ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea (Triennale) in Ingegneria e Scienze Informatiche

Il Protocollo MultiPath TCP

Tesi di laurea in
(Reti di Telecomunicazioni)

Relatore
Prof. Franco Callegati

Candidato
Riccardo Squarcialupi

2° Sessione di Laurea
Anno Accademico 2020-2021

Sommario

Ad oggi, il Transmission Control Protocol (TCP) è il protocollo più popolare per trasmettere e consegnare informazioni su Internet in modo affidabile. Tuttavia, esso fa uso di una connessione a percorso singolo, non sfruttando il multihoming e i percorsi multipli che sono sempre più disponibili per i dispositivi endpoint, cioè dispositivi mobili e server in configurazioni ad alta resilienza. Il MultiPath TCP (MPTCP) è stato sviluppato per affrontare queste limitazioni. Esso, infatti, mira a fare uso della diversità dei percorsi, al fine di offrire una migliore connettività di rete complessiva, aumentando così la resilienza ai guasti, eseguendo il bilanciamento del carico tra i percorsi disponibili (quando più di uno è disponibile). Consente, inoltre il supporto multihoming, senza la necessità di modificare i dispositivi già esistenti attualmente sparsi sulla rete.

Alla mia famiglia, ai miei amici, sisghè.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

L'evoluzione dei dispositivi portatili, come telefoni cellulari, tablet e notebook, ha reso importante essere sempre raggiungibili e avere una connessione ad alta velocità, dal momento che, la maggior parte delle applicazioni critiche, che girano su questi dispositivi, distribuiscono il loro calcolo su sistemi cloud ricorrendo a centri dati sparsi in tutto il mondo.
Allo stesso tempo, molti dispositivi hanno sviluppato la capacità di connettersi a Internet con almeno due diverse interfacce in ogni tipo di dispositivo, come WiFi e 3G, o Ethernet e WiFi, al fine di ottimizzare le infrastrutture di comunicazione disponibili. D'altra parte, i centri dati sparsi nel mondo di solito supportano il multihoming, essendo collegati a due o più reti per migliorare la resilienza dei servizi forniti. Per molti anni il Transmission Control Protocol (TCP) è stato un componente fondamentale dello stack protocollare di Internet e il protocollo di comunicazione più affidabile per la trasmissione dei dati, ma permette solo un unico percorso tra una fonte e una destinazione.
Anche se il modello base del TCP comprende i meccanismi essenziali necessari per controllare il flusso e la congestione, da solo non assicura la consegna in tempo reale in caso di connessioni critiche, di congestione o di interruzioni. Con la crescente mobilità dei dispositivi, con applicazioni ubique e critiche, è diventato molto importante avere connessioni affidabili.
Internet sta arrivando a un punto in cui il forte aumento del numero di utenti, fornitori e servizi sta cominciando a stressare la sua scalabilità, è quindi importante adattare i protocolli esistenti per rendere possibile esplorare i benefici che possono derivare dalla connettività multipath. Con l'obiettivo di aggirare le limitazioni del protocollo TCP e aumentare l'affidabilità delle connessioni, molti autori hanno proposto diversi approcci, come Stream Control Transmission Protocol (SCTP), Concurrent Multi-Path Stream Control Transmission Protocol (CMP-SCTP), multiple paths TCP (mTCP), Parallel TCP (pTCP), Shim6 Protocol e altri. Tutti questi protocolli hanno limitazioni che li rendono difficili da distribuire su Internet,

come vedremo più tardi.
Recentemente l'Internet Engineering Task Force (IETF) ha creato un gruppo di lavoro per lo sviluppo di uno standard per il protocollo multipath a livello di trasporto che può essere facilmente implementato. Considerato ciò, è stata proposta un'estensione per il protocollo TCP, il MultiPath TCP (MPTCP), dove ogni connessione tra due punti può effettivamente utilizzare più percorsi paralleli, utilizzando il rilevamento della congestione per determinare il percorso effettivo da seguire. Come vedremo, MPTCP ha il potenziale per aumentare il throughput, l'affidabilità e la flessibilità delle connessioni. Il fatto che sia un'estensione del TCP e mantenga la retrocompatibilità con quest'ultimo, lo rende più facile da distribuire su Internet rispetto ad altre proposte precedenti.

1.1 Motivazioni

La motivazione per implementare un protocollo TCP multipath è quella di migliorare la robustezza e le prestazioni delle connessioni end-to-end. Questa soluzione permette l'uso di percorsi multipli da parte della stessa connessione TCP, per massimizzare l'uso delle risorse, aumentare la ridondanza e la resilienza. Tale protocollo offre capacità di multihoming, ricorrendo all'uso delle diverse interfacce di rete disponibili sui dispositivi attuali, fornendo un migliore throughput. Per i dispositivi mobili, questo protocollo può permettere l'handover di connessione senza problemi, senza perdere la connettività delle applicazioni.

Capitolo 2

Soluzioni Multipath e Multihoming

Questo capitolo esamina i diversi approcci dell'IETF per gestire scenari multipath e multihoming, e le possibili applicazioni.

2.1 Approcci multipercorso

Nel corso del tempo, c'è stato un certo dibattito su quale approccio debba essere adottato per fornire un protocollo che possa beneficiare dell'uso simultaneo di più percorsi, in modo da sfruttare le risorse disponibili nella rete, cercando di rendere possibile l'uso efficiente di più interfacce per garantire una connettività costante sui dispositivi mobili.
I ricercatori sostengono diverse soluzioni di implementazione. alcuni ritengono che l'implementazione dovrebbe essere effettuata a livello di rete, per fornire diversi benefici. Altri invece hanno implementato la loro soluzione ricorrendo ad applicazioni specifiche, che distribuiscono parti di file in diversi peer, preoccupandosi solo di aumentare il throughput; nelle reti locali, é possibile implementarlo a livello di collegamento per esplorare le diversità di percorso locali. Un'implementazione al livello di trasporto, invece, beneficia della trasparenza a livello di rete e fornisce una relativa indipendenza a livello di applicazione, evitando il requisito di dover cambiare i ben stabiliti livelli di collegamento e di rete. D'altra parte, poiché MPTCP è un'estensione di TCP, le applicazioni a percorso singolo di quest'ultimo possono essere utilizzate senza alcun cambiamento nel livello di applicazione. Nelle seguenti sottosezioni, si analizzano in dettaglio alcuni di questi approcci alternativi.

2.1.1 Approccio a livello di collegamento

Una soluzione per fornire multipath a livello di link è quella di usare il Link Aggregation Control Protocol (LACP). Questo protocollo usa un approccio di bundling a livello di link. Il link bundling avviene tramite l'aggregazione delle porte dello switch, al fine di utilizzare più connessioni di rete in parallelo per aumentare il throughput e creare ridondanza in caso di guasto del collegamento. Una soluzione che sfrutta uno schema multipath a livello di link è lo Shortest Path Bridging (SPB). Questo sistema supporta una soluzione molto più ampia di topologie di livello due, che possono essere utilizzate nei nodi del centro dati, ma non può fare uso di più interfacce disponibili.
Un'altra implementazione dell'approccio multipath è stata suggerita per il livello di collegamento per ottenere un throughput più elevato nelle Wireless Mesh Networks (WMN): questa soluzione ha bisogno di implementare un multi-canale a livello di collegamento in combinazione con il multi-path routing, al fine di instradare in modo efficiente e intelligente il traffico ed ottenere un migliore throughput in queste reti.
Il livello di collegamento è responsabile del canale e dello scheduling dei pacchetti: il primo è usato per controllare in quale canale l'informazione sarà ricevuta e il secondo per definire quando inviare i pacchetti. Il multipath-routing è responsabile della selezione dei due migliori percorsi verso il gateway. Una soluzione a questo livello ha un grande potenziale per ottenere una buona performance e un più alto throughput end-to-end per questo tipo di reti, ricorrendo alla decomposizione del traffico attraverso canali diversi, programmando tempi diversi e infine utilizzando percorsi diversi. Le soluzioni a livello 2 multipath sono per lo più limitate alle reti locali.

2.1.2 Approccio a livello di rete

L'implementazione di un protocollo TCP multipath a livello di rete sembra naturale. In questo caso abbiamo una singola connessione a livello di trasporto e i pacchetti sono divisi in diversi flussi. Il bilanciamento del carico viene eseguito a livello di connessione e non a livello di pacchetto.
Questa soluzione offre solo una singola connessione a livello di trasporto. In questo modo, il throughput sarà dettato dal link più congestionato: inoltre, può mitigare le ritrasmissioni non necessarie al livello di trasporto a causa del riordino dei pacchetti, causando una significativa riduzione del throughput della connessione. Una delle cause di questa ritrasmissione inutile, si verifica perché la maggior parte dei dispositivi di rete sparsi in Internet non supporta e non riconosce il traffico generato. Per implementare una soluzione multipath a livello di rete sarebbe ne-

cessario ricorrere a un aggiornamento totale dei dispositivi di rete attualmente in uso su Internet.

2.1.3 Approccio a livello di trasporto

L'implementazione di un protocollo multipath al livello di trasporto ha la possibilità di raccogliere informazioni come capacità, latenza e lo stato di congestione in ogni percorso utilizzato. Con queste informazioni è possibile reagire alle congestioni della rete e spostare il traffico per evitare i percorsi congestionati. Un'implementazione multipath a questo livello permette di essere trasparente sia per i livelli superiori che per quelli inferiori, il che significa che userà flussi multipli che assomigliano a connessioni TCP. L'implementazione del protocollo multipath al livello di trasporto, può offrire funzioni di gestione del percorso, programmazione dei pacchetti, controllo della congestione senza bisogno di modificare i livelli superiori e inferiori. In questo senso, per supportare un collegamento multipath, è richiesto solo che gli endpoint supportino il protocollo e non è necessario aggiornare alcun router esistente o componente di livello tre tra gli endpoint.

2.1.4 Approccio a livello di applicazione

Un esempio di soluzioni a livello applicativo sono i protocolli Peer-to-Peer (P2P), come BitTorrent: gli approcci multipath hanno l'obiettivo di aumentare esclusivamente il throughput. Essi raggiungono il loro obiettivo scaricando le diverse parti di un file attraverso diversi peer, scegliendo di scaricare dai server più veloci disponibili. BitTorrent realizza il pooling delle risorse a livello di lavoro nei trasferimenti many-to-one. Si comporta come un controllo di congestione multipath disaccoppiato, eseguendo un controllo di congestione indipendente su ogni percorso, con percorsi che hanno diversi punti finali. È possibile sviluppare un'applicazione multipath che può fornire il multihoming per i dispositivi e i server disponibili utilizzando protocolli P2P. Il problema con questo tipo di soluzione è che può portare limitazioni agli altri utenti della rete. Questo approccio fa uso della diversità di percorso, dato che ci sono più server disponibili, ma non affronta il multipath end to end.

2.2 Approcci alternativi

Nel corso degli anni, l'IETF ha creato un gruppo di lavoro che ha studiato l'uso simultaneo di percorsi multipli al livello di trasporto. L'MPTCP non è stato il primo approccio, ma è il più recente e promettente.

2.2.1 Soluzioni multipercorso

Una delle prime soluzioni per un protocollo di trasporto, che permetta flussi multipli attraverso percorsi diversi, è sicuramente Stream Control Transmission Protocol (SCTP), che offre un trasferimento affidabile di messaggi utente tra due endpoint SCTP. Durante l'associazione di avvio, fornisce una lista di indirizzi IP multipli in combinazione con una porta SCTP, in modo che ogni endpoint possa formare un indirizzo di trasporto compatibile. La motivazione di questo protocollo si basava sulle limitazioni del TCP per le applicazioni attuali, al fine di inviare dati in modo affidabile sopra lo User Datagram Protocol (UDP). Un'altra motivazione era quella di supportare nativamente il multihoming a livello di trasporto. Il problema principale di questo protocollo deriva dal fatto che non è facilmente distribuibile su Internet, a causa dei dispositivi non compatibili che non riconoscono il protocollo e marcano il suo traffico come malevolo o sconosciuto.
Un approccio interessante ed importante per l'evoluzione di questa tipologia di protocolli, è stato multiple paths TCP (mTCP). Questo è un protocollo end-to-end, che può unire la larghezza di banda disponibile tra percorsi paralleli ridondanti, molto comuni tra una coppia di host, ottenendo una migliore prestazione e più robustezza ai guasti del percorso. Può fornire un più alto throughput end-to-end e supporta anche un meccanismo condiviso di rilevamento della congestione al fine di evitare percorsi che hanno una congestione condivisa. Un altro approccio è il protocollo pTCP. Questo protocollo è un approccio end-to-end a livello di trasporto per l'aggregazione della larghezza di banda su agenti multihoming (host mobili): può separare il recupero delle perdite dal controllo della congestione ed esegue anche uno stripping intelligente dei dati attraverso le connessioni.
Altri approcci basati su SCTP, come Sender-Based Packet Pair SCTP (SBPP-SCTP) e Westwood SCTP (W-SCTP), hanno anche la capacità di permettere l'uso simultaneo di più flussi. Un altro approccio, quello che mostra più somiglianza con MPTCP, è il Concurrent Multi-Path Stream Control Transmission Protocol (CMP-SCTP). Si tratta di un'estensione del protocollo SCTP, che beneficia della caratteristica del multihoming per aumentare il throughput tra due endpoint multihomed. Tale estensione aggiunge la tolleranza ai guasti, inviando dati simultaneamente su tutti i percorsi. Il mittente può ricevere informazioni di riconoscimento su entrambi i percorsi di ritorno, il che realizza un'efficace distribuzione del carico a livello di trasporto.

