A Network Layer Approach to Enable TCP over Multiple Interfaces

Kameswari Chebrolu, Bhaskaran Raman, Ramesh Rao

***Abstract*— The mobile Internet is set to become ubiquitous with the deployment of various wireless technologies. When heterogeneous wireless networks overlap in coverage, a mobile terminal can potentially use multiple wireless interfaces simultaneously. In this paper, we motivate the advantages of simultaneous use of multiple interfaces and present a network layer architecture that supports diverse multi-access services. Our main focus is on one such service provided by the architecture: Bandwidth Aggregation (BAG), specifically for TCP applications.**

While aggregating bandwidth across multiple interfaces can improve raw throughput, it introduces challenges in the form of packet reordering for TCP applications. When packets are reordered, TCP misinterprets the duplicate ACKS received as indicative of packet loss and invokes congestion control. This can significantly lower TCP throughput and counter any gains that can be had through bandwidth aggregation. To improve overall performance of TCP, we take a two-pronged approach: (1) We propose a scheduling algorithm that partitions traffic onto the different paths (corresponding to each interface) such that reordering is minimized. The algorithm estimates available bandwidth and thereby minimizes reordering by sending packet pairs on the path that introduces the least amount of delay. (2) A buffer management policy is introduced at the client to hide any residual reordering from TCP. We show through simulations that our network-layer approach can achieve good bandwidth aggregation under a variety of network conditions.

***Index Terms*— Network Architecture, Scheduling Algorithm, Simulation, TCP applications**

I. Introduction

With the growth of the mobile Internet, a variety of wireless technologies are being deployed for Internet access. Examples include GPRS, HDR, UMTS, Iridium, 802.11, Bluetooth, etc. Several research challenges [1]–[4] related to the use of a single wireless technology at the mobile client have been explored so far. Considerable research has also gone into enabling seamless vertical handoffs [5], content adaptation [6] when moving from one interface to another. However, most research in this domain has been confined to single interface use at any given time to meet the connectivity requirements of the mobile client.

When coverage areas of the different wireless technologies overlap, end users need not restrict themselves to a single interface. They can choose *multiple* interfaces to make use of all available resources on their interfaces. The feasibility of using multiple interfaces simultaneously helps solve some of the limitations of wireless media, and enables new and exciting services. There are several advantages with such a setting:

- Bandwidth Aggregation: Aggregating bandwidth offered by the multiple interfaces can help improve quality or support demanding applications that need high bandwidth.
- Mobility Support: Handoff related delay can be minimized by keeping an alternate communication path alive.
- Reliability: High levels of reliability guarantees can be achieved (for applications that require it) by duplicating/encoding some or all packets on the multiple paths.
- Resource Sharing: The idea of multiple-path use can also be extended to scenarios involving more than a single client host. For instance, in an ad-hoc network of nodes connected in a LAN (via say 802.11 or Bluetooth), a subset of nodes may have WAN connections. The multiple WAN bandwidth resources can be shared effectively across all the nodes to access the Internet.
- Data-Control Plane Separation: The WAN interfaces in an ad-hoc/sensor network can also be used for *out of band* control communication, to enable simplified and/or efficient distributed ad-hoc protocols such as routing.

An architecture that supports multiple communication paths is required to realize scenarios such as those listed above. In this paper, we begin by providing a general framework in the form of such an architecture. We focus our attention on one of the services provided by the architecture: **B**andwidth **Ag**gregation (BAG). BAG in essence tries to achieve the following - if we can obtain, say a bandwidth of 200kbps and 100kbps from two interfaces, can we aggregate bandwidth and obtain in total a bandwidth of 300kbps? Our focus is on the performance of TCP over such bandwidth aggregation. In concurrent work [7] we have considered bandwidth aggregation for real-time interactive applications with strict QoS requirements.

The architecture can be addressed at different layers of the protocol stack. Though many link layer solutions [8]–[10] exist, they are infeasible in our setup, as networks span different domains. While application and transport layer solutions as proposed in [11]–[14] can be used, they are cumbersome to implement as they involve many changes in the infrastructure – applications and/or server software have to be changed. Network layer solutions, on the other hand have the advantage of being totally transparent to applications and involve only minimal changes. A network layer solution is easy to deploy, and legacy applications in particular can benefit from this approach.

Our network layer architecture consists of an infrastructure proxy. A single proxy may provide services to a set of mobile clients, and multiple proxies may be provisioned for reliability and scalability. Some of the features of the network proxy are similar in spirit to that provided by Mobile IP [15]. The client acquires a fixed IP address from the proxy and uses it in establishing connections with the remote server. The proxy (like the Home-Agent in Mobile-IP) captures packets destined for the client. The proxy is aware of the multiple interfaces of the client, and tunnels the captured packets to the client using IP-in-IP encapsulation. Unlike Mobile IP, the proxy can manage multiple care-of-addresses and perform intelligent scheduling of (tunneled) packets across the corresponding multiple paths.

One of the services provided by the architecture is that of bandwidth aggregation (BAG) for TCP applications. While the use of multiple interfaces allows us to increase throughput, the varying characteristics of the different paths (corresponding to different interfaces) introduce problems in the form of packet reordering. Packet reordering can degrade TCP performance due to the following reasons:

- For every reordered packet, a TCP receiver generates a duplicate ACK (DUP-ACK). On receiving more than 3 DUP-ACKs, the TCP sender considers the packet lost and enters fast retransmit and resends the packet that was only delayed (on one of the interfaces) – this wastes scarce bandwidth.
- The TCP sender also assumes loss as indicative of network congestion and reduces its sending rate by cutting down the congestion window by half.
- Depending on the particular TCP implementation, reordering can also generate bursts of packets. If the TCP sender is not allowed to send packets in response to DUP-ACKs, when a new ACK covering new data arrives, it produces a burst[1].
- Reordering can also affect calculation of round-trip time (RTT) estimation and hence retransmission timeout (RTO) as for every packet that is needlessly retransmitted, the RTT sample is ambiguous and cannot be used.

