# Project Report

## Implementing T/TCP over a Network

Nidhi Dhamnani (CS15BTECH11028), Durga Keerthi (CS15BTECH11024)

## Abstract

In this project we implemented T/TCP using UDP socket and simulated routing protocol to demonstrate the process of complete data transfer in a network. We compared the performance of T/TCP with TCP.

Keywords: T/TCP, TCP, Routing Protocols

## 1. Introduction

### 1.1 T/TCP - TCP Extensions for Transactions

This variant of TCP aims at improving the client/server (request/response) communication where the messages exchanged are of type transactions (typically small messages). Considering that an explicit open and close of TCP connection causes overhead for these type of messages, this variant merges all of them into one. That is, a request from client to server will open connection, get the work done by server and closes the connection in just one go. To achieve this, the specification says that the time elapsed in, (i)performing three way handshake for establishing connection and (ii)time_wait phase while closing connection needs to be reduced.

- 32-bit number called "connection count" (CC) is introduced in a TCP option in each segment.
- A mechanism TCP Accelerated Open (TAO) is followed to bypass the threeway handshake.
- In case TAO is not possible, this protocol always falls back to TCP to allow the backward compatibility
- From the experimental summaries as mentioned here, TIME-WAIT state for T/TCP may be shortened to 12 seconds.

Behaviour of TAO, truncation of time_wait and usage of connection count are elaborated in design part (section 2.1).

### 1.2 Routing Protocols

Routing protocols are used for communication between two routers and to transfer the data from source to destination. It determines the choice of path through which we should send the packet such that the overall cost for sending the packet is minimum. We implemented two routing protocols namely Link State and Distance Vector.

- Link State: In this algorithm, the network topology is known to all the routers and then each router individually build it's routing table by calculating the minimum path to all the other node.

  Pseudo Code:

  ```
  Initialization:
  N={routerID}
  For all other node v {
          If (v is adjacent to routerID)
                  Dist[v]=cost[u][v]
          Else
                  Dist[v]=INT_MAX
  }
  while(Not all nodes are in N) {
          If (w notIn(N) && Dist(w) is min) {
                  Add w to N
                  For all nodes v adjacent to w && notIn(N)
                          Dist[v]=min(D[v], D[w]+cost[w][v]
          }
  }
  ```

- Distance Vector: In this algorithm, each node sends it's table to it's neighbours. When the node receives the table from it's node, it checks it's own table using Bellman-Ford algorithm.

  Pseudo Code:

  ```
  Whenever x receives new table from neighbour x

  For all nodes y {
          If (cost[x][y] > cost[z][y] + Dist[x][z])
                  cost[x][y] = cost[z][y] + Dist[x][z]
  }
  ```

# 2. Design

## 2.1 T/TCP

This implementation of T/TCP follows the experimental protocol as mentioned here.
Structure of the packet is same as that of TCP except for the additional fields. I defined the packet as struct containing the fields as follows:

```
struct packet{
  int src_port;
  int dest_port;
```

```
    int seq;
    int ack;
    int headerlen;
    int options;
    int window_size;
    int checksum;
    int connection_count;
    int cc_option;
    int msglen;
    int packnum;
    char text[32];
};
```

And the options field in actual TCP header is replaced by enum fields like syn, fin, syn_ack etc.

This provides reliable communication by resending the packets if ack is not received in fixed time. Checks if messages are not corrupted. Adds flow control by sending the buffer size at server back to client (however this won't be effective, as we are having finitely large buffer at server).

Establishing the connection:

| **Client** | **Server** |
|---|---|
| Sends packet with<br>syn_fin option,<br>Data<br>the previous connection count and cc as cc_option<br>if already had a connection<br>Else cc_new as cc_option | |
| | checks if ccc value sent is > the value server stored in the previous |
| connection | |
| | if satisfies, data is processed and connection is closed, syn_fin_ack is sen<br>otherwise three way handshake is performed to establish connection<br>Sends syn_ack |

