Using MPTCP subflow association control for heterogeneous wireless network optimization

Jianwei Liu, Anjan Rayamajhi, James Martin
Clemson University Clemson, SC 29634
{jianwel, arayama, jmarty}@clemson.edu

Abstract—Multipath TCP (MPTCP) was designed to increase the throughput and reliability of TCP, with specific motivation coming from scenarios including data center and cloud computing. The use of MPTCP has been recently explored to support heterogeneous wireless networks (HetNets) involving hosts that have multiple network interfaces. However, current solutions generally involve many simplifying assumptions. In this paper, we propose a new framework to collect scheduling information from various scheduling network entities and conduct optimization from a global view. The framework uses the existing or readily accessible MPTCP parameters. Under this framework, we introduce a centralized optimization algorithm to realize general proportional fairness of user throughput. Based on results from NS3 simulations, we provide evidence that the approach provides a low cost solution for improved performance from the perspectives of both applications and network operators.

I. INTRODUCTION

The explosive growth of mobile data has forced wireless operators to rethink deployment and technology directions. A recent report from Cisco suggests that the volume of mobile data traffic in 2019 will be 10 times larger than that in 2014 [1]. Optimal resource allocation strategies will be required to meet this demand. While the academic community has studied resource allocation in wireless HetNets for more than 20 years, the vast majority has been more theoretic in nature rather than applied for practical systems. Early work that did take real systems into account focused on the network selection decision to achieve better fairness or improve system efficiency [2], [3], [4], [5]. More recently, Multipath TCP (MPTCP), a standardized TCP extension that enables a TCP session to use multiple (sub-)flows over multiple interfaces [6], has been considered in HetNets.

Most of the published MPTCP research has focused on the congestion control algorithm design for wired network [7], [8], [9], [10]. There are two established operating modes for MPTCP, fullmesh mode and backup mode. The former will try to send data over all available interfaces while the latter limits the use of all subflows other than the primary for backup purposes. One MPTCP session configured with several subflows via HetNet enhances reliability and minimize the impacts of handoffs. [11] introduces a mobility architecture based on the fullmesh mode, and compares the throughput and energy performance with the single-path TCP. The results show that MPTCP operated in fullmesh mode can boost throughput, but reduce energy efficiency. Therefore, [12] further explores the backup mode which enables smooth handover of TCP sessions, and potentially higher energy efficiency.

While much of the published research related to MPTCP does lead to improved throughput, fairness, or robustness, there are few studies that explore the use of MPTCP in a HetNet with the specific goal of globally\footnote{This global concept is defined inside some area, e.g., the area covered by an LTE BS.} optimized allocation. It has been pointed out that an MPTCP subflow controller that decides when a flow/interface should be opened or closed could benefit from global information [13]. Globally optimized allocation can lead to improved spectral efficiency and fairness among users. On the other hand, user considerations must be taken into account as the use of a subflow consumes power. These issues are complex and represent open areas of research [14].

The research presented in this paper is based on our observation that the backup mode concept represents an effective mechanism to apply centralized resource allocation result in large scale HetNets. The advantage of this approach is increased reliability as MPTCP can enable backup interfaces when associations break. Several studies have addressed the issue including [15] which explores MPTCP to boost fairness in a HetNet. The results suggest, as one would expect, that greedy devices that utilize all available interfaces may result in a non-Pareto optimal allocation. The authors propose both distributed and centralized approximation algorithms to achieve better \( \alpha \)-fairness among users. The distributed algorithm selects the second interface based on the ratio of the second best Reference Signal Received Quality (RSRQ) and the best RSRQ. A centralized greedy heuristic is proposed which repeatedly iterates through all interfaces trying to find the association that leads to the most improved utility.

The problems with this approach include: 1) 3GPP standards would have to change; 2) Information required from the network, such as RSRQ, might not be available; 3) The approach does not operate at the MPTCP layer and therefore cannot serve as an MPTCP association controller; 4) The throughput model fails to take into account dynamics from the network and upper layers.

In this paper, we present and validate an approach to optimize the resource allocation in a HetNet using an approach that leverages MPTCP. We provide a sample centralized scheduling algorithm, and demonstrate how it can be used to implement the subflow association controller of MPTCP [13]. The approach utilizes information available from the two ends of the TCP session and treats the underlying network as a blackbox.
Our results show that the approach allows the scheduler to more accurately estimate flow throughput (compare with [15], [3]) which is necessary for improved optimization results.