[1]If the TCP implementation uses max-burst factor as outlined in [16], burst sizes can be reduced.

In this paper, we systematically address this challenge of reordering in the presence of multiple communication paths. We first demonstrate the degradation in performance that can be caused due to reordering. Based on this, we identify a set of criteria that will help improve the overall TCP performance. We then propose a scheduling algorithm - PET (Packet-Pair based Earliest-Delivery-Path-First algorithm for TCP applications) that partitions the traffic onto the multiple paths to minimize reordering while utilizing bandwidths of the interfaces effectively. PET minimizes reordering by estimating the delivery time of packets on each Internet path and scheduling packets on the path that delivers it the earliest. Since, fundamental to scheduling is an estimate of the available bandwidth on any path, it obtains this estimate by sending packets in pairs as far as possible and using their inter-arrival spacing for calculating the estimate.

Given the dynamic nature of Internet paths, some amount of reordering is inevitable. To get around this, we propose a client-side buffer management policy (BMP) that tries to hide any residual reordering from TCP so that unnecessary retransmissions are avoided. BMP buffers out of order packets at the network layer and passes them to TCP in order. It also attempts to detect losses and react to them in a timely fashion.

We study the performance of the proposed approaches through simulations under a variety of network conditions. PET in conjunction with BMP outperforms by a large margin naive schemes like weighted round robin (WRR) that don't attempt to minimize reordering. Also the performance of PET-BMP is close to an application layer bandwidth aggregation scheme MTCP, where multiple TCP connections are opened, one on each interface. Our network layer approach is effective in addressing the challenge of reordering, and is thus performance-effective in addition to being easily deployable.

The rest of the paper is organized as follows. In the next section (Section II), we describe our architecture. Section III presents our experimental design methodology. We identify a set of design criteria for improving performance of TCP in Section IV. Subsequently, in Section V the scheduling algorithm and buffer management policy are described. Simulation results are presented in Section VI, while in Section VII, we discuss some of our assumptions and their validity. We discuss related work in Section VIII and finally conclude in Section IX.

II. Architecture and Services

In this section, we briefly present the motivation behind a network-layer architecture, and the functional components of our architecture. We also describe one of the services provided by the architecture - BAG, which is the focus point of this paper. Additional details of the architecture can be found in [17].

A. Why a Network Layer Architecture?

The architecture can potentially be addressed at different layers of the protocol stack. Link layer solutions are infeasible in this setup, as the networks span different domains, and we may have heterogeneous wireless networks. An application-level solution is a possible design alternative, and works by making applications aware of the multiple interfaces. Application specific optimization is possible here and can lead to better efficiency. However, given the diversity of applications, this approach would mean modifying/rewriting the various applications while ensuring compatibility with existing infrastructure, making wide spread deployment a difficult job. Further, the applications need to keep track of the state of different interfaces. And when multiple applications share common client resources (interfaces), they also have to be designed carefully to avoid negative interaction. These factors can increase application complexity.

Transport layer solutions (e.g. [13], [14]) share some of the same features as application layer solutions. While they can be efficient, they still need all server software to be changed, and require cooperation during standardization to prevent negative interaction.

With IP emerging as a unifying standard for wireless networks, a network layer approach has the advantages of being transparent to applications and transport protocols. No changes are needed in server software, making wide spread deployment lot easier. Legacy applications, in particular, can benefit from this approach, as they have no other design alternative. Another advantage with this setting is a centralized approach (at the network proxy) to end user flow management that can potentially prevent any negative interaction.

While the network layer approach overcomes most limitations of the other approaches, there are efficiency concerns as it operates further down the stack. However, we believe that with careful design, most inefficiencies can be minimized (we demonstrate this for the case of TCP applications in this paper). Further, our design choice as such does not preclude further optimization at the higher layers. In fact, our architecture can enhance higher layer approaches in terms of mobility support. In the absence of this solution, higher layer approaches may have to handle mobility themselves or rely on multiple Mobile IP initiations to handle the multiple interfaces (which to our knowledge is not supported by Mobile IP). We now briefly describe the main details of our architecture.

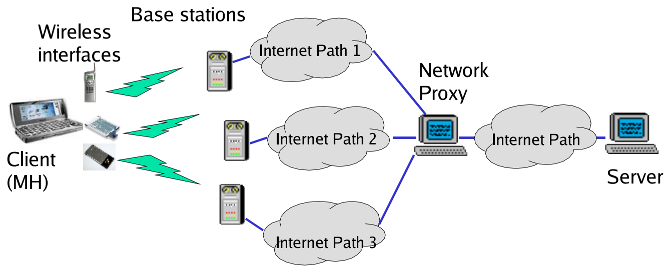

Fig. 1. Architecture to Support Multiple Communication Paths

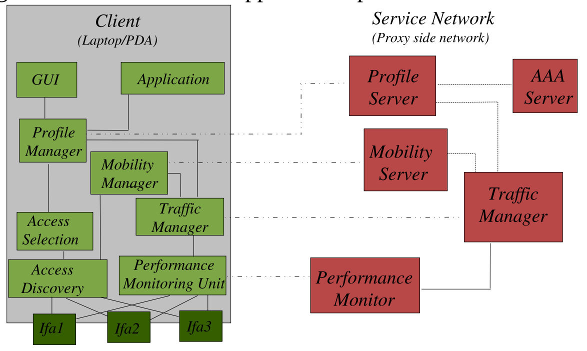

Fig. 2. Functional Components of the Architecture

B. Architecture

Fig 1 shows a high level overview of the architecture. The network proxy can provide many different services (Bandwidth Aggregation, mobility support, resource sharing etc) to the client (the MH). The MH is connected to the Internet via multiple network interfaces, with each interface having a care-of IP address. In addition, the MH acquires a fixed IP address from the network proxy and uses it to establish connections with the remote server. The MH also registers its multiple care-of IP addresses with the proxy. When the application traffic of the MH passes through the domain of the proxy (i.e., through the fixed IP address acquired from the proxy), the proxy intercepts the packets and performs necessary application or transport-specific processing. It then tunnels them using IP-in-IP encapsulation to the client's different interfaces. This mechanism is similar to that used in Mobile IP [15] but has been extended to handle multiple interfaces. Note that this mechanism is needed in our architecture even when the client is stationary – for simultaneous use of interfaces, not just for mobility support.