2.2.2 Soluzioni multihoming

Un aspetto interessante di MPTCP è che può trarre vantaggio dall'utilizzo simultaneo di diverse interfacce disponibili in un dispositivo, permettendo di migliorare l'affidabilità della connettività nelle apparecchiature mobili. Un approccio per la

mobilità è il Mobility Support in IPv6 (MIPv6). Il MIPv6 è un protocollo che permette ai nodi di essere raggiungibili mentre sono in movimento, purché siano in un Internet IPv6. Questo è possibile perché il nodo mobile ha tre indirizzi IPv6: il primo è l'indirizzo di casa del nodo, il secondo l'indirizzo link-local del nodo e il terzo è il care-of-address del nodo mobile. Questi tre indirizzi permettono all'home-agent e al foreign-agent di creare un tunnel, mentre il nodo è lontano: ciò permette all'home agent di sapere come trovare il nodo mobile indipendentemente dalla sua posizione. Prima del MIPv6, si utilizzava un'implementazione su IPv4, MIPv4. Si tratta della base per il MIPv6, con la differenza che presupponeva che ogni nodo della rete avesse un indirizzo unico, e proprio per questo è diventato obsoleto, data la crescita del numero di personal computer e l'emergere di oggetti quotidiani connessi a Internet, come telefoni cellulari, tablet, orologi, TV, automobili e così via.
Un altro approccio alla mobilità che ha capacità di multihoming è il protocollo Shim6. Questo è un protocollo di livello 3, il che significa che questa soluzione IPv6 permette l'implementazione di multipath a livello di pacchetto invece di routing a livello di connessione. Lo svantaggio principale di questo protocollo è che il bilanciamento del carico è effettuato a livello di connessione, il che significa che il throughput è altamente influenzato dal collegamento più lento.
Anche se la parte di sicurezza del protocollo non è l'obiettivo di questa tesi, è importante tenere a mente che ci sono alcune minacce alla sicurezza per le soluzioni multihoming. Ecco una lista di quelle più comuni:

- L'hacker viene identificato come una fonte/destinazione valida da parte dell'utente, il che può portare all'iniezione di pacchetti nella rete dell'utente.
- Un utente invia dei pacchetti a un destinatario inesistente, l'hacker può eseguire il reindirizzamento di questi pacchetti per creare un attacco Denial-of-Service (DoS) su terze parti.

2.2.3 Approcci alternativi agli scenari multipercorso

I progetti successivamente descritti sono in fase di sviluppo, pertanto non sono ancora in una versione finale e non sono stati approvati dalla IETF.
Una di queste soluzioni è quella sviluppata da Iljitsch van Beijnum. Beijnum ha sviluppato un protocollo TCP multipath che è implementato solo sul lato dell'host mittente, senza richiedere alcuna modifica all'estremità del ricevitore. Questa vista permette al protocollo di utilizzare la capacità disponibile sui diversi percorsi multipli, o dinamicamente trovare il migliore, con l'obiettivo finale di ottenere un throughput più elevato.
Un approccio totalmente diverso è il Transparent MPTCP Proxy (TMPP). L'idea

alla base di questo lavoro è di permettere agli host, che non supportano MPTCP, di utilizzare il supporto ai percorsi multipli.
TMPP funziona come un gateway in un ambiente NAT (Network Address Translation), cambiando i pacchetti ricevuti e trasmettendoli di nuovo.

Capitolo 3

MPTCP

Multipath TCP (MPTCP) è un insieme di estensioni al TCP standard per fornire un servizio TCP Multipath, che consente a una connessione di trasporto di operare su più percorsi contemporaneamente. Questo capitolo presenta le modifiche al protocollo necessarie per aggiungere capacità multipath al protocollo TCP e ne descrive il funzionamento. Tutte le operazioni MPTCP vengono segnalate utilizzando i campi opzionali dell'header TCP. Un singolo numero di opzione TCP ("Kind") è stato assegnato da IANA per MPTCP , e quindi i singoli messaggi saranno determinati da un "sottotipo", i cui valori sono anche memorizzati in un registro IANA. Come con tutte le opzioni TCP, la lunghezza del campo è specificata in byte e include i 2 byte di Kind e Length. I messaggi MPTCP sono definiti nelle sezioni seguenti.

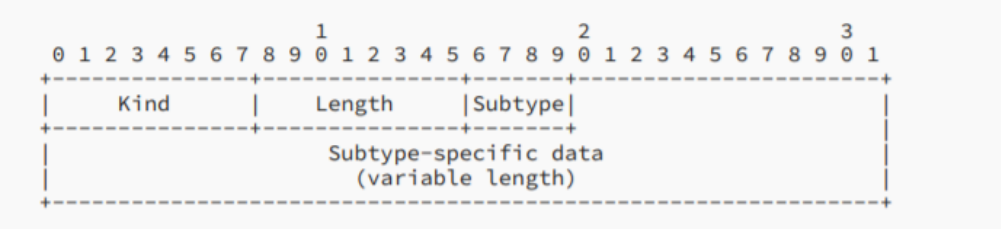


Figura 3.1: Formato opzione MPTCP

Queste opzioni MPTCP associate con l'inizializzazione di un subflow, sono usate nei pacchetti con il flag SYN settato. Inoltre, esiste un'opzione MPTCP per la segnalazione dei metadati, così da garantire che i dati segmentati possano essere ricombinati per la consegna all'applicazione. Mentre un'implementazione potrebbe desiderare di inviare le opzioni MPTCP il prima possibile, potrebbe non essere possibile combinare tutte le opzioni desiderate (sia quelle per MPTCP che per il normale TCP) su un singolo pacchetto. Pertanto, un'implementazione può scegliere di inviare ACK duplicati contenenti le informazioni di segnalazione aggiuntive.

Questo cambia la semantica di un ACK duplicato; questi vengono solitamente inviati solo come segnale di un segmento perso nel normale TCP. Pertanto, un'implementazione MPTCP che riceve un ACK duplicato, che contenga un'opzione MPTCP, non deve essere trattato come un segnale di congestione. Inoltre, un'implementazione MPTCP non dovrebbe inviare più di due ACK duplicati di fila, allo scopo di inviare le sole opzioni MPTCP e garantire che nessun middlebox lo interpreti erroneamente come un segno di congestione. Inoltre, i controlli TCP di validità (come garantire che il numero di sequenza e il numero di riconoscimento siano all'interno della finestra) devono essere effettuati prima di elaborare qualsiasi segnale MPTCP.

3.1 Ipotesi per la realizzazione

Al fine di limitare lo spazio di progettazione si hanno 2 vincoli chiave:

- Deve essere retrocompatibile con l'attuale TCP, per aumentarne le possibilità di implementazione.
- Si può presumere che uno o entrambi gli host siamo multihomed e multindirizzo.

Per semplificare la progettazione, assumiamo che la presenza di più indirizzi su un host sia sufficiente per indicare l'esistenza di più percorsi. Questi percorsi non devono essere del tutto disgiunti: possono condividere uno o più router tra loro. Anche in una situazione del genere, l'utilizzo di più percorsi è vantaggioso, consentendo di migliorare l'utilizzo delle risorse e la resilienza a un sottoinsieme di nodi guasti. L'algoritmo di controllo della congestione definito in RFC6356 [7] garantisce che l'uso di più percorsi non agisca in modo dannoso. Inoltre, ci possono essere alcuni scenari in cui differenti porte TCP su un singolo host possono fornire percorsi disgiunti, e quindi il design MPTCP supporta anche l'uso di porte nell'identificazione del percorso.
Ci sono tre aspetti della retrocompatibilità sopra elencati:

- Vincoli esterni: il protocollo deve funzionare attraverso la stragrande maggioranza degli esistenti middlebox come NAT, firewall e proxy e, in quanto tali, devono assomigliare il più possibile al TCP esistente in rete. Inoltre il protocollo non deve presumere che i segmenti che invia arrivino invariati a destinazione: possono essere divisi o riuniti; Le opzioni TCP possono essere rimosse o duplicate.
- Vincoli dell'applicazione: il protocollo deve essere utilizzabile senza modifiche dalle applicazioni esistenti che utilizzano API TCP(sebbene sia ragionevole

che non tutte le funzionalità siano disponibili per tali applicazioni legacy). Inoltre, il protocollo deve fornire all'applicazione lo stesso modello di servizio del normale TCP.

- Fallback: il protocollo dovrebbe essere in grado di tornare al TCP standard senza interferenze da parte dell'utente, per poter comunicare con host legacy.

3.2 Struttura del livello di trasporto

In questa sezione descriviamo la struttura che MPTCP usa al livello di trasporto. Il MPTCP divide il livello di trasporto in due sottolivelli, come si può vedere nella figura 3.2. Il sottolivello superiore è responsabile della raccolta delle informazioni necessarie per gestire la connessione e opera end-to-end. Il sottolivello inferiore è responsabile dei sottoflussi, per farli vedere come un unico flusso TCP e permette al componente TCP di operare segmento per segmento. Questa struttura è stata progettata in modo da essere trasparente sia per gli strati superiori che per quelli inferiori. Per gestire i sottoflussi multipli TCP sottostanti, l'estensione MPTCP deve implementare la gestione dei percorsi, la programmazione dei pacchetti, l'interfaccia subflow e le funzioni di controllo della congestione.

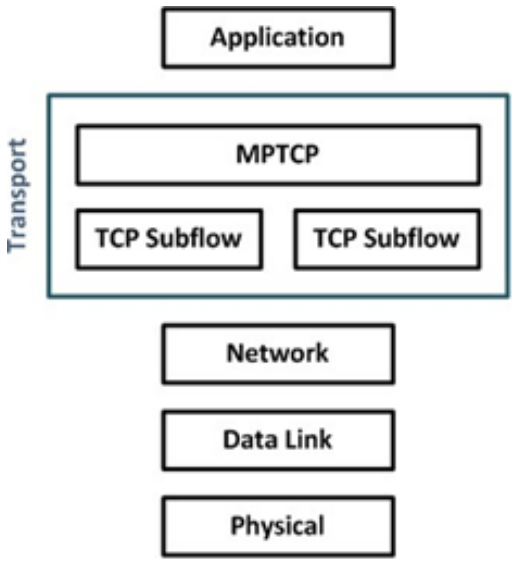


Figura 3.2: Struttura MPTCP a livello di trasporto

La gestione dei percorsi è responsabile del rilevamento e dell'utilizzo dei percorsi

disponibili tra due host. Questa funzione è anche responsabile del meccanismo di segnalazione di indirizzi alternativi agli host e ad impostare nuovi sottoflussi uniti ad una connessione TCP MultiPath esistente. Lo scheduling dei pacchetti è il luogo dove il flusso di byte ricevuto dall'applicazione viene spezzato in segmenti al fine di trasmetterli su uno dei sottoflussi disponibili. Sempre nella programmazione dei pacchetti, il riordino a livello di connessione viene eseguito ogni volta che i pacchetti vengono ricevuti dai sottoflussi TCP. Per permettere il corretto ordine dei segmenti inviati sui diversi sottoflussi, MPTCP utilizza una mappatura della sequenza dei dati, associando i segmenti a una numerazione di sequenza a livello di connessione. Per avere a disposizione le informazioni corrette dei sottoflussi, lo schedulatore di pacchetti dipende dalle informazioni acquisite dal componente di gestione del percorso. La funzione di controllo della congestione è responsabile del coordinamento del controllo della congestione attraverso i sottoflussi. //Questo coordinamento determina la programmazione dei segmenti, determinando quali di essi debbano essere inviati su quali subflow e a quale velocità. L'interfaccia subflow è responsabile di trasmettere sul percorso specificato i segmenti ricevuti dal componente di pianificazione dei pacchetti. Alla ricezione di un segmento, il subflow passa i dati al pacchetto programmazione per il riassemblaggio a livello di connessione. Poiché MPTCP utilizza TCP per la compatibilità di rete, viene assicurata una consegna in ordine e affidabile. Per rilevare e ritrasmettere i pacchetti persi a livello di subflow, il TCP aggiunge ai segmenti la propria sequenza numeri.