Our system involves mobile devices that send periodic reports of observed link status measurements and usage. The scheduler then predicts subflow and Mobile Node (MN) throughput from the current report and historical report data. It makes decision on which flow should be kept alive or suspended. Though any objective can be set to work with the proposed framework, we provide a sample algorithm that takes into consideration global objectives defined with utility functions based on throughput and fairness criteria.

The contributions of this paper are threefold,

1) We propose a system-wise interface association scheduling framework based on MPTCP.
2) The proposed framework can at the same time serve as the subflow controller of MPTCP stack.
3) A centralized algorithm that utilizes the collected scheduling information is proposed, and evaluated compared to other methods in literature.

The paper is organized as follows. Section II introduces selected related work of this paper. Section III details the design of the proposed framework and algorithm. Section IV evaluates the proposed framework and algorithm using NS3 simulations. We summarize and present our conclusions in Section VI.

II. RELATED WORKS

Most of the published research on MPTCP focuses on congestion control schemes. [7] provides a simple extension of running TCP NewReno on each subpath. This algorithm, however, can be Single-path TCP (SPTCP) unfriendly. This motivates the coupled congestion control algorithm which is fair because it has the same underlying utility function as TCP NewReno, e.g. [16]. The current default congestion control algorithm for MPTCP, called Linked-Increases Algorithm (LIA) [8], is more responsive than the coupled algorithm. But, it is sometimes excessively aggressive toward SPTCP users without any benefit to multipath users [9]. OLLA [9] and Balia [10] are proposed to solve this problem. [17] investigated the improvement space when delay is of concern in the wireless network for real-time applications. While end-to-end congestion control algorithms can prevent congestive collapse, it only is reactive. The proposed framework functions differently with a subflow association controller that can control the association of interfaces proactively.

While system-wise association optimization has not been studied extensively in the context of MPTCP, there is a tremendous amount of research that explores optimal association and load balancing in HetNets. Bu. et cl. [2] introduces the general proportional fairness (GPF) problem, which aims for proportional fairness of user throughput under multiple overlapped cellular Base Stations (BS). It tries to achieve a trade-off between maximizing the sum rate and the fairness of users by utilizing the close-formed long term optimization goal of proportional fairness schedulers. However, as noticed in that paper and some other later publications, the problem is NP-hard to solve. Therefore, the idea in [2] is to map it to a 3-d maximum weight matching problem by fixing the number of users associated to every BS. It is easy to see that with the increasing number of users, the computation cost increase exponentially.

Ye. et cl. [3] tried to solve a similar problem but in a cellular HetNet context. Based on similar assumptions in user traffic and fading model, they proved the PF scheduler can be reduced to a round-robin time sharing scheduler. They model the problem based on this assumption with an objective similar to [2]. Furthermore, they propose a solution to the problem using Lagrangian dual decomposition relaxation. A distributed algorithm with lower computation in the central server is also proposed. However, the convergence of this algorithm in real system with large number of MNs is not guaranteed.

We identify the following issues related to [2], [3]:

1) Handoffs. Those literature do not explicitly describe handoff solution, which is critical for mobile network overall performance. Even though Mobile IP based solutions can be adopted to alleviate this problem, it requires heavy network side deployment. An approach based on MPTCP provides a simple end-to-end solution independent of underlying networks.
2) Traffic model assumption. The literature assumes backlogged traffic, which is not realistic, and leads to inaccurate estimation of flow throughput.
3) MN connection assumption. The literature assumes a MN is limited to one network at a time. The usage of MPTCP can solve that limitation and introduce new optimization possibilities as envisioned by [15], and additional reliability.

III. FRAMEWORK DESIGN

A. System architecture

Fig. 1 shows the system diagram of the designed framework. Mobile nodes (MN) and Corresponding Nodes (CN) are both MPTCP compatible. MN runs N_{app} applications (apps) to various CNs, each of which can run one or more applications. Each app can utilize one or more subflows via HetNet and multiple
routes. The CNs can be either wired servers or mobile devices. MN and CN both collect necessary information for scheduling from the MPTCP stack and lower layers. (We detail what information needs to be collected in section III-G). Information is aggregated at the MN side and sent to a Regional Scheduling Server (RSS) (Mobile CN is actually an MN in another domain managed by a different RSS). The RSS is an edge computing server that connects to the backhaul of all the regional wireless networks. Its IP address is known to every BS in that region. We use the terms BS and AP interchangeably in this paper. Each app on MN initiates a connection to the RSS using new MPTCP connection. Via that connection, scheduling information is periodically sent to the RSS. The RSS runs periodically optimization algorithm and returns the association plan to the app. Each association plan consists of tuples in the format of \{SRC_IP, DEST_IP, IS_BACKUP\}. SRC_IP and DEST_IP are the source and destination IP respectively. IS_BACKUP controls whether backup mode should be enabled. The MN then controls subflows based on it and its device level information, like its current battery level. We assume there are incentives from the network/carrier side to encourage users to adopt the network-side decisions whenever possible. E.g., every time a user conform to the decision to close a subflow, they receive credits which can grant priority to her later. Much analysis can be conducted here, however, this is not the focus of this paper. Note that single-path TCP connections can also report to RSS, or receiving scheduling decision. The only difference is the number of subflows when collecting flow statistics, and the loss of reliability and transparent handover benefits.