The functional components that make up our architecture reside on the MH and on the network proxy, and are as shown in Fig. 2. The *Profile Manager* generates a profile based on user input and application needs for each application. The profile specifies how to handle the application flow – the interfaces to use, the granularity of sharing (per packet or per session) while scheduling, any additional functionality needed (reliability, content adaptation etc). Based on this profile generated, the MH activates the necessary interfaces (if not already up) using the *Access Selection* component in conjunction with *Access Discovery*. The profile information is also conveyed to the *Profile Server* at the proxy. The *Mobility Manager* registers the care-of IP addresses with the *Mobility Server* on the network proxy. The *Traffic Manager* performs the necessary processing and scheduling of traffic onto the multiple interfaces based on the profile information, as well as input from the *Performance Monitoring Unit (PMU)*. The PMUs on both ends monitor various characteristics such as the throughput/delay of the path from the proxy to the different interfaces, power consumption at the MH, etc. They also communicate with each other periodically to keep this information up to date.

BAG services: One of the services provided by the architecture towards increasing application throughput is that of Bandwidth Aggregation (BAG). While peak data rates in wireless networks have shown an increasing trend: 9.6kbps (GSM-TDMA) in 2G to 2Mbps(UMTS) in 3G, the typical rates a user can expect to see in a loaded network are still very small [18] - 40kbps in 1xRTT, 80kbps in EDGE, 250kbps in UMTS. Even 802.11 interfaces that can provide speeds of upto 11Mbps, often are constrained in bandwidth since most hot spots these days connect to the Internet via "broadband" (DSL/Cable) which constitutes a bottleneck under load. Supporting real-time applications with stringent QoS requirements, large file transfers, intense web sessions is a difficult task and may not even be possible if confined to a single interface. Using bandwidth available from all possible sources increases one's bandwidth, and may be the only option to support demanding applications.

In this paper, we focus our attention on BAG services for TCP applications. In the context of the overall architecture presented above, a crucial aspect that dictates TCP performance is the scheduling algorithm (PET) residing on the TM at the proxy. This algorithm PET, splits the traffic onto the different paths with the objective of minimizing reordering. The client-side TM has a buffer management policy (BMP) that processes the incoming data before passing it on to the TCP layer. BMP tries to hide from TCP any residual reordering that happens. The above explanation corresponds to down-link traffic. The same holds for up-link traffic, with the roles of PET and BMP reversed at proxy and MH. We discuss the design of BAG and BMP in Section V. Prior to that, we present our experimental design methodology to help understand the design and the results presented in future sections.

III. Experimental Methodology

Our design and evaluation are based on experimental simulations since this allows us to quickly explore a wide range of possibilities and design choices in a controlled manner. We use the ns-2 network simulator [19] (version 2.1b9a) for our simulations.

We use the generic network topology captured in Fig. 1. In our experiments, the main TCP flow is an FTP transfer from the server to the MH. In our studies, we consider a wide variety of scenarios to understand the performance of PET-BMP. We consider both: (a) the presence of cross traffic and losses at the BS, and (b) their absence. While the first is a more realistic setting, the second helps us understand behavior of PET-BMP in response to each parameter better.

For the cross-traffic, we consider a mix of both FTP and web flows that compete with the main flow for the BS's link capacity. Losses are introduced via - 1) congestion at the BSs, where each BS has a maximum queue size and implements a drop-tail queuing policy and 2) channel errors, where the BSs introduce uniformly distributed errors in the packets.

We use Weighted Fair Queuing (WFQ) [20] for packet scheduling at the base stations where all flows through the base station are given the same weight. This permits equal sharing of the scarce wireless link capacity among all the flows. Our WFQ implementation uses a single buffer for storing packets from all the flows.

A. *Parameter Settings*

The details of the various parameter settings of our experiments are as follows. We consider either 2 or 3 wireless interfaces (communication paths). We do not consider more than 3 interfaces since such a scenario is unlikely in practice.

The main FTP/TCP flow lasts for 60 seconds, which is the duration of the experiment. The server uses the New-Reno variant of TCP, where the maximum congestion window size is set to 50 packets. The packet size used is 1500 bytes. We also use a max-burst factor, which limits to four the number of packets that can be sent in response to a single ACK. Without this, New-Reno could send a large burst of packets upon exiting Fast Recovery [16]. The TCP sink at the MH does not use delayed Acks.

The number of cross-traffic FTP and web clients considered for the different interfaces vary depending on the experiment. The size of the cross traffic FTP transfers are uniformly distributed between 200 and 2000 kbytes and their start-times are uniformly distributed between 0 and 60 seconds respectively – the total duration of the experiments. The web clients run for the entire 60 sec of the simulation. The details of the CDFs for think/reply/size used in web clients can be found in [21].

We consider a range of values for the link capacities of the various interfaces. For experiments without any cross traffic, we experimented with 3 interfaces with link capacities of 50kbps, 100kbps and 200kbps. These values reflect the bandwidths one can expect to see on WWANs when the wireless channel is dedicated for single use to the MH. In the presence of cross traffic, we increase the link capacities

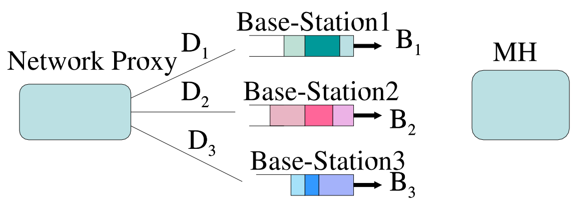

Fig. 3. A Simplified View of the Network between Proxy and MH

of all interfaces to 1000kbps. Note that even in this case, since we consider different cross traffic patterns at the BSs, the average throughput available on the interfaces can be quite asymmetric.