Valore	Simbolo	Designazione
0x0	MP_CAPABLE	Multipath Capable
0x1	MP_JOIN	Join Connection
0x2	DSS	Data Sequence Signal
0x3	ADD_ADDR	Add address
0x4	REMOVE_ADDR	Remove Address
0x5	MP_PRIO	Change Subflow Priority
0x6	MP_FAIL	Fallback
0x7	MP_FASTCLOSE	Fast Close
0x8-0xe	Unassigned	Unassigned
0xf	Reserved for Private Use	Reserved for Private Use

Tabella 3.1: Sottotipi di opzioni MPTCP

3.3 Avvio di una connessione

L'avvio della connessione inizia con uno scambio SYN, SYN/ACK, ACK su un unico percorso. Ciascun pacchetto contiene l'opzione MPTCP Multipath Capable

(MP_CAPABLE). Questa opzione ha una lunghezza variabile e serve a più scopi. In primo luogo, verifica se l'host remoto supporta Multipath TCP; in secondo luogo consente agli host di scambiare alcune informazioni per autenticare l'istituzione di ulteriori subflows.

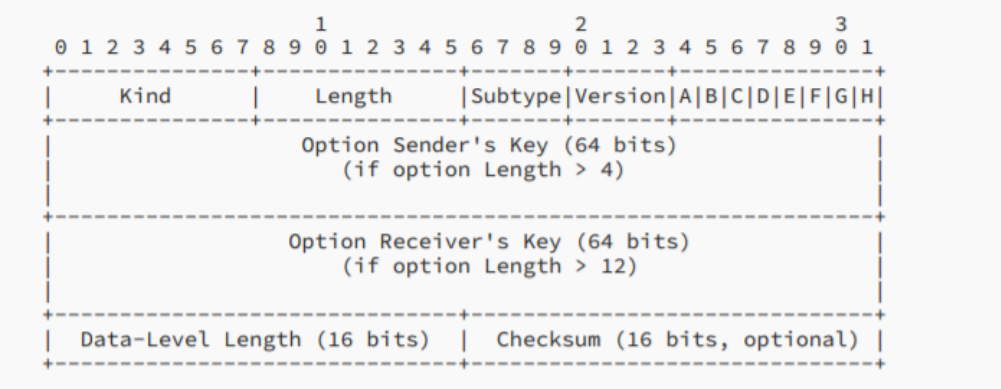


Figura 3.3: Opzione MP_CAPABLE

I dati trasportati da ciascuna opzione sono i seguenti, dove A è l'iniziatore mentre B l'ascoltatore.

- SYN (da A a B): solo il primi 4 ottetti (Lunghezza 4).
- SYN/ACK (da B a A): chiave di B per questa connessione (Lunghezza 12).
- ACK (no data) (da A a B): chiave di A seguita da quella di B (Lunghezza 20).
- ACK (con i primi dati) (da A a B): chiave di A seguita da quella di B seguita da Data-Level Length e Checksum opzionale (Lunghezza 22 o 24).

Il contenuto dell'opzione è determinato dai flags SYN e ACK del pacchetto, insieme all'opzione lunghezza. Il SYN iniziale, contenente solo l'intestazione MP_CAPABLE, viene utilizzato per definire la versione di MPTCP richiesta e scambiare determinati flags per negoziare le funzionalità di connessione. Questa opzione viene utilizzata per dichiarare le chiavi a 64 bit che gli host finali hanno generato per questa connessione MPTCP. Queste chiavi vengono utilizzate per autenticare l'aggiunta di futuri subflows a questa connessione. Questa è l'unica volta che la chiave verrà inviata in chiaro (a meno che "Fast Close" venga utilizzato); tutti i futuri subflows identificheranno la connessione utilizzando un "token" a 32 bit. Questo token è un hash crittografico di questa chiave. L'algoritmo per questo processo dipende dall'algoritmo di autenticazione selezionato; il metodo di

selezione è definito più avanti in questa sezione. Alla ricezione del segmento SYN iniziale, un stateful server genera una chiave casuale e risponde con un SYN/ACK. La chiave deve ovviamente essere univoca per il mittente in tutte le sue connessioni MPTCP. Le connessioni verranno indicizzate in ogni host dal token (un hash unidirezionale della chiave). Pertanto, un'implementazione richiederà una mappatura da ciascun token alla connessione corrispondente e, a sua volta, alle chiavi per la connessione. Inoltre, un'implementazione dovrebbe controllare il suo elenco di token di connessione per assicurarsi che non ci siano collisioni prima di inviare la sua chiave e, nel caso ci sia collisione, dovrebbe generare una nuova chiave. Tuttavia, questo sarebbe costoso per un server con migliaia di connessioni. Il meccanismo degli handshake dei subflows garantirà che il nuovo subflows si unisca solo alla connessione corretta. Quindi, nel peggiore dei casi, se c'è stata una collisione simbolica, il nuovo subflow non sarà creato, ma la connessione MPTCP continuerebbe a fornire un regolare servizio TCP. Poiché la generazione della chiave è specifica per l'implementazione, non è necessario che siano semplicemente numeri casuali. Un'implementazione è libera di scambiare materiale crittografico fuori banda e generare queste chiavi da questo materiale, al fine di fornire meccanismi aggiuntivi mediante i quali verificare l'identità delle entità comunicanti.

Come tutte le opzioni MPTCP, l'opzione MP_CAPABLE inizia con Kind e Length per specificare il tipo e la lunghezza dell'opzione TCP. Queste informazioni sono seguite dall'opzione MP_CAPABLE. I primi 4 bit del primo ottetto nell'opzione MP_CAPABLE definiscono il sottotipo di opzione MPTCP (per MP_CAPABLE, questo valore è 0x0) e i restanti 4 bit di questo ottetto specificano la versione MPTCP in uso. Il secondo ottetto è riservato ai flags, così ripartiti:

- A: viene impostato ad 1 per richiedere il Checksum
- B: è un flag di estendibilità, deve essere lasciato a 0 per le implementazioni correnti. Verrà usato in futuro come parte di un meccanismo di sicurezza alternativo. Se si riceve un messaggio con questo flags settato ad 1 il SYN deve essere ignorato, tornando ad una normale connessione TCP.
- C: Il terzo bit, etichettato "C", è impostato su 1 per indicare che il mittente di questa opzione non accetterà ulteriori subflows MPTCP dall'indirizzo e porta sorgente.
- da D a H: questi ultimi vengono utilizzati per la negoziazione dell'algoritmo crittografico.

La ritrasmissione di ACK + MP_CAPABLE può avvenire se non è possibile sapere se è stata ricevuta. I seguenti diagrammi mostrano tutti i possibili scambi per il setup iniziale dei subflows.

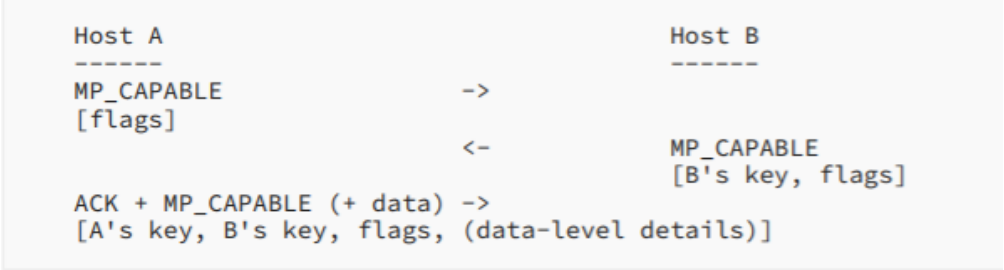


Figura 3.4: Avvio comunicazione

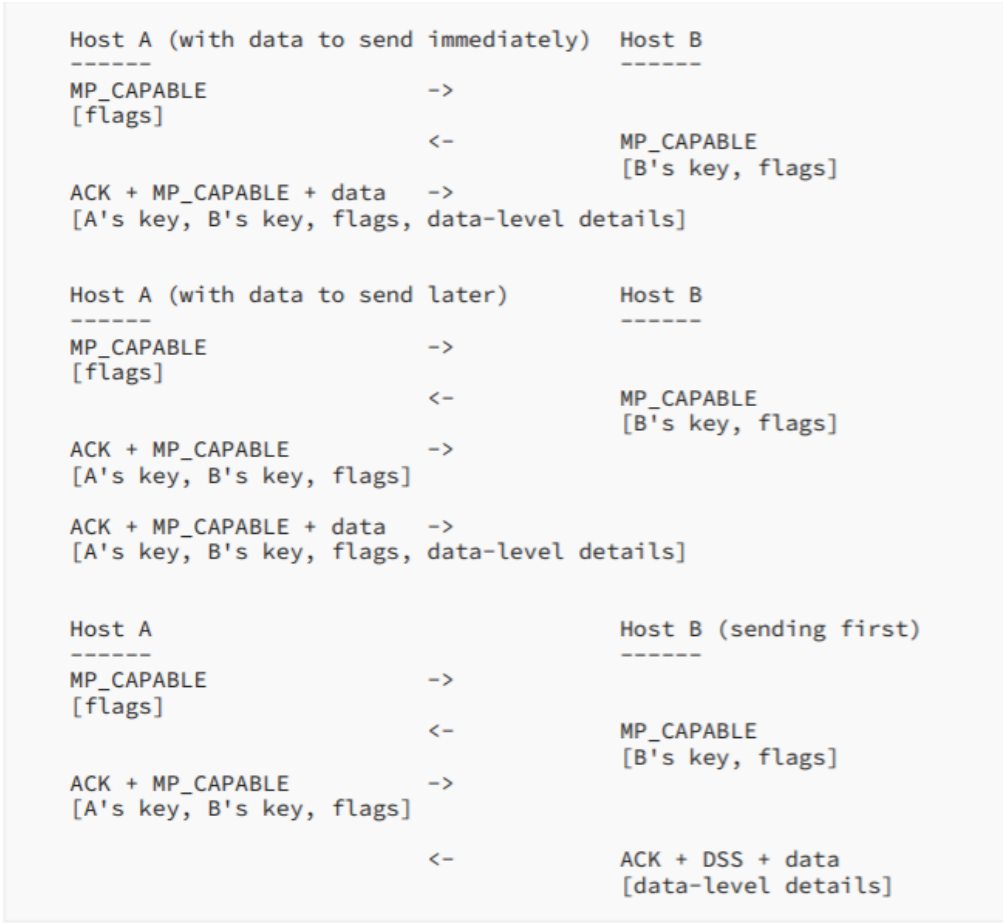


Figura 3.5: Possibili scambi per setup iniziale

3.4 Associazione di un nuovo subflow ad una connessione MPTCP preesistente

Una volta iniziata una connessione MPTCP con lo scambio MP_CAPABLE, ulteriori subflows possono essere aggiunti alla connessione. Lo scambio di chiavi nel MP_CAPABLE Handshake fornisce tutti i dati necessari che verranno usati dagli endpoints quando si vorrà configurare un nuovo subflows. I subflows vengono aggiunti come se si instaurasse una normale connessione TCP, ma i pacchetti SYN, SYN/ACK, e ACK contengono l'opzione MP_JOIN. È consentito a entrambi gli host di una connessione avviare la creazione di un nuovo subflow, ma ci si aspetta che questo venga fatto dall'iniziatore della connessione originale. Un nuovo subflow viene avviato come un normale scambio TCP SYN/ACK. L'opzione Join Connection(MP_JOIN) MPTCP viene utilizzata per identificare la connessione a cui deve unirsi il nuovo subflow. Utilizza il materiale per le chiavi che è stato scambiato nell'handshake MP_CAPABLE iniziale, e quell'handshake negozia anche l'algoritmo crittografico in uso per l'handshake MP_JOIN. L'host A avvia un nuovo subflows tra uno dei suoi indirizzi e uno degli indirizzi dell'Host B. Il token, generato dalla chiave, viene utilizzato per identificare a quale connessione MPTCP si sta collegando e l'Hash-based Message Authentication Code (HMAC) viene utilizzato per l'autenticazione. L'HMAC utilizza le chiavi scambiate nel MP_CAPABLE handshake e i numeri casuali scambiati in queste opzioni MP_JOIN. Questa sezione specifica il comportamento di MP_JOIN utilizzando l'algoritmo HMAC-SHA256.

Figura 3.6: Associazione nuovo subflow

Un'opzione MP_JOIN è presente in SYN, SYN/ACK e ACK, sebbene in ogni caso con un formato differente. Nel primo MP_JOIN sul pacchetto SYN, illustrato in Figura 5, l'iniziatore invia un token, un numero casuale e un Address ID.