We want to first clarify the concept of subflow used in this paper. As illustrated in Fig. 2, an MPTCP connection can be seen as a single socket connection from user application’s perspective. Each MPTCP connection operates over one or more paths. MPTCP can even run multiple subflows on a single path. In this paper, we first discuss the solution for a single-subflow-per-path setting, and then the extension for the multiple-subflow-per-path setting in section III-I (Therefore, the term subflow below means the same as path).

In the device level, as shown in Fig. 2, the proposed framework consists of an information collection module (ICM), a scheduler communication module (SCM) and a MPTCP subflow association controller module (SACM). The ICM retrieves subflow statistics such as RTT, loss rate from MPTCP stack, and sends them to the SCM. The SCM is responsible for communicating with the RSS. SACM controls the association of MPTCP subflows based on the centralized decisions received from SCM. The SCM should be implemented outside the Linux kernel, while the other two modules can be implemented either as kernel modules or as user space applications.

B. Network model and terminology

In this paper, \(i\) is used for the indexes for users, while \(a\) for apps, \(j\) for APs and \(k\) for subflows. Therefore, \(f_{iak}\) means the \(k\)th subflow of user \(i\)’s \(a\)th application \((Appia)\). \(T_{iak}\) denotes the goodput of subflow \(f_{iak}\), while \(T_i\) is the total goodput of user \(i\). The term goodput is defined as the application level throughput. \(T_i = \sum_a T_{iak}\), where \(T_{iak}\) is the goodput of application \(a\). As observed in [18], the overall throughput of MPTCP connection over multiple network interfaces depends on whether the traffic is network-bounded. To generalize the model, we think \(T_{iak} = \sum_k \alpha_k * T_{iak}\), where \(0 \leq \alpha_k \leq 1\), and \(\sum_k \alpha_k \geq 1\). Every BS has a known capacity \(C_j\). We think a BS is overloaded when \(\alpha_{\text{thresh}} \%\) of its capacity is used, as indicated in Fig. 6 of [19]. \(K_j\) denotes the set of subflows under \(AP_j\), while \(M_j\) denotes the set of users under \(AP_j\). \(N_{uj}\) is the number of users under \(AP_j\), i.e. \(N_{uj} = |M_j|\). \(K_{ia}\) denotes the set of flows created by the \(a\)th application of user \(i\). \(T_j\) is the total goodput of users under \(AP_j\), i.e. \(T_j = \sum_{i \in M_j} T_i\). We denote the scheduling period of RSS as \(t_s\). The total number of users is denoted as \(N_i\), that of apps as \(N_a\), and that of flows as \(N_k\). The total of applications of user \(i\)’s application \(a\) is denoted as \(N_{ia}\). If any new subflow emerged between \(t_s\), it will start using the MN’s local setting, which can be full-mesh mode, back-up mode, etc. The RSS will later suggest MNs to close any unnecessary subflows.

C. Optimization objective

We design the scheduler to optimize a system-wide utility function \(U\). There are many possible optimization objectives, such as proportional fairness [2] and \(\alpha\)-fairness [15]. In this paper, we provide one exemplar optimization goal using general proportional fairness (GPF). Proportional fairness is commonly used in cellular networks as it provides an acceptable trade-off between total throughput and fairness of the system. We also introduce additional constraint to take into consideration connection reliability and system level congestion avoidance. The centralized scheduler tries to optimize system-wide utility \(U\) in the next scheduling time interval \(t_s\), by varying selecting the primary path and identifying the backup paths on an application basis.