The server and the proxy are connected by a 10Mbps link with a one-way delay of 15ms. The proxy and Base Stations (BS) are connected by 10 Mbps links with one-way delay of 50ms on each. In next generation networks, the BSs are considered to be an extension of the Internet. Accordingly, we set the one way delay from proxy to BSs values typical of present day Internet paths. The results are not particularly sensitive to the exact value of the one-way delay. The bandwidth value of 10 Mbps ensures that the wireless interfaces are the bottleneck.

B. Algorithms Under Comparison

For comparison purposes, we consider three *ideal* systems which place a limit on the best that can be achieved by a network layer approach to bandwidth aggregation. One is an application-layer solution, *MTCP*, where we open multiple TCP connections one on each interface and sum the throughputs achieved on the individual interfaces. The other point of comparison is Aggregated Single-Interface TCP *(ASI)*, where we replace the multiple interfaces with a single interface of the aggregate capacity. The third is a system that employs an idealized scheduling policy Earliest Delivery Path First (EDPF) [7], [22] at the proxy. A brief overview of the algorithm is as below. Further details of this algorithm can be found in [7], [22]. (In [7], this algorithm was used to achieve bandwidth aggregation for video applications that run on UDP).

The overall idea behind EDPF is to (1) take into consideration the overall path characteristics between the proxy and the MH – delay, as well as the wireless bandwidth, and (2) schedule packets on the path which will deliver the packet at the earliest to the MH. The network between the proxy and the MH can be simplified as shown in Fig. 3. Each path l (between the proxy and the MH) can be associated with three quantities: (1) D_l, the one-way wireline delay associated with the path (between the proxy and Base Station - BS), (2) B_l, the bandwidth available at the BS, and (3) a variable A_l, which is the time the wireless channel becomes available for the next transmission at the BS. If we denote by a_i, the arrival instance of the i^{th} packet (at the proxy) and by L_i, the size of the packet, this packet when scheduled on path l would arrive at the MH at d_i^l.

$$d_i^l = MAX(a_i + D_l, A_l) + L_i/B_l \qquad (1)$$

The first component computes the time at which transmission can begin at the BS, and the second component computes the packet transmission time. EDPF schedules the packet on the path p where, $p = \{l : d_i^l \leq d_i^m, 1 \leq m \leq N\}$, N being the number of interfaces. That is, the path with the earliest delivery time. EDPF then updates A_p to d_i^p. EDPF tracks the queues at each of the base-stations through the A_l variable. By tracking the queues at the base-stations and taking it into account while scheduling packets, EDPF ensures that it uses all the available path bandwidths, while achieving minimal packet reordering. EDPF has the property that when packets are of the same size, it eliminates reordering fully.

We note that comparison with ASI is meaningful only in the no-cross-traffic case. This is because, if we introduce cross traffic in ASI by summing up the cross traffic at each individual BS, the throughput of the main TCP flow goes down considerably. This is in turn because, the available bandwidth in ASI now gets distributed equally among all the flows. On the contrary, when using multiple interfaces the available bandwidth at a BS gets distributed only among the flows served by it. Also, note that MTCP is in general more aggressive than any single end-to-end TCP connection since it uses multiple congestion windows.

IV. Design Criteria

To motivate the design of the Earliest Estimated Delivery Path First (PET) and the Buffer Management Policy (BMP), we now present some preliminary results and derive a set of design criteria from them. We first state the criterion, and subsequently explain the reasoning behind it, presenting simulation results as necessary.

Criterion 1: Utilize bandwidth of all interfaces

Our objective is to achieve the maximum possible throughput from the server to the MH using TCP over an underlying heterogeneous network, without any modifications to TCP. The maximum throughput is achieved only if we utilize the bandwidth of all the interfaces – hence this criterion.

Criterion 2: Minimize reordering

Let us now look at what happens when one uses all the interfaces. A simple scheduling policy that can be implemented at the proxy is the Weighted Round Robin (WRR), where the number of packets sent on a path (corresponding to an interface) is proportional to the link capacity of the interface. Table I shows the performance of WRR in comparison with MTCP, ASI and EDPF (focus on the first four rows). As can be seen from the table, the throughput

Algorithm	Thr(kbps)	DupAcks	Retrx
MTCP	339.6	0	0
ASI	339.6	0	0
EDPF	339	0	0
WRR	210.6	533	128
WRR-BUFF	338.0	0	0
PET-BMP	336.8	0	0

TABLE I

IDEAL SITUATION - NO CROSS TRAFFIC, NO LOSSES

achieved by WRR is much lower than MTCP, ASI or EDPF. This is due to several unnecessary retransmissions. Whenever packets are reordered, the TCP sink generates DUP-ACKs. On receipt of more than 3 DUP-ACKs for a packet, the TCP sender considers the packet lost and invokes congestion control by reducing the sending rate (halving the congestion window). On the other hand, as can be seen, EDPF has performed as well as ASI and MTCP. Now, EDPF is an idealized scheduling policy that has perfect knowledge of the system parameters, and is thereby able to *eliminate* reordering altogether. In reality however, one can only estimate these parameters and schedule accordingly. So, eliminating reordering totally may not be feasible, so the best one can do is to *minimize* reordering.

Criterion 3: Hide reordering from TCP

Since reordering is inevitable in practice and can have quite a negative impact on TCP, let us see if its possible to overcome its effects. The main problem with reordering is the generation of DUP-ACKs. Since we do not wish to make any changes to TCP, we can prevent the generation of DUP-ACKs by buffering packets at the client (at the network layer) and passing them in order to TCP. So, in the previous example, suppose we employ such a simple buffering mechanism at the client, the performance of WRR can be significantly improved. As can be seen from Table I, WRR-BUFF (WRR with client buffering) performs similar to EDPF, MTCP, and ASI. Hence this third design criterion: "hide reordering from TCP".