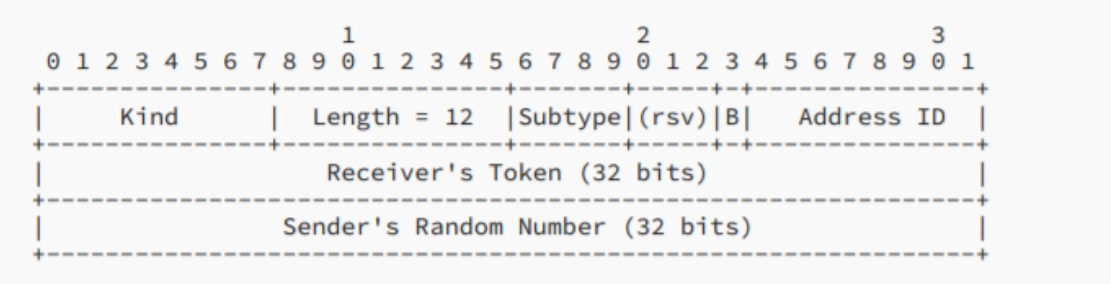


Figura 3.7: Opzione MP_JOIN

Il token viene utilizzato per identificare la connessione MPTCP ed è un hash crittografico della chiave del destinatario, come scambiato nell'handshake MP_CAPABLE iniziale. In questa specificazione, i token presentati in questa opzione sono generati dall'algoritmo SHA-256, troncato ai più significativi 32 bit. Il token incluso nell'opzione MP_JOIN è il token che il destinatario del pacchetto utilizza per identificare questa connessione; cioè, Host A invierà Token-B (che è generato da Key-B). Si noti che l'algoritmo di generazione dell'hash può essere sovrascritto dalla scelta dell'algoritmo di handshake crittografico. MP_JOIN SYN invia non solo il token (che è statico per una connessione) ma anche numeri casuali che vengono utilizzati per impedire attacchi di "replay" al metodo di autenticazione. L'opzione MP_JOIN include un Address ID. Questa è un'identità generata dal mittente dell'opzione, utilizzato per identificare l'indirizzo di origine di questo pacchetto. Il valore numerico di questo campo è generato dal mittente e deve essere mappato in modo univoco ad un indirizzo IP per l'host di invio. L'Address ID consente la rimozione dell'indirizzo tramite NAT, consente inoltre la correlazione tra nuovi subflow e previene la creazione di duplicati nello stesso percorso. Gli Address ID dei subflow utilizzati nello scambio SYN iniziale nella connessione sono impliciti e hanno valore zero. Un host deve memorizzare le mappature tra Address ID e indirizzi IP sia per se stesso che per l'host remoto. Un'implementazione dovrà anche sapere quali Address ID locali e remoti sono associati a quale subflow, per quando gli indirizzi verranno rimossi da un host locale o remoto. L'opzione MP_JOIN sui pacchetti con SYN flag include anche 4 bit, di cui 3 attualmente riservati e impostati a 0 dal mittente. Il bit finale, etichettato "B", indica se il mittente di questa opzione desidera utilizzare questo subflow come percorso di backup (B=1) in caso di guasto di altri percorsi o vuole utilizzarlo immediatamente come parte della connessione. Impostando B=1, il mittente dell'opzione richiede che l'altro host invii dati solo su questo subflow se non ci sono subflow disponibili dove B=0. Quando si riceve un SYN con un'opzione MP_JOIN che contiene un token valido per una connessione MPTCP esistente, il destinatario dovrebbe rispondere con un SYN/ACK con un'opzione MP_JOIN contenente un numero casuale e un HMAC troncato (64 bit più a significativi). Questa versione dell'opzione è mostrata in

Figura 3.8. Se il token è sconosciuto o l'host vuole rifiutare il subflow, il ricevitore invierà un segnale di reset (RST), analogo a una porta sconosciuta in TCP, contenente un'opzione MP_TCPRST con un "MPTCP specific error" come codice di errore.

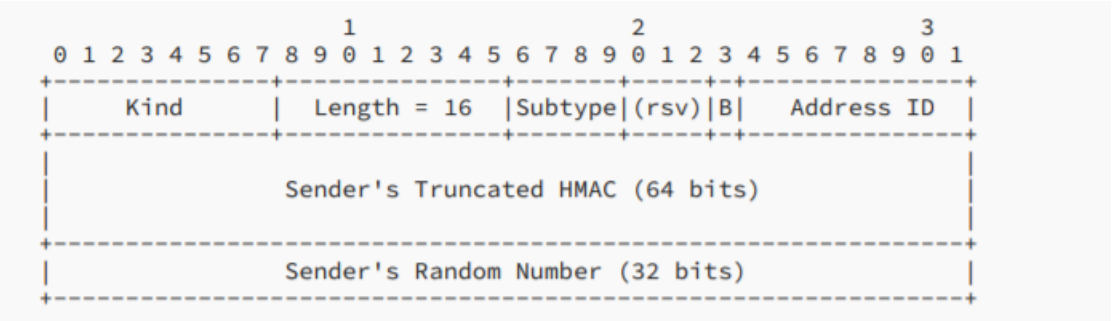


Figura 3.8: Opzione MP_JOIN in risposta a SYN/ACK

Un HMAC viene inviato da entrambi gli hosts, dall'iniziatore (Host A) nel terzo pacchetto (l'ACK) e dal risponditore (Host B) nel secondo pacchetto (il SYN/ACK). Fare lo scambio HMAC in questa fase, consente a entrambi gli host di avere i primi dati casuali scambiati (nei primi due pacchetti SYN) utilizzandoli come "messaggio". HMAC viene utilizzato insieme all'algoritmo hash SHA-256 con output troncato ai 160 bit più significativi. Le informazioni di autenticazione dell'iniziatore vengono inviate nel suo primo ACK (il terzo pacchetto), come mostrato in Figura 7. Questi dati devono essere inviati in modo affidabile, poiché è l'unica volta che viene inviato questo HMAC; quindi, ricevuto questo pacchetto B dovrà inviare un TCP standard ACK in risposta.

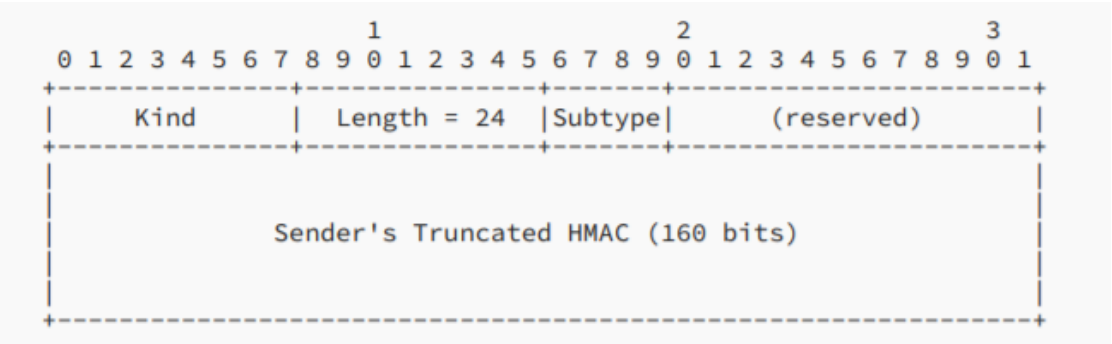


Figura 3.9: Opzione MP_JOIN per il primo ACK

La chiave dell'algoritmo HMAC, nel caso del messaggio trasmesso dall'Host A, sarà Key-A seguita da Key-B; e nel caso di Host B, Key-B seguito da Key-A. Queste sono le chiavi che sono state scambiate nell'handshake MP-CAPABLE

originale. Il "messaggio" per l'algoritmo HMAC in ogni caso è la concatenazione di numeri casuali per ciascun host (indicato con R): per Host A, R-A seguito da R-B; e per Host B, R-B seguito da R-A. Segue Figura 3.10 dove le varie opzioni elencate fino ad ora abilitano il setup di un subflow.

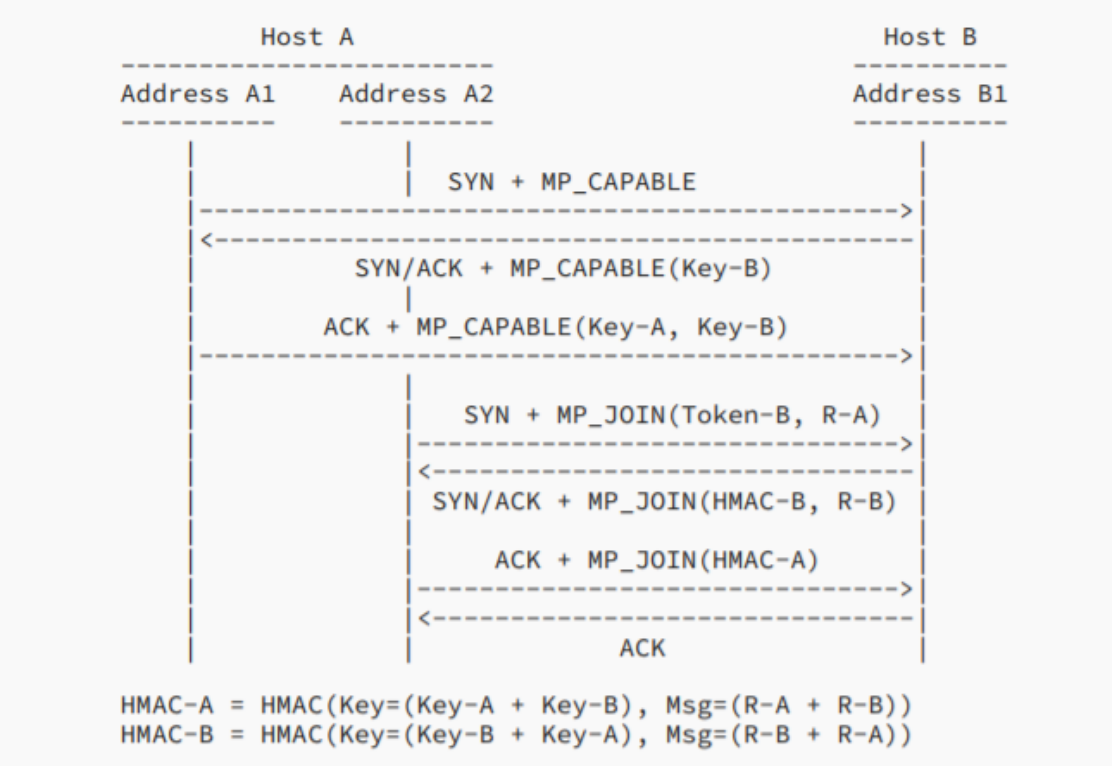


Figura 3.10: Autenticazione MPTCP

3.5 Informare l'altro Host per un nuovo potenziale indirizzo

L'insieme di indirizzi IP associati a un host multihomed può cambiare durante la connessione. MPTCP supporta l'aggiunta e la rimozione di indirizzi su un host sia in modo implicito che esplicito. Se l'host A ha stabilito un subflow che inizia dalla coppia indirizzo/porta IP-A1 e vuole aprire un secondo subflow, a partire dalla coppia indirizzo/porta IP-A2, avvia semplicemente la creazione di esso come spiegato sopra. L'host remoto verrà quindi informato implicitamente del nuovo indirizzo. In alcune circostanze, un host potrebbe voler annunciare all'host remoto la disponibilità di un indirizzo senza stabilire un nuovo subflow. Nell'esempio

seguente, l'Host A informa l'Host B della sua coppia indirizzo IP/porta alternativa (IP-A2). L'host B può successivamente inviare un MP_JOIN a questo nuovo indirizzo. L'opzione ADD_ADDR contiene un HMAC per autenticare l'indirizzo come inviato dal mittente della connessione. Il destinatario di questa opzione ne fa eco al client per indicare che la ricezione è andata a buon fine.

Figura 3.11: Host A informa B di un nuovo potenziale indirizzo

Esiste un segnale corrispondente per la rimozione dell'indirizzo(REMOVE_ADDR), che utilizza l'Address ID segnalato nell'handshake ADD_ADDR.

```
Host A                                  Host B
------                                  ------
REMOVE_ADDR                  ->
[IP#-A2's Address ID]
```

Figura 3.12: Host A informa B di rimuovere un indirizzo

3.6 Trasferimento dati MPTCP

Questa sezione discute il funzionamento di MPTCP per il trasferimento dei dati. Ad un livello elevato, un'implementazione MPTCP prenderà un flusso di dati di input da un'applicazione e lo dividerà in uno o più subflows, con sufficienti informazioni di controllo per consentirne il riassemblaggio e la consegna in modo affidabile e in ordine all'applicazione ricevente. Le seguenti sottosezioni definiscono questo comportamento in dettaglio. Per garantire una consegna affidabile e ordinata dei dati su subflows che possono apparire e scomparire in qualsiasi momento, MPTCP utilizza un Data Sequence Number (DSN) a 64 bit per numerare tutti i dati inviati tramite la connessione MPTCP. Ogni subflow ha un Sequence Number a 32 bit composto da il numero di sequenza dell'header TCP, e un'opzione MPTCP, successivamente viene mappato ed esteso a 64 bit. In questo modo i dati possono essere ritrasmessi su differenti subflows (mappati con stesso DSN) in caso di guasto. Il Data Sequence Signal (DSS) trasporta il Data Sequence Mapping costituito da: subflow Sequence Number, DSN e la lunghezza per cui questa mappatura è valida. Questa opzione può anche contenere un riconoscimento a livello di connessione (il "Data ACK") per il DSN ricevuto.

```
Host A                                  Host B
------                                  ------
DSS                          ->
[Data Sequence Mapping]
[Data ACK]
[Checksum]
```

Figura 3.13: Invio DSS da A a B

Il Data Sequence Mapping e il Data ACK sono segnalati nell'opzione DSS (Figura 3.14). Uno o entrambi possono essere segnalati in un DSS, a seconda dei flags settati.