Therefore, we model the optimization problem as follows,

\[
\begin{align*}
\text{Maximize} & \quad \sum_j \sum_a \log(T_{iak}[P]) \\
\text{subject to} & \quad \sum_k x_{iak} \geq 1; \\
& \quad \sum_i T_i \leq \alpha_{\text{thresh}} * C_j, \forall j \in B; \quad (1) \\
& \quad T_{iak} = \sum_a \alpha_k * T_{iak}; \\
& \quad x_{iak} \in \{0, 1\};
\end{align*}
\]
where $\mathcal{B}$ is the set of BSs in the system, and $x_{iak}$ the boolean association variable representing whether $f_{iak}$ is enabled. The path is enabled when $x_{iak} = 1$. Configuration $P$ is a vector of $x_{iak}$ for all the paths available in the system. $T_i[P]$ is the application layer goodput of user $i$. The third constraint defines how it is calculated. The first constraint ensures the reliability of mobile applications. It means every MN must use at least one interface. We call the path that is always maintained the primary path. The primary path should be designed to avoid changing frequently on account of handover cost. The second constraint assures that BS capacity limit will not be exceeded, which aims to prevent network congestion.

### D. TCP subflow goodput estimation

Schedulers need to make accurate goodput estimations of $T_{iak}[P]$, so that correct decisions can be made. The configuration $P$ here can be the current configuration ($P'$) or any planned configuration the scheduler needs to search for. In [15], SINR is used to model the goodput of a flow no matter it is an existing flow or not. We design a better goodput estimator by differentiating existing flows and new flows, and utilize link status measurements inside MPTCP stack.

#### 1. Existing flows

The assumption is that goodput of a flow is can be predicted with high precision from the time-average of previous records. We verify this assumption in Section IV-A.

1) The ICM module estimates the goodput of the current subflow configuration $(T_{iak}[P'])$ in the next $t_s$ by a time-average of $T_{iak}$ in the past $N_w t_s$s. The ICM module obtains the measurements from byte counters inside every subflow’s TCP (meta-)socket [13].

2) To estimate the $T_{iak}$ of any configuration $P$, the RSS estimates it based on $T_{iak}[P']$ and $T_j$. The goodput of an association plan is modeled based on the current goodput and the load on $AP_j$, i.e. $T_{iak}[P] = f(T_j, T_j') + T_{iak}[P']$, where $f(T_j, T_j')$ is a piecewise function that maps to a scalar. For example, one simplified version of $f$ we used in the evaluation section is $f(T_j, T_j') = T_j / T_j'$.

#### 2. New flows

When an MN roams to a new AP, or a new subflow is available near or at the beginning of a scheduling period $t_s$, the above estimation method cannot be applied. Therefore, we need a model to predict the throughput of a TCP flow. Matthew Mathis et al. proposed a simple model based on some link characteristics, such as the Maximum Segment Size (MSS), Round-Trip Time (RTT) and packet loss rate (LossR) [20].

\[
T = \frac{MSS \cdot C}{\sqrt{RTT \cdot LossR}} \quad (2)
\]

where $C = \sqrt{2}$. The detailed proof of how this is derived is explained in [20]. This paper uses a similar model, but further considers other factors like the demand of flows. We choose this model because of its simple form and verified accuracy. The simple form can reduce the overhead introduced by the framework, and the complexity of the machine learning algorithms to build the lookup table in the next section.

The MSS in the Eq. 2 is reported from every subflow after connection initiation. If the subflow is not connected and MSS not yet negotiated, the RSS uses 1400 bytes as the default value of MSS as described in the Internet and MPTCP standards. To achieve the RTT in the Eq. 2, every subflow will send one keepalive packet during one $t_s$, no matter data transfer is enabled or not. RTT of every subflow will then be always available from the statistics inside subflow sockets.

To estimate the loss rate ($\text{Loss}_R$) of subflows, we think about two cases,

1) At least one subflow is transferring data at the same interface. $\text{Loss}_R$ can be estimated from the loss rate of the other running flows at that interface. A simple method is to use the average loss rate of all the other running flows.

2) No subflow sending data on that interface currently. Then the RSS estimates the loss rate from historical data. We explain this in detail in section III-E.

#### E. Loss rate estimation for new subflow without any reference

To estimate the loss rate of a new subflow without any reference subflows at the same interface, the RSS needs a model to predict subflow loss rate. We assume the loss rate is related to the overall goodput of $AP_j$ which the new subflow connects to ($T_j$), and the SINR of the MN to that AP ($\text{SINR}_{ij}$). Therefore, we propose the following model to predict the loss rate given a configuration $P$.

\[
\text{Loss}_{Riak}(P) = g(T_j, \text{SINR}_{ij}) \quad (3)
\]

where $g$ is a function that is formed overtime using historical data at the RSS from MNs.