Note that this does not mean that we can relax the second criterion of minimizing reordering assuming that its effects can be masked by buffering. In the previous example experimental setup, the amount of reordering was small. Hence buffering helped in overcoming reordering.

Simply buffering may not help if the amount of reordering is large. To see this, suppose we increase the delay on the third interface (the one with capacity 50kbps) from 50ms to a much higher value of 1s, there is a lot more reordering in WRR. This is because WRR considers only link capacities and not path delays while scheduling. In this setup, EDPF achieves 337.4 kbps, WRR 95.6 kbps, and WRR-BUFF only 70.4 kbps. EDPF achieves good bandwidth aggregation as the scheduling ensures that there is no reordering (unlike WRR, EDPF considers path delays in addition to link capacities). On the other hand, WRR and WRR-BUFF have much lower throughput, even lower than what we could have achieved had we not performed bandwidth aggregation but just used the highest bandwidth interface (200kbps). Further, the performance of WRR with buffering is even worse than without buffering. This is because there are as many as 40 retransmission timeouts in case of WRR-BUFF. Since no DUP-ACKs reached the sender to trigger fast retransmission (due to client buffering), this forced the sender to enter slow start each time.

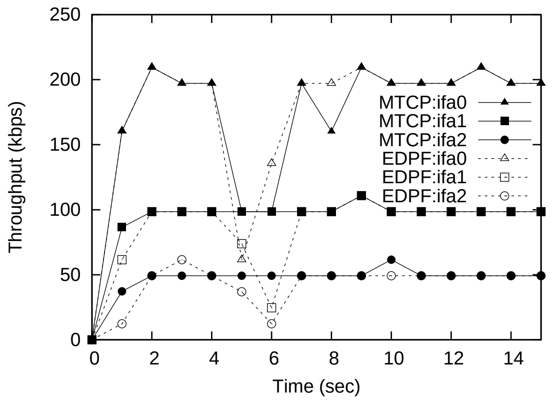

Fig. 4. Isolation of Losses

Criterion 4: Detect packet losses and react to them in a timely fashion

So far we have not considered losses. The simple buffering policy above works because of this. However in the presence of losses, we could potentially wait indefinitely until the next expected sequence number comes while storing out-of order packets. This would eventually trigger a retransmission timeout at the TCP sender for each packet loss. When losses are present, we need to react to them in a timely fashion, or otherwise risk retransmission timeouts which lower throughput significantly. So in the presence of buffering it is important to detect losses and react to them in a timely fashion.

Criterion 5: Avoid Burstiness of Traffic

Another problem with buffering out of order packets is that sending them all at once to the TCP receiver will generate a burst of ACKs and they in turn generate a burst of packets at the TCP sender. In general bursty traffic is not a good feature and is better avoided as it increases queuing delay, introduces more losses and lowers throughput [23].

Criterion 6: Isolate losses

Since we are using multiple interfaces, the different paths can have different loss rates. Since TCP reacts to losses by reducing the sending rate, it is important to ensure that losses on one path don't affect the achievable throughput on the other paths. We term this as *loss isolation*. Note that MTCP achieves such isolation naturally.

Figure 4 shows the instantaneous throughput (averaged over 1 second intervals) of EDPF and MTCP when losses (congestion based) are introduced at the BS with 200kbps capacity by setting the maximum queue size to 30kbytes. As can be seen in the figure, in case of MTCP, the losses on one interface have not affected other interfaces but in case of EDPF, losses on one interface (ifa-0) have lowered the throughput on other interfaces (ifa-1 and ifa-2). This is not desirable. So, the final design criterion is to isolate losses.

Note that criteria 1, 2 and 6 dictate the design of the scheduling algorithm at the proxy while criteria 3, 4 and 5 dictate the design of buffer management at the client.

V. Scheduling and Buffer Management

We now present our design of a network layer solution to bandwidth aggregation for TCP applications based on the design criteria described in the previous section. There are two main parts of our solution: Packet-Pair based Earliest-Delivery-Path-First scheduling algorithm for TCP applications (PET) at the proxy, and the Buffer Management Policy (BMP) at the MH. We look at each in turn.

A. Packet Pair based Earliest-Delivery-Path-First Scheduling Algorithm for TCP applications (PET)

Consider the EDPF scheduling discipline. It is able to achieve good utilization of bandwidth on all interfaces while minimizing reordering (criteria 1 and 2) because it has perfect knowledge of the system parameters. In reality, we can only *estimate* these parameters. So, if the design of PET is based on the same concept of EDPF but with perfect knowledge replaced by estimates, we can hope to meet criteria 1 and 2 to some extent. The parameters of concern for PET scheduling at the proxy are: (1) the wireline delay on each of the communication paths from the proxy to the BSs, and (2) the available bandwidth on the wireless link. The available bandwidth in turn translates to the transmission and queuing delay at the BSs.

In the next generation Radio Access Networks, Base Stations (BSs) are considered to be an extension of the IP based Internet. Accordingly, we consider the delay experienced by the packets up to the BS, similar in nature to Internet path delays. This *wireline* delay can be estimated by sending signaling packets to the MH during connection setup (clock synchronization is not required since only the relative delay between the different paths matters). This in general suffices because Internet path delays are known to vary only slowly, over several tens of minutes [24]. Further, any errors in estimation are usually small (as average delays on the backbone are themselves small), and will likely be masked by the transmission and queuing delay at the bottleneck bandwidth [2].

The second parameter, the available bandwidth, is dependent on the amount of cross traffic, fluctuating channel conditions etc. These can definitely change in the middle of a connection. Hence the available bandwidth, unlike delay, needs to be estimated and updated continually throughout the duration of the connection. Our overall approach for estimating this available bandwidth is based on the packet-pair technique [25]. The packet-pair technique estimates the bottleneck capacity of a path from the dispersion (spacing) experienced by two packets which were sent back-to-back. Since the wireless link is often the bottleneck in the network path and since we assume that the BSs implement WFQ, bandwidth estimation based on this technique is feasible.