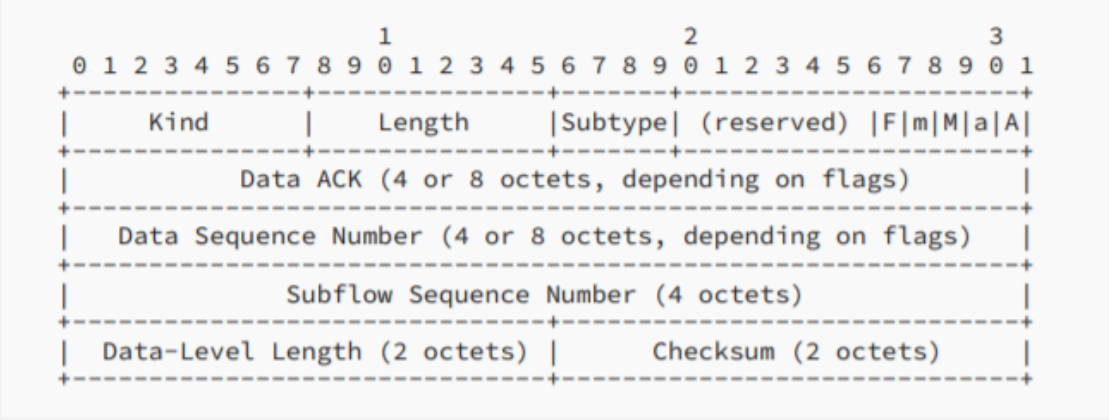


Figura 3.14: Opzioni DSS

I flags, quando impostati, definiscono il contenuto di questa opzione, come segue:

- A = ACK dati presente
- a = Data ACK è 8 ottetti (se non impostato, Data ACK è 4 ottetti)
- M = Numero di sequenza dati (DSN), Subflow Sequence Number (SSN), Data-Level Length e Checksum (se negoziato) presenti
- m = Numero sequenza dati è 8 ottetti (se non impostato, DSN è 4 ottetti)

I flags "a" e "m" hanno significato solo se i corrispondenti flag "A" o "M" sono impostati; in caso contrario, verranno ignorati. La lunghezza massima di questa opzione, è 28 ottetti. Il flag "F" indica "Data FIN". Se presente, significa che questa mappatura copre i dati finali dal mittente. Questo è l'equivalente a livello di connessione del FIN flag su TCP single-path. Una connessione non viene chiusa a meno che non vi sia stato un Data FIN exchange, un messaggio MP_FASTCLOSE o un timeout di invio a livello di connessione. I restanti bit riservati devono essere impostati a 0. Notare che il checksum è presente solo in questa opzione se abilitato nell'handshake MP_CAPABLE. La presenza del checksum è desumibile dalla lunghezza dell'opzione. Se è presente un checksum ma il suo utilizzo non è stato negoziato nell'handshake MP_CAPABLE, il ricevitore deve chiudere il subflow con un RST, in quanto non si sta comportando come negoziato in precedenza. Similmente accade se non è presente il checksum ed in precedenza si era stabilito

il suo utilizzo. In entrambi i casi, questo RST dovrebbe essere accompagnato da un'opzione MP_TCPRST con il codice "MPTCP-specific error".

3.6.1 Data Sequence Mapping

Il flusso di dati nel suo insieme può essere riassemblato attraverso l'uso dei componenti Data Sequence Mapping dell'opzione DSS, che definisce la mappatura dal "subflow sequence number" al "data sequence number". Questo viene utilizzato dal destinatario per garantire la consegna in ordine al livello dell'applicazione. La Data Sequence Mapping è stata scelta per favorire la compatibilità con situazioni in cui la segmentazione TCP/IP viene eseguita separatamente dalla fonte che sta generando i dati, consentendo inoltre un'unica mappatura per coprire molti pacchetti. Una mappatura è fissa, in quanto il "subflow sequence number" è associato al "data sequence number" dopo che la mappatura è stata elaborata, ma è possibile mappare lo stesso "data sequence number" su un diverso subflow per scopi di ritrasmissione. Ciò consente anche l'invio simultaneo degli stessi dati su più subfows per aumentare la resilienza o l'efficienza, in particolare nel caso di collegamenti non stabili. L'algoritmo di checksum utilizzato è quello TCP standard descritto in RFC0793 [6], operando sui dati coperti da questa mappatura, insieme a uno pseudo-header come mostrato in Figura 3.15.

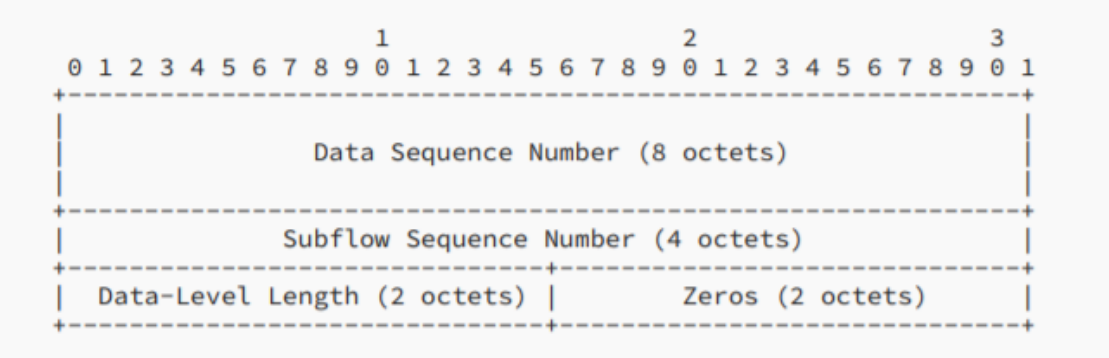


Figura 3.15: Pseudo-Header per il checksum DSS

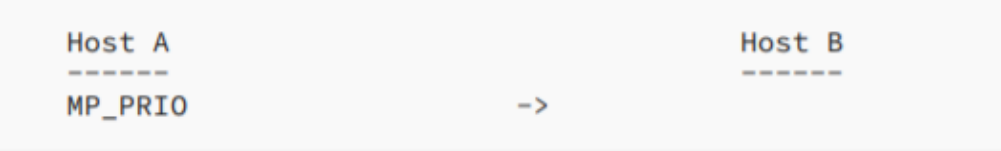
3.6.2 Riconoscimento Dati

Per fornire una resilienza end-to-end completa, MPTCP fornisce un riconoscimento a livello di connessione, che funge da ACK cumulativo per la connessione nel suo insieme. Questo viene fatto tramite il campo "Data ACK" nell'opzione DSS. Il Data ACK è analogo al comportamento dell'ACK cumulativo del TCP standard , che indica quanti dati sono stati ricevuti con successo. Data ACK specifica

il successivo numero di sequenza di dati che ci si aspetta di ricevere. Il Data ACK, come per il DSN, può essere inviato come valore completo a 64 bit o come valore inferiore a 32 bit. Se i dati vengono ricevuti con un DSN a 64 bit, deve essere riconosciuto con un Data ACK a 64 bit. Se il DSN ricevuto è di 32 bit, un'implementazione può scegliere se inviare un Data ACK a 32 o 64 bit. Il Data ACK dimostra che i dati richiesti sono stati ricevuti e accettati dall'estremità remota. La finestra di ricezione è condivisa da tutti i subflow ed è relativa al Data ACK. I Data ACK dovrebbe essere inclusi in tutti i segmenti; tuttavia, nelle implementazioni più avanzate per questioni di ottimizzazione il Data ACK è presente nei segmenti solo quando il valore Data ACK viene incrementato. Questo comportamento assicura che il buffer di invio venga liberato, riducendo l'overhead quando il trasferimento dei dati è unidirezionale.

3.7 Richiesta di modifica della priorità di un percorso

Gli host possono indicare al setup iniziale di un subflow se desiderano utilizzarlo come percorso standard o di backup: un percorso di backup viene utilizzato solo se non sono disponibili percorsi standard. Durante una connessione, l'Host A può richiedere una modifica della priorità di un subflow attraverso il segnale MP_PRIO a Host B.



3.8 Chiusura di una connessione MPTCP

Nel normale TCP, un FIN annuncia al destinatario che il mittente non ha più dati da inviare. Per consentire ai subflows di operare in modo indipendente e mantenere l'aspetto di un normale TCP, un FIN in MPTCP ha effetto solo sul subflow nel quale viene inviato. Ciò consente ai nodi di esercitare una notevole libertà su quali percorsi sono in uso in qualsiasi momento. La semantica di un FIN rimane come per il normale TCP; fino a quando entrambe le parti non hanno confermato(ACK) reciprocamente le FIN il subflow continua ad esistere.

Quando un'applicazione chiama close() su un socket, questo indica che non ha più dati da inviare; per il normale TCP, ciò risulterebbe in un FIN sulla connessione. Per MPTCP è necessario un meccanismo equivalente; questo è indicato come DATA_FIN.
Un DATA_FIN è l'indicazione che il mittente non ha più dati da inviare e come tale può essere utilizzato per verificare che tutti i dati siano stati ricevuti con successo. Un DATA_FIN, come il FIN su una normale connessione TCP, è un segnale unidirezionale. Il DATA_FIN viene segnalato impostando il flag "F" nell'opzione DSS a 1. Un DATA_FIN occupa 1 ottetto dello spazio di sequenza a livello di connessione. Si noti che quando DATA_FIN non è collegato a un segmento TCP contenente dati, il DSS deve avere un subflow con numero di sequenza di 0, una lunghezza del livello di dati di 1 e il DSN che corrisponde al DATA_FIN stesso.

Un DATA_FIN ha la stessa semantica e comportamento di un TCP standard FIN, ma a livello di connessione. In particolare, un host non deve chiudere tutti i subflows funzionanti fino a quando tutti i dati in sospeso non sono stati confermati(ACK) o fino a quando il segmento con DATA_FIN settato è l'unico segmento rimasto. Una volta che un DATA_FIN è stato riconosciuto, tutti i rimanenti subflow devono essere chiusi con scambi FIN standard.

Figura 3.17: Chiusura di un subflows

Esiste un ulteriore metodo di chiusura della connessione, denominato "Fast Close", che è analogo alla chiusura di una connessione TCP a percorso singolo con un segnale RST. Il segnale MP_FASTCLOSE viene utilizzato per indicare al peer che la connessione verrà bruscamente chiusa e nessun dato verrà più accettato. Questo può essere utilizzato su un ACK (che garantisce l'affidabilità del segnale) o un RST (che non garantisce affidabilità). Entrambi gli esempi sono mostrati nella figura seguente.