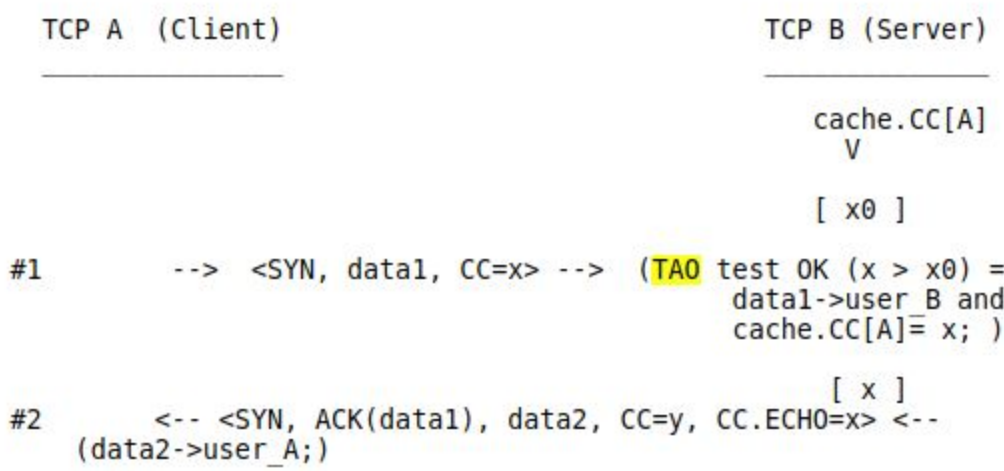


Figure 1. TAO: Three-Way Handshake is Bypassed

In our server-client system, server records what the client sends to it. And only replies ack, if the message is in sequence and not corrupted.
Though server responds to different requests from client, the ones used are limited for demonstration purpose.

Client sends a packet with cc_option as cc_new when it first establishes the connection, server replies with three-way handshake as in the following figures. After this sender sends the packets with syn_fin field which implies that it want to establish the connection, get the data processed and close the connection. So the server performs TAO and it will give true as the client and server already had a connection. Server processes the data, calculates checksum, records the data, in line and then replies accordingly.
Since the client already sent fin, Server send fin_ack, fin. Client then sends last_ack which closes the connection.

A buffer is maintained at sender which implies that the sender can't send more or have to wait till the server acknowledges.

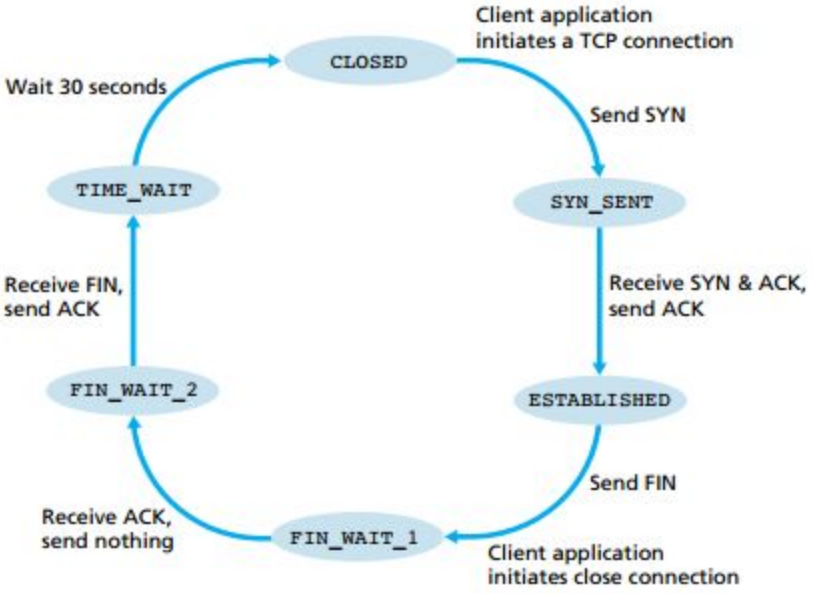


**Figure 3.41** ♦ A typical sequence of TCP states visited by a client TCP

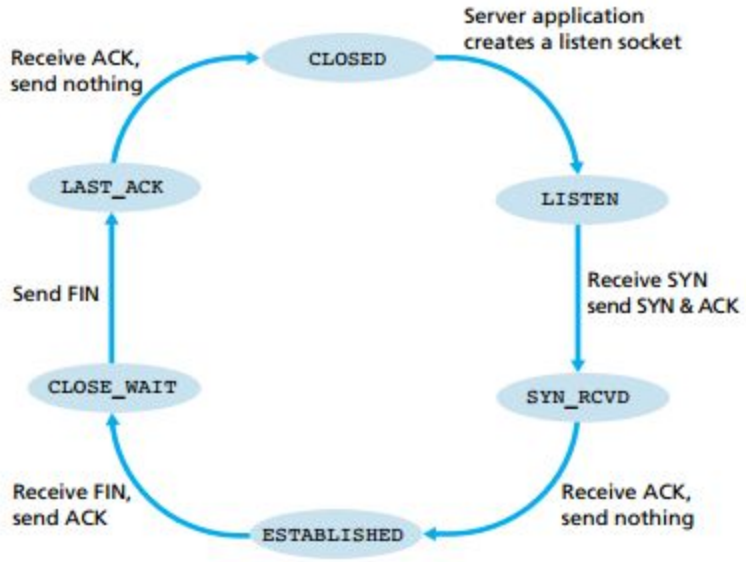


**Figure 3.42** ♦ A typical sequence of TCP states visited by a server-side TCP

## 2.2 Routing Protocols

Each router knows the list of routers to which it is connect and the corresponding link weight to that router. The more the link weight, the higher the cost, therefore the aim is to find the path to all the routers with minimum cost. We assumed the network to be a connected graph.