Using separate signaling packets to probe bandwidth continuously is excess overhead. Further, the probing packets will compete with the main flow for available bandwidth. Hence we rely on the incoming TCP packets themselves for bandwidth estimation by treating every incoming TCP packet as part of a pair and sending packets in pairs on any path.

Since PET at the proxy needs the inter-arrival time between packet pairs to update its bandwidth estimate, we introduce an additional mechanism in the form of a feedback loop between the MH and the proxy to get this information. We achieve this by means of *Signaling-ACKs (SIG-ACKs)* sent from the MH to the proxy for each TCP packet received from the sender (via the proxy). Note that these are different from the regular ACK stream, which may not even pass through the proxy. The MH reports the packet arrival times in the SIG-ACKs (again, clock synchronization not necessary since the proxy only needs the inter-arrival times).

Once PET has the delay and bandwidth estimates, it can use EDPF with idealized delay and bandwidth values replaced by the estimates. In essence, the working of PET is as follows. PET treats every incoming packet as part of a pair. To begin with, PET has no bandwidth estimates to perform scheduling. So, there is an *initial phase* where it sends packet-pairs on the various paths in a round-robin fashion, until it gets a bandwidth estimate of the bottleneck in the path through SIG-ACKs. Then, it uses these bandwidth estimates to perform EDPF based scheduling to determine the path (interface) on which to send the first packet of a pair. The second packet of a pair is always sent on the same path as the first packet. Retransmitted packets

[2]Present day wireless technologies such as GPRS, 1xRTT show a high degree of delay variation. These systems are very young and the delay variation is likely due to initial setup problems. Moreover, we believe that the variation is caused on the wireless hop (due to release grant/retransmissions as captured by available bandwidth parameter), than on the path between proxy and BS (as captured by wireline delay parameter).

are not part of any pair as the bandwidth estimate can be ambiguous. As PET clocks out more packets, it gets fresher bandwidth estimates, which it uses to schedule incoming packets with the goal of minimizing reordering.

Some additional details on how the PET scheduling mechanism works are as follows.

- While TCP is in slow start, every TCP ACK generates two packets that arrive back-to-back at the proxy, which helps bandwidth estimation. But once in congestion avoidance phase, packets may not arrive back-to-back at the proxy. However, these packets can still form a pair for bandwidth estimation since during this phase, normally the TCP pipe is not empty [26] and thereby both packets will be buffered at BS (before the bottleneck wireless link) and still give a valid estimate.
- It is possible for bandwidth estimates to be incorrect due to transient changes in cross traffic, or during multiple losses per congestion window where the TCP pipe gets cleared. In this case, there will be more reordering which is normally masked by the BMP at the MH. As new samples arrive, the history clears and the estimate converges to the correct value[3].
- As long as there is a backlog, PET/EDPF ensure that bandwidths on all interfaces are utilized effectively. However there is a danger of getting stuck to a single interface – this can happen when the available bandwidth of one interface is estimated to be much higher than another. If losses at this stage slow down the TCP sending rate, to avoid reordering, we may never end up using the low bandwidth interface. This prevents us from getting any future bandwidth estimate updates on it. In the future even if more bandwidth is available on it, we may never use it. To alleviate this, it is important to send TCP packet-pairs on an interface periodically (even if PET chooses another interface) to estimate its bandwidth.

PET thus attempts to achieve design criteria 1 and 2. Design criterion 6, isolation of losses, as we argue now, is not always possible to achieve in a purely network layer solution. This depends heavily on the loss pattern. The reason for this is as follows. When a single loss occurs, if W is the window size just before the loss detection at the TCP sender, the sender does not send any packets for the first W/2 DUP-ACKs [26]. Normally, this should not clear the TCP pipe (backlog) on all the interfaces, and when TCP resumes after fast-retransmit, the pipes slowly fill up. In this case, the losses on one interface do not affect the others. However, if many packets are lost within a window, by the time W/2 DUP-ACKs arrive, some of the pipes would have cleared. The scheduler at the proxy cannot help in this situation by clever scheduling of packets because of the way TCP reacts to losses.

[3]In our bandwidth estimate update mechanism, we use a large weight (0.8) for the current estimate, and a corresponding small weight (0.2) for the history as its important to react rapidly to current conditions, thereby minimizing reordering.

B. *Buffer Management Policy (BMP)*

Due to the use of packet-pairs, and also due to errors in bandwidth estimation, PET scheduling would result in some amount of reordering. In accordance with design criterion 3, we use a client-side buffer to hide this reordering. The main challenge in the design of the Buffer Management Policy (BMP) is the detection of losses when they happen (design criterion 4). We discuss this now.

Since we buffer packets, it is important to know if a packet is lost or merely reordered. A mechanism to do this is as follows. Suppose we are expecting (an in-order) sequence number N. We start a timer associated with it – when the timer expires we conclude that N could not have been reordered, and hence was lost. We then send the buffered packets to TCP so that DUP-ACKs can be sent and fast-retransmit triggered. We call this *timer-based* loss detection.

Timer-based loss detection requires adaptation of the timer value, which can potentially be done based on the amount of reordering seen. However, a simpler mechanism to detect losses exists if we assume that packets always arrive in order on an interface (which is usually the case). Suppose we receive sequence numbers greater than N on all of the interfaces, we can conclude that N is lost. We call this *comparison-based* loss detection. Even if this mechanism is used, a timer based mechanism cannot be dispensed with totally. This is because, if an interface (say ifa-2) is not used for a long time due to low bandwidth, we could wait indefinitely to conclude that N was lost (for a comparison-based loss detection, some sequence number above N must be received on ifa-2 as well, to conclude loss). This would eventually trigger a TCP timeout, which is undesirable. Similar problems would arise if a loss happens towards the end of a connection, when there are no more new packets to be sent on all the interfaces. Hence, a timer-based mechanism is required, but can act as a backup for comparison-based loss detection. In such a case, since the timer kicks in only rarely, its value is not so crucial, and can be set at a conservative value. (In our experiments, we set it to 0.5sec.).