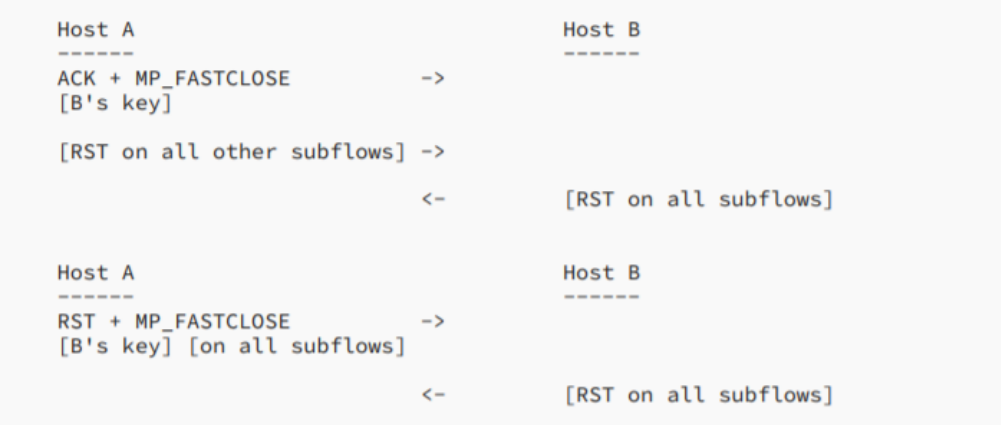


Figura 3.18: Chiusura "Fast Close"

3.9 Politiche nei Subflow

All'interno di un'implementazione MPTCP locale, un host può utilizzare qualsiasi politica locale desidera, per decidere come condividere il traffico da inviare sui percorsi disponibili. Nel caso d'uso tipico, in cui l'obiettivo è massimizzare il throughput, tutti i percorsi disponibili verranno utilizzati contemporaneamente per il trasferimento dei dati. Si prevede, tuttavia, che compariranno altri casi d'uso. Ad esempio, una possibilità è un approccio "tutto o niente", cioè avere un secondo percorso pronto per l'uso in caso di guasto del primo percorso, ma le alternative potrebbero includere la saturazione completa di un percorso prima di utilizzare un percorso aggiuntivo. Tali scelte sarebbero molto probabilmente basate sul costo monetario dei collegamenti, ma potrebbero anche essere basate su proprietà come il ritardo o il jitter dei collegamenti, in cui la stabilità (di ritardo o larghezza di banda) è più importante del throughput. Requisiti dell'applicazione come questi sono discussi in dettaglio in RFC6897 [8]. La capacità di fare scelte efficaci presso il mittente richiede la piena conoscenza del "costo" del percorso, il che difficilmente si verificherà. Sarebbe auspicabile che un ricevitore fosse in grado di segnalare le proprie preferenze per i percorsi, poiché spesso sarà la parte multihomed a dover pagare per la larghezza di banda in entrata misurata. Per abilitare questo comportamento, l'opzione MP_JOIN contiene il bit "B", che permette ad un host di indicare al proprio peer che questo percorso deve essere trattato come un percorso di backup da utilizzare solo in caso di guasto di altri subflows funzionanti (cioè, un subflow dove il ricevitore ha indicato che B=1 NON DOVREBBE essere utilizzato per inviare dati a meno che non ci siano subflows utilizzabili dove B=0). Nel caso in cui l'insieme di percorsi disponibili cambi, un host potrebbe voler segnalare un

cambiamento nella priorità del subflows al peer. Pertanto, l'opzione MP_PRIO, mostrata in figura, può essere utilizzata per modificare il flag "B" del subflow come viene inviato

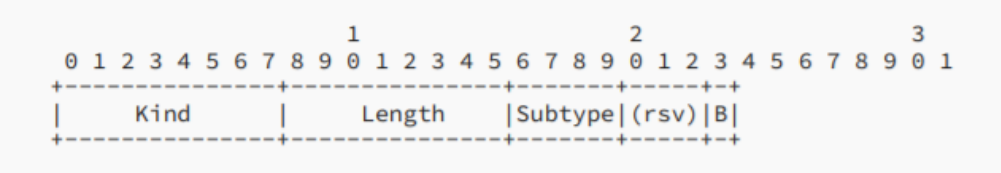


Figura 3.19: Cambio priorità subflow (MP_PRIO)

Un altro uso dell'opzione MP_PRIO è impostare il flag "B"(back-up flag) su un subflow per ritirare il suo uso prima di chiuderlo e rimuoverlo con REMOVE_ADDR - ad esempio, per supportare la continuità della sessione make-before-break, dove nuovi subflows vengono aggiunti prima che i subflows usati vengano chiusi. Va notato che il flag "B" è una richiesta esclusivamente inviata dal destinatario dei dati a un mittente dei dati. Si noti anche che il mittente di questa opzione potrebbe scegliere di continuare a utilizzare il subflow anche se ha segnalato B=1 all'altro host.

3.10 Gestione dei Percorsi

Con la gestione dei percorsi facciamo riferimento allo scambio di informazioni necessario tra host per creare percorsi aggiuntivi. I metodi per condividere queste informazioni sono due, il primo è la configurazione diretta del nuovo subflow dove l'iniziatore ha un indirizzo aggiuntivo. Il secondo (descritto nelle sottosezioni seguenti) segnala gli indirizzi esplicitamente all'altro host, per consentirgli di avviare nuovi subflows propri. Insieme questi meccanismi danno la possibilità agli indirizzi di cambiare al "volo"(e quindi supportare un funzionamento tramite NAT, poiché non è necessario conoscere l'indirizzo di origine).
Esempio di funzionamento:

- Inizialmente viene stabilita una connessione MPTCP tra l'indirizzo/porta A1 dell'Host A e l'indirizzo/porta B1 dell'Host B. Se l'Host A è multihomed e multiindirizzo, può avviare un subflow uscendo dal suo indirizzo A2 a B1, inviando un SYN con un'opzione MP_JOIN da A2 a B1, utilizzando il token precedentemente dichiarato di B per questa connessione. In alternativa, se B è multihomed, può provare a impostare un nuovo subflow da B2 ad A1, utilizzando il token precedentemente dichiarato di A. In entrambi i casi, il SYN verrà inviato alla porta già in uso per il subflow originale sull'host ricevente.

- Contemporaneamente (o dopo un timeout), un'opzione ADD_ADDR viene inviata su un subflow esistente, comunicando al destinatario l'indirizzo o gli indirizzi alternativi del mittente. Il destinatario può utilizzare queste informazioni per aprire un nuovo subflow causa dell'indirizzo o degli indirizzi aggiuntivi del mittente. Nel nostro esempio, A invierà l'opzione ADD_ADDR informando B dell'indirizzo/porta A2. Il mix dell'utilizzo di SYN basata sull'opzione e l'opzione ADD_ADDR, inclusi i timeout, è specifico per l'implementazione e può essere adattato per concordare con la politica locale.

- Ora che A2-B1 è stato impostato correttamente, l'Host B può utilizzare l'Address ID nell'opzione MP_JOIN per correlare questo indirizzo di origine con l'opzione ADD_ADDR che arriverà su un subflow esistente; ora B sa di non aprire A2-B1, ignorando ADD_ADDR. Altrimenti, se B non ha ricevuto A2-B1 MP_JOIN SYN ma ha ricevuto ADD_ADDR, può provare ad avviare un nuovo subflow da uno o più dei suoi indirizzi all'indirizzo A2. Ciò consente di aprire nuove sessioni se un host è dietro un NAT.

3.11 Algoritmi di controllo della congestione

Con la possibilità di utilizzare percorsi multipli, sono sorte preoccupazioni sul controllo della congestione dei dati inviati su questi percorsi.
Una grande mole di lavoro e risorse è stata adibita alla ricerca sugli algoritmi di controllo della congestione e negli schemi di routing che dividono in modo ottimale i pacchetti tra i percorsi disponibili, al fine di migliorare le prestazioni ed essere più resistente ai problemi su percorsi particolari.
Mettiamo a confronto tre algoritmi di controllo della congestione, Fully Coupled, Linked Increases Algorithm, Uncoupled TCP, Dynamic Window Coupling e Opportunistic Linked Increases Algorithm:

- L'algoritmo Fully Coupled considera il caso in cui tutti i flussi hanno RTT(Round trip time) simili: la preoccupazione maggiore su questo algoritmo è che causa ciò che viene chiamato "flappiness". Si dice che il flappiness si verifica quando il traffico di una connessione multi-path tende a concentrarsi su un percorso e poi su un altro.

- L'algoritmo Uncoupled TCP non diffonde la congestione, ma ha difficoltà a spostare il traffico dai link congestionati.

- L'algoritmo Linked Increases(LIA) è proposto per ridurre la flappiness e combinare il pooling delle risorse: è una combinazione della regola di aumento

del Fully Coupled e della regola di diminuzione del Uncoupled TCP. Per compensare le diverse connessioni Round-Trip Time (RTT), è stato studiato un RTT ComPensator Algorithm, che sacrifica un po' di bilanciamento della congestione per una migliore stabilità. Per implementare questo algoritmo è necessario scegliere tra flappiness indesiderabile e sottoflussi disaccoppiati senza risorse di pooling della rete, ma questo algoritmo è ancora una variante di un migliorato ed equo algoritmo di controllo della congestione.

- Dynamic Window Coupling (DWC) è un algoritmo che tende a differenziare se i flussi condividono o meno un collo di bottiglia comune, e accoppia solo i flussi MPTCP che condividono un collo di bottiglia comune. Si è visto che con i possibili scenari di implementazione, gli obiettivi di progettazione delle soluzioni precedenti possono variare le loro prestazioni da buone a scarse. Il DWC può individuare accuratamente un collo di bottiglia e migliorare il throughput ma con lo svantaggio di essere meno amichevole per gli utenti con connessioni TCP singole.

- L'Opportunistic Linked Increases Algorithm (OLIA) è stato proposto al fine di risolvere problemi di LIA e rispettare gli obiettivi di progettazione del MPTCP. Inoltre è anche un algoritmo di controllo della congestione basato su finestre capace di compensare i diversi RTT. Si avvicina ad una soluzione ottimale.

Capitolo 4

Possibili Scenari di base

Questo capitolo elenca diverse configurazioni di base che è possibile adottare a seconda del tipo di scenario.

4.1 Scenari convenzionali wireless e cablati

In questo contesto wireless e wired, vedremo quattro scenari di ambiente statico. Si presuppone che i dispositivi dispongano di un kernel con MPTCP abilitato.

4.1.1 Soluzioni multihoming

Il primo scenario potrebbe consistere in una soluzione multihoming di MPTCP, utilizzando due interfacce di rete, WLAN ed Ethernet, come mostrato in fig. 4.1. I computer in uso hanno due interfacce di rete e sono configurati per utilizzare due diversi collegamenti di rete. Il secondo scenario è simile al primo, ma in questo caso vengono utilizzate due interfacce di rete, entrambe Ethernet, come mostrato in fig. 4.2.

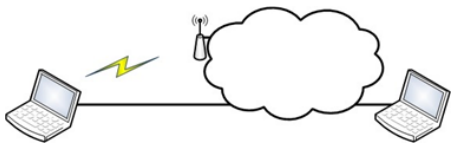


Figura 4.1: Multihoming MPTCP con connessione WiFi ed Ethernet

4.1.2 TCP e MPTCP Scenari concorrenti

Infine, ma non meno importante, il quarto scenario è una situazione mista, come è mostrato nelle figure 4.3. Questo scenario consiste nell'utilizzare due computer

Figura 4.2: Multihoming MPTCP con due connessioni Ethernet

collegati a 2 reti con l'implementazione MPTCP abilitata, e altri due che utilizzano il normale TCP collegati ad una rete. Sicuramente questo scenario metterà in luce il comportamento egoistico di MPTCP quando ci sono flussi TCP standard sulla stessa rete.

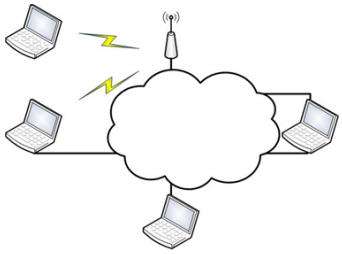


Figura 4.3: Scenari concorrenti TCP e MPTCP con connessioni Ethernet e WiFi

4.2 Soluzioni mobile

In questa soluzione abbiamo invece un'implementazione mobile per MPTCP, al fine di valutare la capacità di mobilità offerta dal protocollo e il suo handover tra diversi protocolli di rete. Si tratta di un approccio interessante che utilizza uno smartphone con connessioni 3G/4G e WLAN.

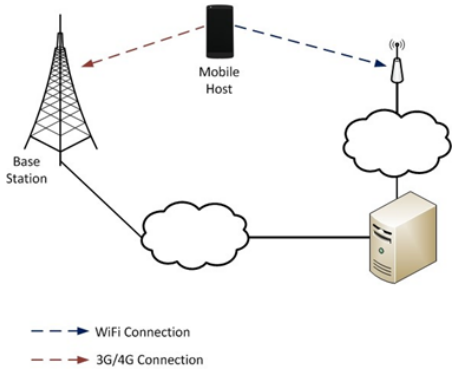


Figura 4.4: Scenario del dispositivo mobile

Capitolo 5

Testing MPTCP su Ambiente Virtuale

In questo capitolo si documenta l'esperienza svolta in ambiente virtuale per capire come abilitare MPTCP e relativi test in merito alla perfomance di esso.

5.1 Installazione e Configurazione su Vmware

Come primo passo si è dovuto scegliere una distro Linux dove fosse possibile installare e abilitare MPTCP. La scelta di Ubuntu (in particolare ver.20.04.3 LTS - Local Fossa) è sicuramente stata dettata dalla sua facilità d'uso, ma soprattutto dalla possibilità di avere una documentazione per quanto riguarda l'installazione di MPTCP. Dopo aver effettuato il deploy su due VM, si è passati alla configurazione di esse.
L'hardware sottoelencato è stato messo a disposizione per ciascuna macchina virtuale:

- 4 Core(Intel i5-10210U)
- 3 GB di RAM
- 30 GB SSD
- 2 schede di rete virtuali

5.2 Installazione MPTCP

Per l'installazione di MPTCP è necessario scaricare i sorgenti qui presenti: (github.com/multipath-tcp/mptcp). Una volta scaricati ed estratti i file è necessario installare GCC e i development tools attraverso il comando "apt-get install build-essential libncurses-dev bison flex libssl-dev libelf-dev". Successivamente all'installazione, sarà necessario configurare il Kernel attraverso il comando "make menuconfig" che aprirà una GUI con tutto l'occorrente.