#### F. Rationale behind the system

The framework is designed to estimate the subflow goodput with more accuracy. Most of the existing flows can use time-averaged goodput measurement instead of prediction, like in [15]. The measurements take into account end-to-end dynamics in network and the above layers, like changes of kernel buffer size, or the application sending rate, etc. Secondly, a few designs are aimed to minimize the system overhead. Only the flows of the case in section III-E need SINR to predict goodput. Therefore, SINR is only sent when a new subflow is available, or it has changed by certain threshold for several $t_s$. Other SINR reports are sent for training data, and therefore can allow collection of larger time scale. This greatly reduces the overhead comparing with other SINR based scheduling algorithm like [15]. Thirdly, RSS predicts loss rate instead of goodput. This is because goodput is related to the demand/sending rate of the application, while loss rate is less related, and therefore can be better predicted with historical data.

#### G. Parameter measurements at Mobile Nodes

Both ends of the MPTCP subflow collects scheduling information of the current subflows. To summarize the information needed, 1) For any subflow, RTT and negotiated MSS need to
be collected at the beginning of connection establishment. 2) For existing flows, MN needs to send the time-averaged $T_{iak}$, $RTT_{iak}$, and $SINR_{ij}$ to the RSS every $t_s$. However, to reduce the overhead, those values only need to be sent at subflow establishment. MNs also monitor the change of $SINR_{ij}$ in every $t_s$, and only one subflow at every interface need to send update when the change is over certain threshold.

H. Decision algorithm

We design an algorithm that tries to achieve the optimization objective in section III-C, i.e. general proportional fairness over all the user goodput. As shown in Algorithm 1, the algorithm has four stages, i.e. initiation, primary path determination, extra subflow pruning, and new subflow enrollment.

1) **Initiation** (line 1-5). For every scheduling interval, the RSS first scans for running subflows that have submitted their reports, and then calculates the $T_{iak}[P']$ using the method in section III-D, III-E. The RSS then puts the subflows into bins of users, and calculates $T_{ij}[P']$. Overall utility $U(P')$ can be therefore calculated based on $T_{ij}[P']$. RSS sorts the list of subflows of every user $i$'s application $f_{siak}$ into decreasing order of $T_{iak}$. Step 2) and 4) use this order. $P_0$ denotes the final configuration decided by the RSS, and is initiated with $P'$. If $\text{max}(\{K_{ia}\})$ can be seen as a constant, the time complexity of the sorting is $O(N_u)$. Therefore, the overall time complexity of this stage is $O(N_k)$.

2) **Primary path determination** (line 6-10). If the first subflow in $f_{si}$ is not the current primary path for user $i$, and it is a stable connection whose goodput is larger than the current primary path over $T_{diff}$%, the first subflow in $f_{si}$ is set to primary path. To test whether a path is stable, the RSS simply examines the goodput of that path in past several $t_s$. If it has $n_{ps}$ intervals which $T_{iak} > \text{TH}$, it is considered stable. $\text{TH}$ is the threshold, and can be set to the average subflow throughput of the app so far. The time complexity of this stage is $O(N_k)$.

3) **Extra subflow pruning** (line 11-26). If $T_j$ of any $AP_j$ exceeds its limit, certain subflows will have to be turned into backup mode or shutdown to avoid congesting. The RSS first runs one scan of all existing non-primary subflows under $AP_j$ in decreasing order of their downlink SINR (if tie on SINR, sort the subflows into increasing order of $T_{iak}$) and set $x_{iak}$ to 0, until the estimated $T_j$ is less than the congestion limit. If the limit was still exceeded, the RSS further process the primary path subflows the same way. This procedure will be done for every AP. We see that the time complexity of this administration control is $O(\sum_{j=1}^{[B]} |K_j| \log(|K_j|))$, where $K_j$ is the set of subflows under $AP_j$.

From current configuration $P'$, it is possible to close any subflow to get a configuration $P$. The RSS basically checks every subflow and see whether any secondary subflows can be closed so that overall utility can be improved. However, the order of this procedure is more carefully designed comparing with [15]. From the user level, the RSS follows an increasing order of user scheduling priority $P$. Note that $P$ can be related to the percentage of times that MN conformed to centralized decisions. It can also adopt weights that operators want to differentiate users of various cellular plans. Furthermore, in the application level, the RSS trials to prune several applications from each user, randomly selected or based on application types. In the subflow level, smaller subflows are checked first. Because we get achieve more fine-grained control by killing the smaller subflows first. The time complexity of this stage is $O(N_k)$.

4) **New subflow enrollment** (line 27-34). For the procedure to enroll new subflow, it includes subflows closed in the previous scheduling periods, and those from new users or new available paths. The procedure is similar to step 3), except that we check whether the capacity limits of BSs will be exceeded every time trying to enroll a new subflow. The time complexity of this stage is also $O(N_k)$.