Design criterion 5 (avoid burstiness) can be achieved in two possible ways. One is to separate the generated ACKs by an interval at the client-side network layer, before sending them out on to the network (ACK pacing [27]). The same effect can also be achieved by separating packets by an interval when sending them to the TCP layer from the client-side buffer. We implemented ACK pacing in our experimental setup.

So in essence, PET attempts to satisfy criteria 1 and 2 by sending packets in pairs to obtain bandwidth estimates which it uses in turn to schedule packets to minimizing reordering. Criterion 6, isolation of losses is difficult to achieve using PET because of default TCP response to losses. BMP on the other hand buffers out-of-order packets and sends them in order to hide the effects of reordering on TCP (criterion 3). It also attempts to react to losses in a timely fashion based on comparison and timer based loss detection (criterion 4). ACK pacing [27] can be used to avoid burstiness (criterion 5).

VI. Experimental Results

The above design of PET-BMP needs detailed evaluation. This section presents the results of our experiments to demonstrate the effectiveness of PET with BMP in achieving our design criteria.

Let us first look at how well PET-BMP performs in the ideal setup described in Section IV, with no cross traffic and no losses. Table I compares the performance of PET-BMP with other algorithms. PET-BMP achieves a throughput of 336.8 kbps (last row in Table I) – very close to that of EDPF. The slight decrease in throughput is mainly due to two reasons: (1) The initial phase, where until an estimate is available, it sends packets in a round-robin fashion. (2) The use of packet pairs which introduce some small amount of reordering.

Now let us relax the idealistic assumptions in the experimental setting and introduce cross traffic and losses. We first look at each effect separately (Sec. VI-A and Sec. VI-B). Later we consider both cross traffic and losses in the same experiment (Sec. VI-C).

A. Cross Traffic and No Losses

In this experiment, we introduce cross traffic at the BSs and ensure that no losses happen by giving adequate queue sizes at the base station. Note that the link capacities here are 1000kbps on each interface. The number of flows that constitute cross traffic during the course of the simulation is 3 ftp and 16 web flows at BS0, 5 ftp and 24 web flows at BS1 and 6ftp and 20 web flows at BS2. These number of flows for the cross-traffic are merely to illustrate the behaviour – we consider various other settings in Sec. VI-C.

Figure 5 shows the variation in the instantaneous TCP throughput. We compare WRR and PET scheduling, both with BMP implemented at the client. We compare these with the MTCP application-level solution for bandwidth aggregation. We see that PET-BMP follows MTCP very closely, whereas WRR-BMP lags behind by a big margin.

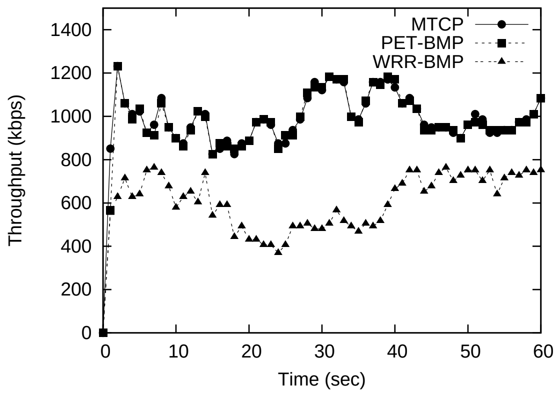

Fig. 5. Cross Traffi c, No Losses

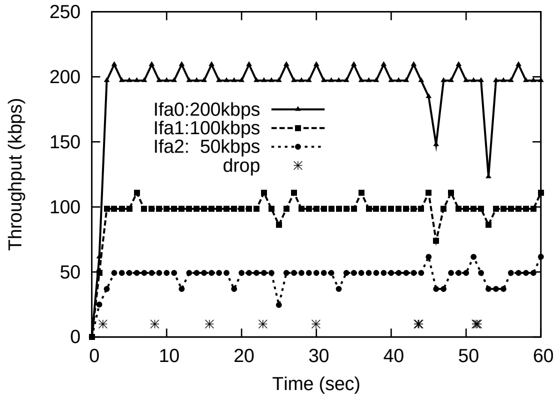

Fig. 6. No Cross Traffi c, Losses

The average throughput obtained by the main TCP flow in comparison to MTCP, PET-BMP, and WRR-BMP are 967.6, 960, and 589 kbps respectively. This illustrates that PET-BMP is able to meet the goal of effective bandwidth aggregation in this setting.

Let us now consider losses but no cross traffic.

B. No Cross Traffic and Losses

As mentioned in Sec. III, we now use 50, 100, and 200 kbps for the link capacities. Instead of introducing random drops, we control the packets that were dropped, so as to explain the behavior of PET-BMP better. We drop a total of 10 packets (this suffices to illustrate the comparative behaviour of PET-BMP). We ensure that there is only one drop per congestion window for the first 5 packets dropped. Later we drop 2 packets per congestion window and then 3.

The throughput achieved by PET-BMP in this case was 330.2 kbps, a decrease of 6.6 kbps from the no loss case (refer to Table I). The number of retransmissions were 15 - five more than what was needed to recover from losses. Note that such unnecessary retransmissions in case of PET-BMP happen only in response to losses unlike in WRR, where they happen on a regular basis due to reordering. Comparing with ASI, the throughput achieved by ASI for same drop pattern was 338.2, 1.4kbps lesser than the no loss case. For ASI, the number of retransmissions were 10, equal to the number of dropped packets.

