrolu, B. Raman and R. R. Rao, "A Network Layer Approach to Enable TCP over Multiple Interfaces," ACM/Kluwer Journal of Wireless networks (WINET), September 2005.

[23] R. Karrer and E. Knightly, "TCP-PARIS: A Parallel Download Protocol for Replicas," In Proceedings of IEEE International Workshop on Web Content Caching and Distribution (WCW 2005), Sophia Antipolis, France, pp. 15-25, September 12-13 2005.

[24] F.P. Kelly, and T. Voice, "Stability of End-to-End Algorithms for Joint Routing and Rate Control," ACM/SIGCOMM CCR 35(2) (2005).

[25] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271, January 2006.

[26] S. Routhier, "Management Information Base for the Internet Protocol (IP)," RFC 4293, April 2006.

[27] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant and D. Towsley, "Multi-Path TCP: A Joint Congestion Control and Routing Scheme to Exploit Path Diversity on the Internet," IEEE/ACM Transactions on Networking, Volume 14, Issue 6, December 2006.

[28] K.-C. Leung, V. O. K. Li, and D. Yang, "An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges," IEEE Transactions on Parallel and Distributed Systems, Volume 18, Issue 4, pp. 522-535, April 2007.

[29] M. Mathis, J. Heffner, and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, May 2007.

[30] L. Stewart and J. Healy, "Tuning and Testing the FreeBSD 6 TCP Stack," CAIA Technical Report 070717B, July 17 2007. Available at: `http://caia.swin.edu.au/reports/070717B/CAIA-TR-070717B.pdf`.

[31] K.-H. Kim and K. G. Shin, "PRISM: Improving the Performance of Inverse-Multiplexed TCP in Wireless Networks," IEEE Transactions on Mobile Computing, Volume 6, Issue 12, pp. 1297-1312, December 2007.

[32] J. He and J. Rexford, "Toward Internet-Wide Multipath Routing," IEEE Network , Volume 22, Number 2, pp.16-21, March-April 2008.

[33] B. Radunovic, C. Gkantsidis, D. Gunawardena, and P. Key, "Horizon: Balancing TCP over Multiple Paths in Wireless Mesh Network," In Proceedings of The 14th Annual International Conference on Mobile Computing and Networking (MobiCom 2008), San Fransisco, California, USA, 14-19 September 2008.

[34] K. Evensen, D. Kaspar, P. Engelstad, A. F. Hansen, C. Griwodz, and P. Halvorsen, "A Network-Layer Proxy for Bandwidth Aggregation and Reduction of IP Packet Reordering," In Proceedings of The IEEE 34th Conference on Local Computer Networks (LCN 2009), pp. 585-592, Zurich, 20-23 October 2009 .

[35] I. A. Ganichev, B. Dai, P. B. Godfrey, and S. Shenker, "YAMR: Yet Another Multipath Routing Protocol," EECS Department, University of California, Berkeley, Technical Report No. UCB/EECS-2009-150, 30 October 2009, Available at: `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-150.pdf`.

[36] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, September 2009.

[37] S.Y. Wang and R.M. Huang, "NCTUns Tool for Innovative Network Emulations," Chapter 13 of The "Computer-Aided Design and Other Computing Research Developments" Book, ISBN: 978-1-60456-860-8, Published by Nova Science Publishers in 2009, Available at: `http://nsl10.csie.nctu.edu.tw/products/nctuns/NovaNCTUnsEmulationNew2009.pdf`.

[38] A. Kostopoulos, H. Warma, T. Leva, B. Heinrich, A. Ford, and L. Eggert, "Towards Multipath TCP Adoption: Challenges and Opportunities," 6th EURO-NF Conference on Next Generation Internet (NGI), pp. 1-8, Paris, June 2-4 2010.

[39] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," 8th USENIX conference NSDI 2011, March 2011.

[40] P. Yang, W. Luo, L. Xu, J. Deogun, and Y. Lu, "TCP Congestion Avoidance Algorithm Identification," In Proceedings of The IEEE 31st International Conference on Distributed Computing Systems (ICDCS 2011), pp. 310-321, Minneapolis, June 20-24 2011.

[41] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," IETF Internet Draft, draft-ietf-mptcp-multiaddressed-06 (work in progress), January 2012.

[42] C. Raiciu, M. Handly, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356, October 2011.

[43] M. Scharf and A. Ford, "MPTCP Application Interface Considerations," IETF Internet Draft, draft-ietf-mptcp-api-03, 30 November 2011.

[44] NetFilter Web Site, `http://www.netfilter.org/`.

[45] NCTUns Web Site, `http://nsl.csie.nctu.edu.tw/nctuns.html`.

[46] Enabling High Performance Data Transfers [PSC], `http://www.psc.edu/networking/projects/tcptune`.

[47] http://fasterdata.es.net/fasterdata/host-tuning/

[48] T. Ayar, B. Rathke, Ł. Budzisz, and A. Wolisz, "TCP over Multiple Paths Revisited: Towards Transparent Proxy Solutions," accepted to the IEEE International Conference on Communications 2012 (IEEE ICC'12), June 2012.