Therefore, the overall time complexity of the proposed algorithm is $O(N_k)$.

**Algorithm 1:** Centralized Association Optimization

```
// Initiation
for every subflow connection f do
  add apps into a list $A_i$;
  add subflows of the user i’s app a into $f_{siak}$;
  calculate $T_{iak}$ for every subflow; update $T_{ij}$;
end

// Primary path determination
for every existing user i do
  if $Prime_i \neq \text{GetIndex}(f_{siak}[1])$ & $T[f_{siak}[2]] - T[f_{siak}[1]] \geq T_{diff}$ & $\text{IsStable}(f_{siak}[2])$ then
    $Prime_i = \text{GetIndex}(f_{siak}[2])$; update $P_o$;
  end
end

// Extra subflow pruning
for every AP $j$ do
  while $T_j \leq o_{thresh} \times C_j$ do
    $x_{iak} = 0$ in order, $\forall f \in K_j$ and not prime; update $P_o$;
  end
  if $T_j \leq o_{thresh} \times C_j$ then
    $x_{iak} = 0$ in order, $\forall f \in K_j$ and prime; update $P_o$;
  end
end

for every user i that has unpruned apps in increasing order of $P$ do
  pick $\omega_{i}(A)$ applications;
  for every application a picked do
    for subflow $f_{siak}[2] \in f_{siak} do
      try to close the subflow to get $P$; and calculate $U(P')$;
      if $U(P') > U(P_o)$ then
        $x_{iak} = 1$; $P_o := P'$;
      end
    end
  end
end

// New subflow enrollment
for every user i that has apps not scanned in increasing order of $P$ do
  pick $\omega_{i}(A)$ applications;
  for every application a picked do
    for subflow $f_{siak}[2] \in f_{siak} do
      try to open the subflow to get $P$; and calculate $U(P')$;
      if $U(P') > U(P_o)$ & $T_j \leq o_{thresh} \times C_j$ then
        $x_{iak} = 1$; $P_o := P'$;
      end
    end
  end
end

// Prime$i$: primary subflow index of user $i$
```

I. Possible extensions

To support multiple applications per MN, the information collection part needs to add application ID for every subflow. In the algorithm, Allowing multiple-subflow-per-path means
more fine-grained control to the subflow association. The algorithm can therefore be updated to consider the impact of multiple-subflow-per-path, and make decision on how many subflows should be enabled for each path. This can allow for additional optimization, but it not the focus of this paper.

The problem of scheduling for uplink is the difficulty to achieve uplink SINR. However, the scheduler may get that from BSs, or infer it based on the loss rate at the CN side. We will deal with those problems in future work.

IV. PERFORMANCE EVALUATION

<table>
<thead>
<tr>
<th>Simulation Parameters</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of mobile nodes</td>
<td>20</td>
</tr>
<tr>
<td>WiFi propagation loss model</td>
<td>FriisPropagationLossModel</td>
</tr>
<tr>
<td>WiFi downlink bandwidth</td>
<td>5MHz</td>
</tr>
<tr>
<td>WiFi downlink frequency</td>
<td>2.12GHz</td>
</tr>
<tr>
<td>Movement speed</td>
<td>m/s</td>
</tr>
<tr>
<td>Scheduling period</td>
<td>15s</td>
</tr>
</tbody>
</table>

To verify the feasibility and performance of the proposed framework, we evaluate it inside NS3, compared with the existing methods. We test the performance of the proposed framework in two aspects, i.e., the goodput prediction accuracy and the network performance.

A. Goodput prediction accuracy verification

For the goodput prediction verification part, we use the Direct Code Execution (DCE) version of NS3, which can use the network stack of a real Linux kernel for simulation, so that we can use the latest MPTCP implementation in the MPTCP branch of Linux kernel. To use the “backup” mode of MPTCP, we upgraded the default MPTCP in DCE to MPTCP v0.89, and compiled the MPTCP patched iproute.