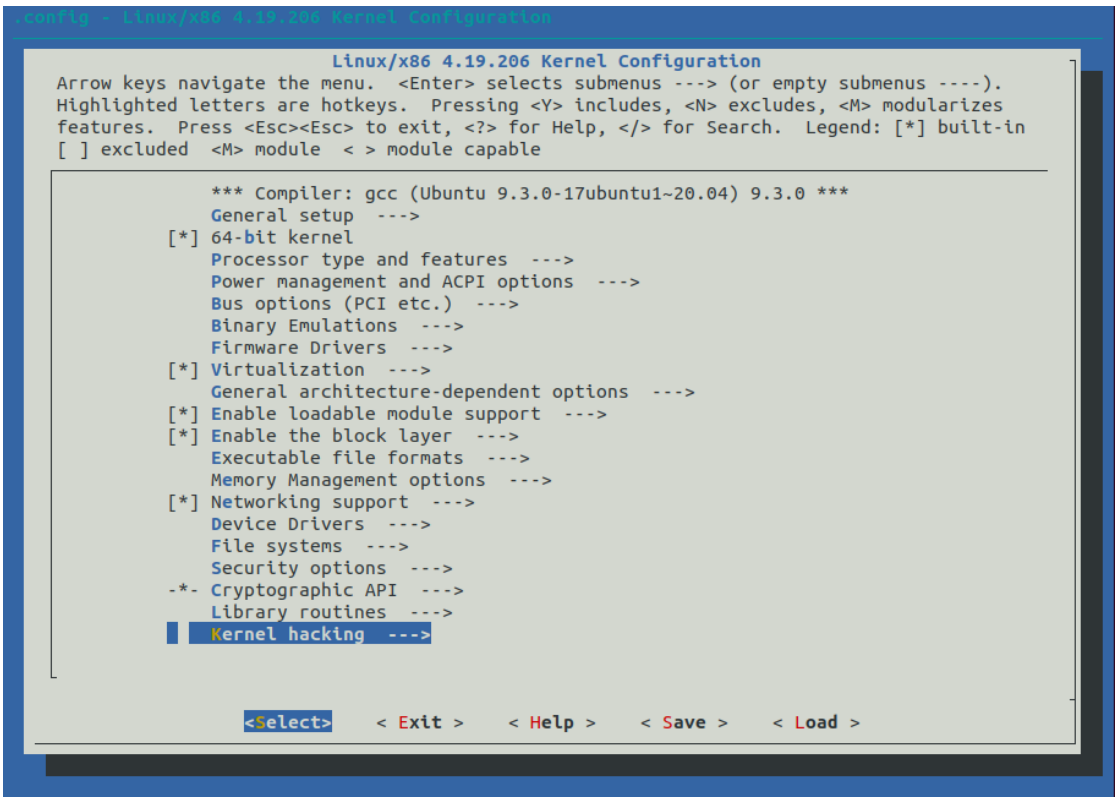


Figura 5.1: GUI per configurazione Kernel

Una volta conclusa la configurazione del Kernel attraverso "make" e "make modules_install" si avvierà la compilazione e l'installazione dei suoi moduli. Con "make install" installeremo il Kernel e il Grub verrà aggiornato.

5.3 Avvio con Kernel MPTCP abilitato

Per avviare Ubuntu con MPTCP abilitato, è necessario selezionare nel Grub "Advanced options for Ubuntu".

Figura 5.2: Prima Schermata del Grub

Successivamente, selezionare il Kernel installato che avrà ".mptcp" nel suo nome, quindi Ubuntu si avvierà.

5.4 Verifica

Dopo aver effettuato l'accesso al Desktop si esegua "sudo sysctl -a | grep mptcp" su terminale per verificare se MPTCP è stato abilitato. Output fornito:

kernel.osrelease = 4.19.126.mptcp

net.mptcp.mptcp_checksum = 1

net.mptcp.mptcp_debug = 0

net.mptcp.mptcp_enabled = 1

net.mptcp.mptcp_path_manager = fullmesh

net.mptcp.mptcp_scheduler = default

net.mptcp.mptcp_syn_retries = 3

net.mptcp.mptcp_version = 0

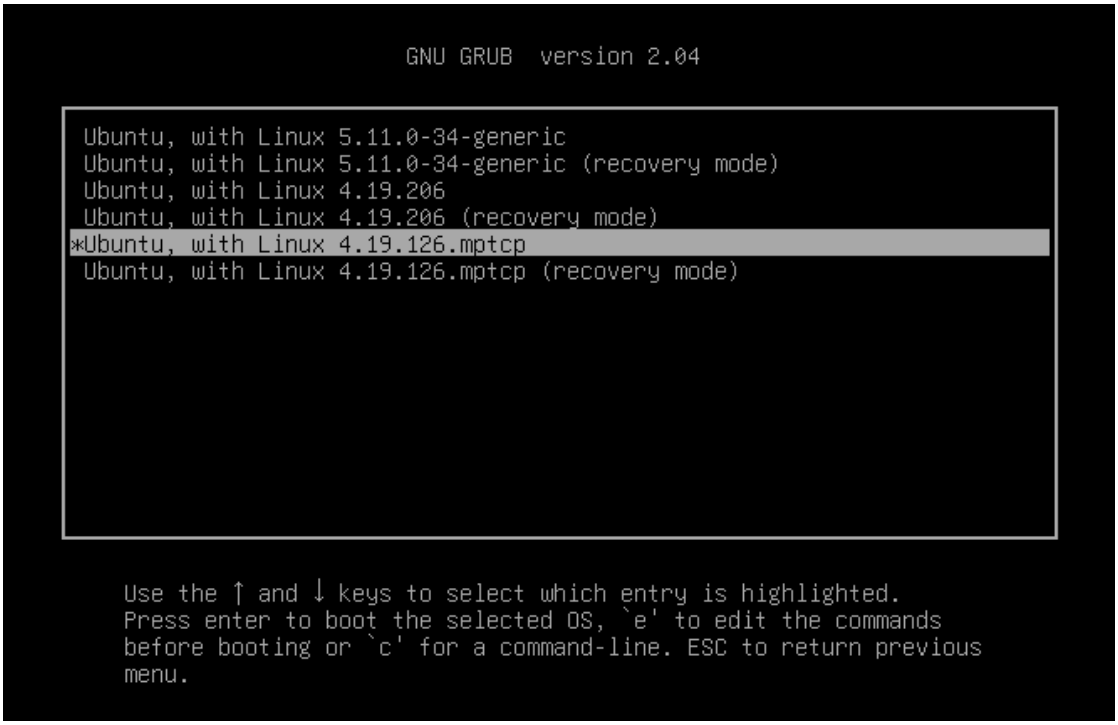


Figura 5.3: Schermata di selezione Kernel

Un ulteriore verifica è disponibile su amiusingmptcp.de dove il sito stesso verifica se l'host dispone di MPTCP, indicando subito il risultato.

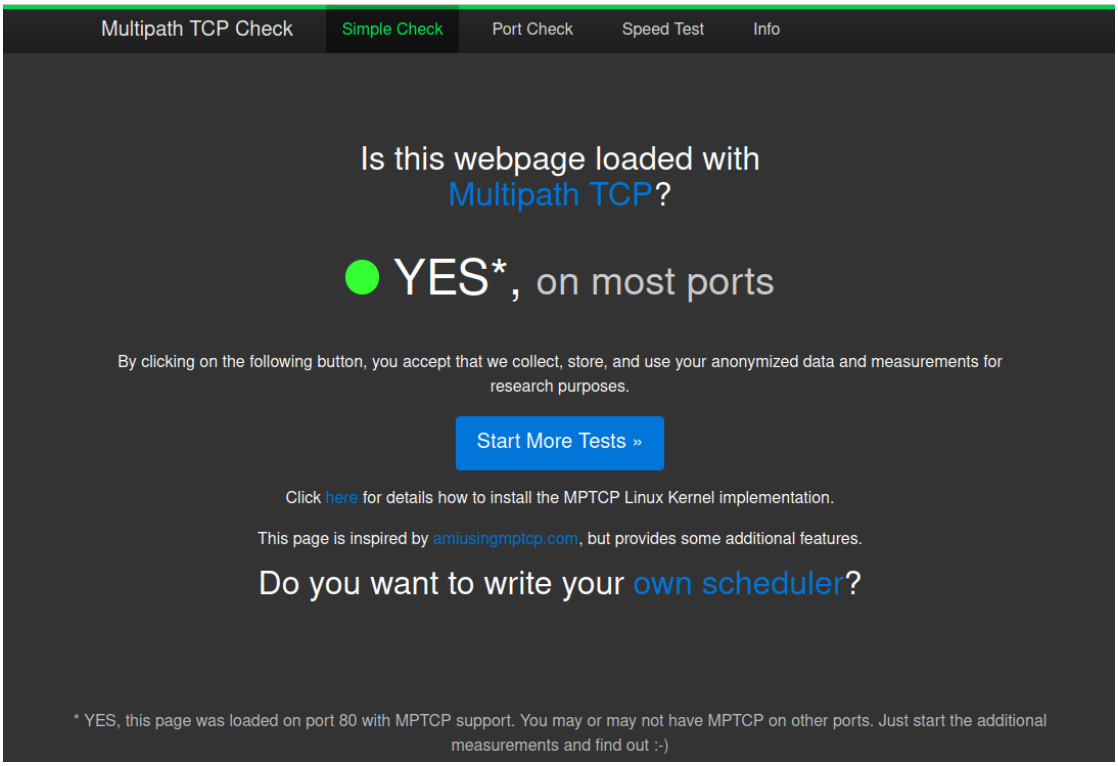


Figura 5.4: Verifica Web

5.5 Testing

Per il testing delle performance, sono state create 2 reti virtuali e ciascuna delle 2 VM è stata collegata ad esse. Gli applicativi utilizzati per il testing sono stati Wireshark, Curl, Ifstat e Speedtest-cli installati successivamente attraverso "apt install" in quanto non presenti nativamente nella distro utilizzata.

5.5.1 Primo Test

Come primo test è stata eseguita la cattura di pacchetti durante la connessione della prima VM su amiusingmptcp.de per verificare che la loro struttura sia coerente con quanto detto in precedenza all'interno di questo documento. Lo strumento Wireshark è stato utilizzato per esaminare i pacchetti. Di seguito vengono presentate le schermate dei messaggi più rilevanti del protocollo MPTCP, che vengono inviati per gestire la connessione, come descritto nella sezioni precedenti. Le informazioni MPTCP sono inviate nel campo TCP Options, come si può vedere nelle figure sottostanti.

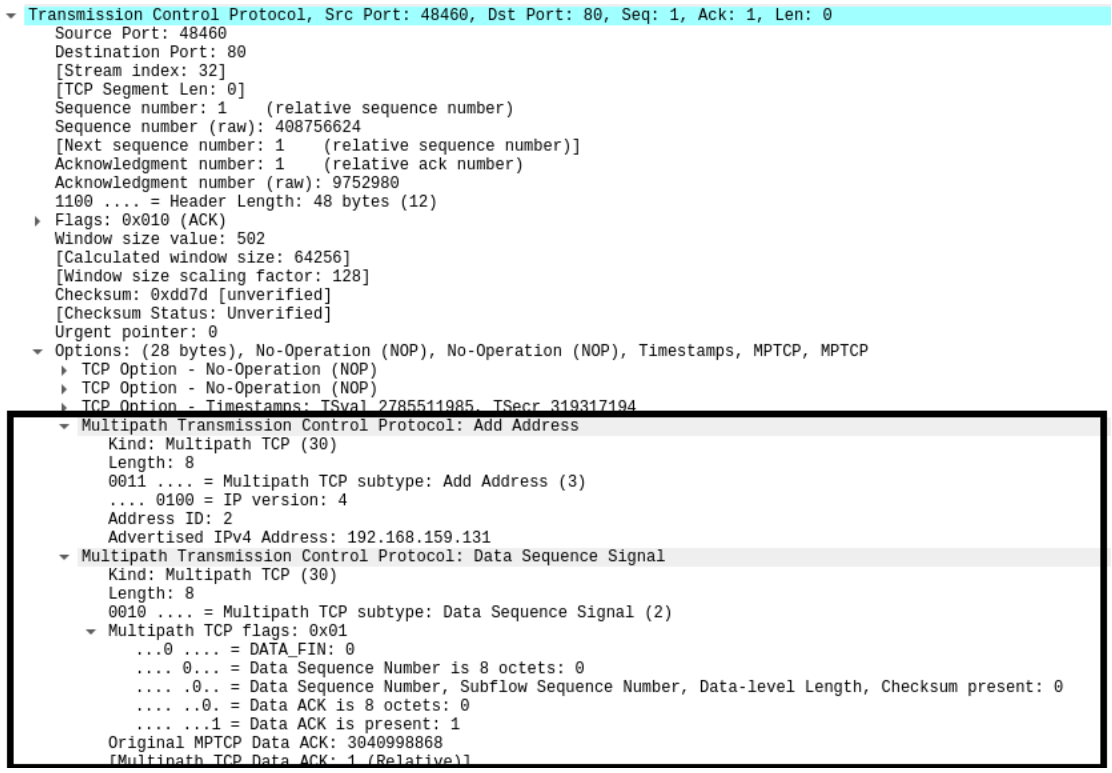


Figura 5.5: Wireshark - MPTCP messaggio inizio connessione

La figura 5.5 mostra l'opzione Add Address MultiPath TCP per stabilire una nuova connessione. Un host invia l'Add Address per annunciare ulteriori indirizzi in cui può essere raggiunto. Per identificare la nuova connessione il protocollo combina il nuovo indirizzo con l'id dell'indirizzo. Come si può vedere nella cattura, lo scambio di informazioni viene effettuato dal Server, informando il Client che contiene un altro indirizzo aggiuntivo, che appartiene all'altra interfaccia disponibile, invia anche l'id dell'indirizzo di protocollo che è 2.