### 2.2.1 Distance Vector

In distance vector, a node sends it's table only to it's neighbours and all the changes are propagated in the network.

Implementation in c++ :

We created a socket to send and receive table from neighbours. In distance vector the table is updated dynamically so we created threads to send, update and receive the messages. The send function sends the updated routing table to the neighbours. The receive function gets a packet and stores it in a queue. The update function pops a table from the queue and updates it's table. If the table is modified, it again sends the updated table to it's neighbours.

Handling Concurrency Issues: To handle the concurrency issues during receiving and updating the table, we acquire lock while accessing the queue.

Getting the routing path: We maintain router ID of node which is sending the packet. Let us say node the network topology is x--y--z--w, where x, y, z, w are routers and "--" represent edges. Node x will receive update from Node y and x will update it's table. Therefore the router ID from which x receives the packet is y but y is not directly connected to w. Hence to get the correct path we maintain updatedFrom variable which stores the actual value from where the table originated. Therefore the changes are propagated in the network and the path is updated accordingly.

Timer: After certain amount of time, the all the shortest paths are correctly determined and the tables don't get updated. Therefore to then start routing the packets we use a timer.

### 2.2.2 Link State

In link state algorithm, each router broadcasts the it's routing table in the network. All the other routers listen to the broadcast messages and get the table of other routers.

Implement in c++ :

We created a socket to receive the broadcast messages. The nodes listen on 0.0.0.0 to get the table. The broadcast messages are sent on broadcast address of the subnet. In link state, we need to send and

receive the tables concurrently, therefore we created threads for sending and receiving. The send function sends the routing table to the neighbours. The receive function gets a packet and stores it in a queue. We receive till we get tables from all the routers, once we get the routing tables from all the neighbours, we run the routing protocol.

Getting the routing path: We maintain router ID of node which is sending the packet. Therefore while processing the packets, we check the node which sent the packet and assign that as the parent. In this as we receive the table from all the nodes, we can get the correct parent and we need not maintain extra variables.

Timer: In this case, we don't need any timer. Once we receive the messages from all the nodes in the network, we calculate the path.

2.2.3 Packet Forwarding

Once we get the updated tables and paths to all the other routers in the subnet, we start sending the messages. We create server & client and messages requests are send from client to server and the server responds to them. All these routing algorithms can run inside the client and server or we can run inside a router.

- Case-1: Router in inside server and client - In this case the router checks the destination address with it's IP address and if both are same then it processes the packet, else it checks the next hop address and destination address and forwards the packet by looking into it's routing table.
- Case-2: Router is not inside client and server - In this case the router checks for next hop address and destination address and forwards the packet by looking up in the routing table.

# 3. Modules

## 3.1 Durga Keerthi (CS15BTECH11024)

Libraries used:

- Standard C++11 libraries

Deliverables: In Folder TransactionTCP

- ttcp_server.cpp
- ttcp_client.cpp
- tcp_server.cpp
- tcp_client.cpp

## 3.2 Nidhi Dhamnani (CS15BTECH11028)

Libraries used:

- Standard C++11 libraries

Deliverables: In folder Routing

- LinkState.h - Code for link state algorithm
- DistanceVector.h - Code for distance vector algorithm
- Main.cpp - Code for simulating routing protocols
- Test0, Test1, Test2, Test3, Test4 - Folders consisting files for testing

NOTE: Readme is included in both the folders. It contains commands to run the code.

## 4. Results

- The RTT taken by TTCP for first connection is same as TCP.
- For the subsequent transactions, the time taken by T/TCP < TCP. In some sense, they are not comparable, to send a transaction data to server, TCP performs connection establishment, data exchange and then it closes the connection in different packets . TCP may not close the connection and reopen it in the actual case.
- The packet loss handling in both are same therefore the no. of packet losses is almost same.

## 5. References


- T/TCP: https://tools.ietf.org/html/rfc1644
- T/TCP: http://d3s.mff.cuni.cz/seminar/download/2000-05-31-kuhn-ttcp.html
- Routing Protocols: http://web.csulb.edu/~rlaster/docs/cecs572.pdf
- Routing Protocol: https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/