We compare our framework with the goodput prediction accuracy in [15] (denoted as SINR_based model). Due to the use of real Linux kernel as network stack in DCE, it is difficult to monitor goodput of individual interfaces from NS3. Therefore, we verify the prediction accuracy for LTE interface and WiFi interface respectively. For example, when running the simulation for LTE, we close the WiFi interface of all the MNs. For WiFi, we use the 802.11g MAC of NS3 WiFi module. For LTE, we use the LENA LTE module in NS3. As shown in Fig. 3, we set one WiFi AP and one LTE BS at the center of a circle with radius R, and randomly distribute 19 nodes inside the circle. The 20th node is a special controlled node, which moves in a constant speed from (0,0) to (-R, 0). For WiFi, R = 80m, which is the maximum radius we found that will not result in zero iPerf values with most of the speed settings below. For LTE, R is selected to be 6000m. There is an equal number of CNs at the other end of the network, connected by a switch. Every CN initiates an iPerf TCP sending session to an MN, so that the iPerf results are not limited by the network capacity of a single CN. For every t_v second, we make iPerf print the test result, which serves as the ground truth goodput one MN can achieve (T_{real}). At the same time, the scheduler predicts the goodput (T_{predict}) using two different methods, i.e., SINR_based method and our proposed method. For the SINR_based method, the equation we used is similar to the one in [15], i.e. $T_{predict} = \beta * \gamma * BW * \log_2 (1 + SINR)/N_{u_j}$. BW is the downlink bandwidth of certain radio access technology, like LTE or WiFi. We use the value listed in Table I in our evaluation. $\gamma$ is the efficiency of the network access technology in various layers. We use $\gamma = 0.33$ for WiFi, and $\gamma = 0.57$ for LTE based on the data in [21]. $N_{u_j}$ is the total number of users connected to $A_{P_j}$. $\beta$ is an additional parameter we used to fit the SINR_based method to our best effort with the iPerf result from NS3 DCE. In essence, we achieve $\beta$ from the ratio of the closest predicted value and iPerf result over time. We then compare the prediction accuracy, which serves as the input for our second simulation.

In the experiment, we vary the MN speed in the set of $\{0, 0.57, 15, 30\}$ m/s, which represents static, walking, city driving, and highway driving speed respectively. We would like to see the influence of moving speed to prediction accuracy, since the historical data based prediction can be sensitive to speed. We run every simulation for 100s, and the scheduler predicts for every 15s. The top half of the Table I shows the important parameters in the NS3 DCE simulation. The prediction error rate (PER, denoted as $P_e$) is calculated as $T_{predict} - T_{real}$. (When we talk about prediction accuracy, it means $1 - P_e$). We first see how our proposed goodput prediction method works compared with the SINR_based method. In Fig. 4(a) and Fig. 4(b), we show the average PER of both methods on LTE and WiFi interfaces respectively (Whenever we talk about average PER, the data is preprocessed into absolute values). First of all, we can see the proposed method achieves much lower PERs for both interfaces. In case of LTE, the maximum PER is only around 0.088, while the standard deviation among users with various locations is only 0.063. We note that the prediction is not that sensitive to speed variations in the case of LTE. We will explain this when we talk about the result of WiFi interface. Meanwhile, we see in Fig. 4(a) that the SINR_based method has much larger PER. The average PER is around 0.19, while the average standard deviation of PER is around
0.52. Note that the results are achieved after multiplied by $\beta$ and $\gamma$, which has considered the efficiency of various network layers, and the difference resulted from the implementation in NS3. Therefore, we think the problem of the SINR-based method lies in the rough logarithm model cannot precisely fit into every distance range. It is hard to generate a single value that can fit all the distances. In deployment environment, it is also hard to figure out the best parameters for networks with various settings like different BS transmission power.

![Fig. 4. Comparison of average goodput PER with various speed.](image)

(a) LTE interface.  (b) WiFi interface.

The result for WiFi interface is shown in Fig. 4(b). The proposed method still has much lower PER compared with the SINR-based prediction model in all speed settings. The maximum average prediction error rate (PER) of the proposed method in various speed settings is only about 0.24. The standard deviation of the results for all users is around 0.063. However, we note that PER of the proposed method increases as the MN speed increases. This is different from LTE because WiFi AP has lower range, and the simulation is conducted in a smaller area. For the same speed, MN can easily get more signal degradation from WiFi AP than from LTE BS. Therefore, speed change has more impact the WiFi interface goodput prediction than that of the LTE interface.

Moreover, we observe that the standard deviation of the proposed method among users is fairly consistent under all the speed settings for both LTE and WiFi. In contrast, that of SINR-based method has significant changes as speed increases. This is understandable because SINR is highly related to location. At the same time, we see that the PER of SINR-based method is pretty high at the speed of 15m/s and 30m/s. This is because the coverage of the WiFi AP is usually limited (with the default power settings in NS3, one AP can cover around 80m). Higher speed usually means frequent out-of-range breaks or SINR changes during one scheduling period. The scheduler can only achieve link status from rough-grained SINR samples. Nonetheless, the goodput used in our proposed method can be seen as a continuous measurement of link status during discrete time intervals. The scheduler therefore achieves better prediction accuracy.