La figura 5.6 mostra l'opzione MPTCP Join Connection, per permettere all'host di creare nuovi sottoflussi. Un host invia Join Connection, flag MP JOIN, per stabilire un nuovo subflow con un altro host, dopo che l'Add Address e l'id di protocollo sono già stati scambiati. Nella cattura il client scambia il token di identificazione e l'Address ID per creare un nuovo subflow con il server.

La figura 5.7 mostra il MPTCP che invia l'opzione DATA FIN selezionata per avviare la chiusura della connessione. Il protocollo imposta anche il flag TCP con FIN, per chiudere la connessione. Nella cattura il server sta inviando il DATA FIN su uno dei sottoflussi utilizzati, per chiudere tutti i sottoflussi.

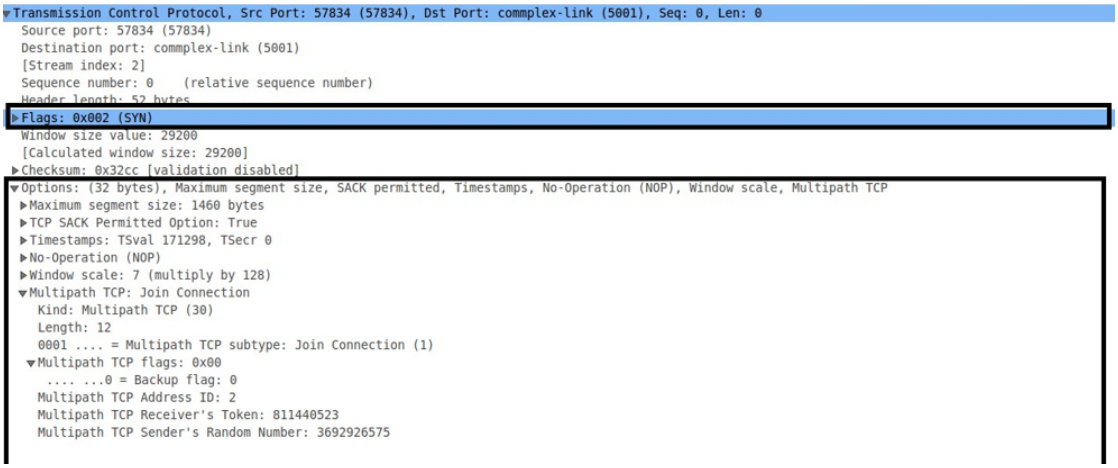


Figura 5.6: Wireshark - MPTCP messaggio creazione nuovo subflow

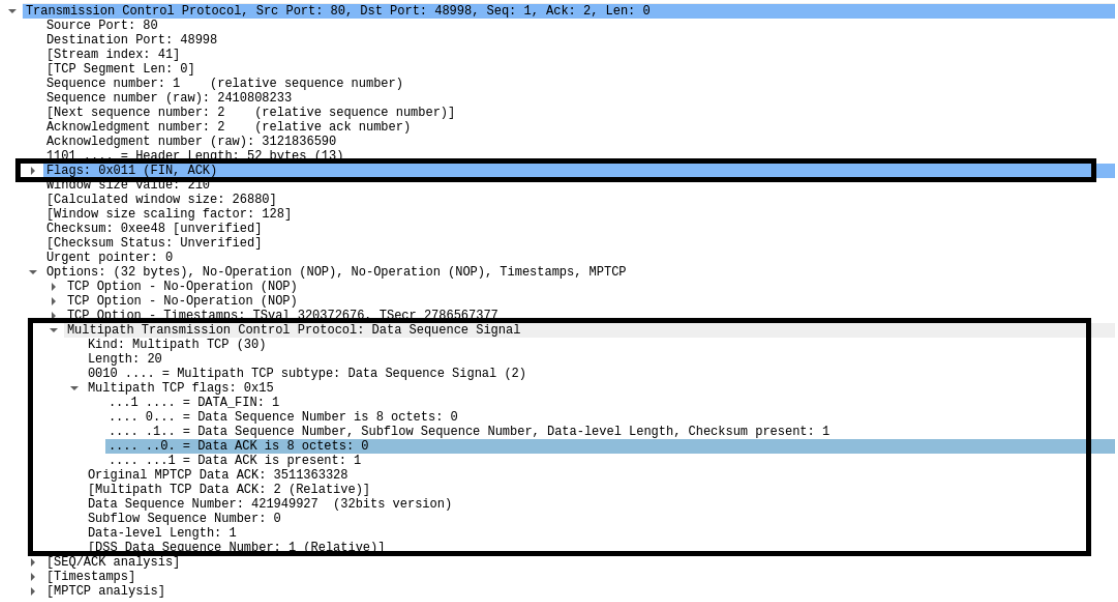


Figura 5.7: Wireshark - MPTCP messaggio per chiusura connessione

5.5.2 Secondo Test

Questo test è stato svolto per verificare le prestazioni di MPTCP all'interno delle 2 reti locali virtuali. In particolare utilizzando Curl e Ifstat è stato possibile visualizzare in tempo reale la velocità di trasferimento.

```
root@a:~# ping 10.0.1.2
PING 10.0.1.2 (10.0.1.2) 56(84) bytes of data.
64 bytes from 10.0.1.2: icmp_seq=1 ttl=64 time=0.952 ms
64 bytes from 10.0.1.2: icmp_seq=2 ttl=64 time=1.09 ms
64 bytes from 10.0.1.2: icmp_seq=3 ttl=64 time=1.06 ms
^C
--- 10.0.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.952/1.036/1.096/0.061 ms
root@a:~# curl 10.0.0.2/testfile > /dev/null
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  2 6000M    2  171M    0     0  28.8M      0  0:03:27  0:00:05  0:03:22 28.7M
```

Figura 5.8: Curl in esecuzione su Host1

```
root@a:~# ifstat
       eth0                eth1
 KB/s in  KB/s out   KB/s in  KB/s out
    0.00      0.00      0.00      1.93
    0.00      0.10      0.00      0.55
    0.00      0.00      0.00      0.47
    0.00      0.00      0.00      0.00
    0.00      0.00      0.00      0.00
  276.33  13341.67    311.56  14810.35
  353.92  17911.32    268.87  13620.30
  290.86  14626.60    254.54  12775.01
  389.98  19736.96    227.81  11483.98
  312.39  15888.00    364.47  18522.22
       eth0                eth1
 KB/s in  KB/s out   KB/s in  KB/s out
  299.59  15141.61    333.29  16883.37
  291.09  14703.93    217.14  11040.50
  295.65  14964.34    219.62  11131.37
  390.99  19849.60    251.59  12719.49
  336.88  17000.39    171.87   8641.16
  197.15  10058.61    237.11  12098.68
```

Figura 5.9: Ifstat in esecuzione su Host2

Si noti l'utilizzo simultaneo delle 2 interfacce eth0 e eth1 di Host2.

5.5.3 Terzo Test

Questo test verifica la differenza di prestazioni in latenza, download ed upload tra MPTCP e TCP utilizzando Speedtest-cli come applicativo per il test. Sono stati effettuati test multipli raggruppati per server al fine di aver un risultato equo. Si riportano i dati in tabella espressi in Mb/s(download e upload) e ms(latenza).

Server ID	TCP	MPTCP
2567	38.8DW 16.8UP 18.5ms	37.0DW 18.3UP 16.1ms
7839	38.0DW 19.2UP 22.4ms	38.7DW 15.9UP 16.2ms
11675	38.8DW 18.7UP 18.9ms	38.8DW 14.5UP 17.7ms
20551	38.8DW 19.1UP 21.3ms	38.1DW 19.5UP 18.4ms
25146	38.9DW 19.1UP 23.4ms	38.7DW 19.0UP 23.9ms
25258	38.8DW 16.7UP 19.9ms	38.8DW 19.7UP 18.6ms
34117	38.9DW 20.2UP 23.7ms	38.7DW 19.5UP 19.8ms
42070	38.6DW 18.9UP 38.9ms	38.5DW 19.0UP 38.9ms
43787	38.3DW 19.6UP 21.5ms	38.3DW 15.4UP 20.1ms
44718	37.7DW 18.8UP 19.5ms	37.2DW 17.1UP 14.2ms

Tabella 5.1: Confronto MPTCP TCP

Dalla tabella si nota che MPTCP impatta in maniera impercettibile in termini di performance.

Capitolo 6

Conclusioni

Dopo aver analizzato e testato MPTCP possiamo affermare che si tratta di un protocollo molto valido, in grado di migliorare TCP senza alterarne le performance. In particolare la possibilità di utilizzare più interfacce permette una maggiore resilienza ai guasti, rendendolo molto appetibile a livello Enterprise. Un altro punto a favore è sicuramente la totale compatibilità con TCP, rendendo questo protocollo il degno successore a livello globale di quest'ultimo, quando tutti i dispositivi connessi ad Internet saranno compatibili.

6.1 Considerazioni per il futuro

Il lavoro sviluppato finora, per un protocollo multipath a livello di trasporto, con un'implementazione del kernel Linux con MPTCP, ha permesso alla comunità di ricerca di studiare diverse topologie e implementazioni. Il lavoro futuro potrebbe concentrarsi sullo sviluppo di un'implementazione che permetta l'uso di un maggior numero di interfacce, con diversi tipi di connessione. Un altro punto di attenzione, potrebbe essere diretto allo sviluppo di un algoritmo di controllo della congestione, che permetterebbe al protocollo di competere più equamente con TCP. Un ulteriore approccio potrebbe essere quello di sviluppare applicazioni consapevoli di MPTCP permettendo di dare priorità al tipo di traffico in uso su ogni interfaccia.
Altre applicazioni potrebbero essere sviluppate per consentire la configurazione del kernel MPTCP negli smartphone, al fine di testare ulteriormente diversi approcci del protocollo per questo tipo di dispositivi e nelle connessioni senza fili.

Bibliografia

[1] Sébastien Barré, Christoph Paasch, and Olivier Bonaventure. Multipath tcp: from theory to practice. In *International conference on research in networking*, pages 444–457. Springer, 2011.

[2] et al. C. Paasch, S. Barre. Multipath tcp in the linux kernel. `https://www.multipath-tcp.org/`, 2021.

[3] Dinamene de Lima Correia Almeida Barreia. Multipath tcp protocols, 2014.

[4] Alan Ford, Costin Raiciu, Mark Handley, Sebastien Barre, Janardhan Iyengar, et al. Architectural guidelines for multipath tcp development. *IETF, Informational RFC*, 6182:2070–1721, 2011.

[5] Alan Ford, Costin Raiciu, Mark Handley, Olivier Bonaventure, and C Paasch. Rfc 6824: Tcp extensions for multipath operation with multiple addresses. *Internet Engineering Task Force*, 2013.

[6] J. Postel. RFC 793: Transmission control protocol, September 1981. See also STD0007 [**?**]. Status: STANDARD.

[7] Costin Raiciu, Mark Handley, and Damon Wischik. Coupled Congestion Control for Multipath Transport Protocols. RFC 6356, IETF, October 2011.

[8] Michael Scharf and Alan Ford. Multipath TCP (MPTCP) Application Interface Considerations. Informational RFC 6897, IETF, March 2013.

Capitolo 7

Ringraziamenti

A conclusione di questo elaborato, desidero menzionare tutte le persone, senza le quali questo lavoro di tesi non esisterebbe nemmeno. Ringrazio il mio relatore Callegati Franco, che in questi mesi di lavoro, ha saputo guidarmi, con suggerimenti pratici, nelle ricerche e nella stesura dell'elaborato. Ringrazio di cuore i miei genitori. Grazie per avermi sempre sostenuto e per avermi permesso di portare a termine gli studi universitari. Un ringraziamento particolare va a tutti i miei amici e compagni di studio in questi anni, con il loro sostegno mi hanno aiutato ad arrivare fin qui. Infine, vorrei dedicare questo piccolo traguardo a me stesso, ai miei sacrifici e alla mia tenacia che mi hanno permesso di arrivare fin qui.