Since the proposed method is not that sensitive to location variation, we further study the prediction error changes of the SINR-based method of the specially controlled node (the 20th node we mentioned above). In this simulation, we generate PER without taking the absolute value operation, because the results are no longer averaged among users. We also did not time the $\beta$ value as we did in the averaged test, because we want the result to reveal the $\beta$ value as a function of distance to BS/AP ($d$). As we see in Fig. 5(a), the prediction of LTE goodput using SINR-based method always overshoots (always positive). In general, the prediction error increases as $d$ increases. The increase of PER at the leftmost point when $d = 0$ is possibly due to the return of a default strongest SINR value to avoid dividing 0. The increase is mostly like steps. The reasons might lies in the implementation of NS3 LTE module, which accounts for certain packets as received if the receiving signal is above certain threshold.

For WiFi interface, we can only generate the result of three speed settings, because 30m/s is too fast for a 80m WiFi AP coverage. It is interesting to see that the prediction error of WiFi decreases when it goes to the edge of the network coverage. This is because the SINR-based method always tends to overestimate the goodput. However, WiFi goodput decreased slower than what the model $1/d^2$ predicts. Therefore, the prediction error can reduce when $d$ increases. The values generated here is an important reference for our experiment in the next section. Meanwhile, similar to what we observed in the averaged result in Fig. 4, the speed variation has little impact on the LTE prediction error, but larger impact for that of WiFi.

### B. Performance evaluation of the proposed algorithm

Due to the nature of NS3 DCE, it is difficult to verify the performance of the designed framework of scale inside it. Therefore, we further verify the performance of the designed framework with larger number of MNs and extended time period, using the data achieved from Section IV-A inside NS3 without DCE enabled. We set 25 WiFi APs on a 5x5 grid with a cell length of 120m. The LTE BS is set on the center of the area. The MN connects to the nearest one when it is inside an overlapped area. 500 MNs are randomly distributed inside the larger square of 660m and do random walk of speed $v$. We use $v = 5$ in the current simulation. The total length of the simulation is 1500s, and the scheduler makes decision every 15s. Parameters are set to $\alpha_k = 1$, $n_{st} = 3$, $\alpha_{thresh} = 85\%$, $T_{diff} = 60\%$, and $N_w = 1$ in our simulation. Every MN runs one downlink streaming application for the whole time. The traffic rate changes every minute by multiplying a random number chosen in $[0.33, 3]$. We use the prediction method in Section III-D.1 for goodput prediction. For comparison, we implement two distributed association control methods Full_Association and Second_Random, and the centralized association control method in [15], denoted as SINR_Centralized. With Full_Association, MN will use all the subflows available to it. The Second_Random enables MN to associate with the secondary link randomly. For the
Throughput (Mbps) 
Time (s) 
SINR_Centralized Proposed Second_Random Full_Association

[0x0]Second_Random
[0x0]Association

More importantly, the average overall mobile/index.html is that our proposed method can result in much higher overall goodput gain.

It has in average 2Mbps higher overall goodput compared with Full_Association, and 64% timer achieved better results than Full_Association. More importantly, the average overall goodput of our proposed method is almost twice as that achieved by the SINR_Centralized method. This proves that the more accurate goodput prediction in the last simulation study really helps the scheduler to generate better association plan, and improve the overall system performance. The reason that the proposed method is lower at the beginning (as low as Full_Association) is that the algorithm initiates with a default association setting before enough historical data is accumulated. The default setting in our simulation is Full_Association.

In the simulation, the upper bound of the overhead is 15 bytes per $t_s$ for one MN (without considering the optimization to reduce SINR sending). Therefore, the ratio of overhead added by the designed framework to overall goodput is averaged to 0.0857%. We believe it is rather small considering the overall goodput gain.

V. FUTURE WORK

We plan to implement the designed framework and scheduling algorithm in a real system. We would like to test the performance of the scheduler with GENI WiMAX/LTE BS and GENI ORBIT testbed.

VI. CONCLUSION

In this paper, we have designed a framework of HetNet association controller based on scheduling information collected from MPTCP stack. The framework can at the same time serve as the subflow association controller of local MPTCP connections. We test the prediction accuracy of the proposed goodput estimation scheme compared with traditional SINR based goodput estimation. From simulation in NS3 DCE, our method achieves better prediction accuracy in various speed settings. A centralized exemplar algorithm is proposed considering system-wise optimization goals. With simulation in NS3, our proposed association scheme can produce almost twice overall goodput comparing with the SINR based method and other distributed schemes.

REFERENCES