The reason for more retransmissions in case of PET-BMP is the following. When a packet is detected lost by the TCP sender, on receipt of 3 duplicate ACKs, it retransmits the packet and enters fast recovery. It is possible that the retransmitted packet may arrive before other outstanding packets when fast recovery was entered. In this case, when the ACK generated by the retransmitted packet arrives, the TCP sender considers the packet immediately following the acknowledged packet as lost and retransmits it. However as we can see, the drop in throughput is small which shows that PET-BMP is able to react to the losses and recover from them in a timely fashion. If we had depended on a timeout in the BMP to react to the losses, the decrease in throughput would have been much higher.

The drop in throughput of PET-BMP in comparison with ASI is due to an important factor - lack of adequate loss isolation. Fig 6 shows the throughput achieved by PET-BMP on the 3 interfaces (ifa-0, ifa-1 and ifa-2) along with the time of dropped packets. As we can see in the figure, the first drop does not affect any interfaces. The next two drops affect ifa-2 but not the other two. The 4th drop affects ifa-2 and ifa-1, but not ifa-0. However, when more than one drop happen per congestion window (the drops after 50 sec), all interfaces suffer. This demonstrates that isolation of losses may be possible with PET-BMP when losses are spread out, but is difficult to achieve when losses are clustered.

C. Cross Traffic and Losses

We finally perform an experiment where we consider both cross traffic as well as losses. For this experiment, we randomly generate 10 different cross traffic scenarios. For each scenario, we randomly choose a value between 2 and 8 for the number of FTP flows and a value between 16 and 32 for the number of web flows at a BS (normally, due to randomness, cross traffic profiles on the interfaces are different, introducing asymmetry). This range of possible cross-traffic covers a range of scenarios of the available bandwidth for the main flow on each of the wireless bottleneck links. Note that not all flows are simultaneously active at any given time.

In each "scenario", the start times and file sizes for the cross traffic varies depending on a random number "seed". So, for each cross traffic scenario, we conduct 10 different runs with different seeds and average the throughput seen by the main TCP flow across the seeds. This run across different seeds ensures averaging across various start/finish times of the cross-traffic.

We consider two types of losses - congestion and channel errors. We present results when considering just congestion losses and also when channel losses are introduced on top

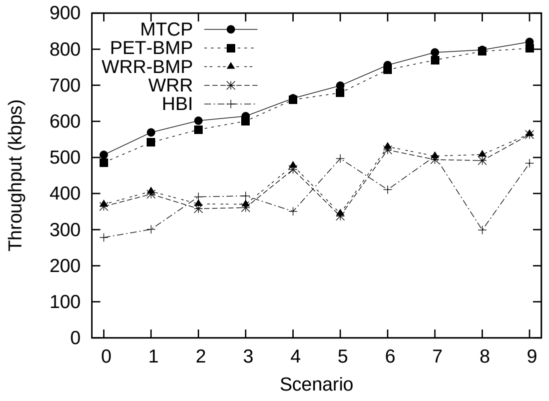

Fig. 7. Congestion Losses, # Interfaces = 2

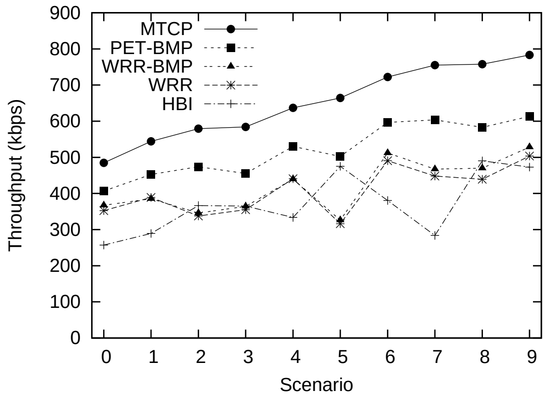

Fig. 8. Congestion Losses and Channel Losses, # Interfaces = 2

of congestion losses. For the second case, we use the same traffic pattern as was used for the case of congestion losses. For introducing congestion losses, we set the maximum queue size at the BSs to 200 kbytes. The distribution used for introducing channel losses is uniform, with a loss rate of 1%.

We first present results for the case of 2 interfaces and then increase the number of interfaces to 3 to show the effect of increased reordering (more interfaces, more scope for reordering) on the performance of PET-BMP.

Fig. 7 compares the throughput achieved by the different algorithms when considering just congestion losses for the case of 2 interfaces. Fig. 8 shows the throughput when channel losses are introduced on top of congestion losses. In these figures, we also show the throughput of MTCP as seen on just the highest bandwidth interface (HBI)– this is what would have been TCP's throughput had we done no bandwidth aggregation and simply used just the highest bandwidth interface. For ease of visualization, we sort the 10 scenarios in the order of increasing bandwidth achieved by MTCP.

When considering congestion losses alone, we find that PET-BMP performs very closely with MTCP (the difference ranges between 4-27 kbps). This is true across the wide range of cross-traffic scenarios we have considered. In contrast, WRR lags behind PET-BMP and MTCP consid-

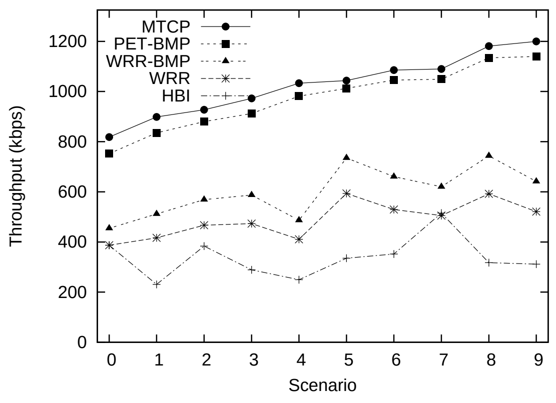

Fig. 9. Cross Traffi c, No Losses

erably. The use of BMP alone with WRR brings in some benefits, but not a whole lot. Compared to the case of using just the Highest Bandwidth Interface, PET-BMP clearly illustrates the advantages of bandwidth aggregation. WRR-BMP performs better than HBI in some cases, while in others it performs worse. This shows that if care is not taken when scheduling to minimize reordering, effects of bandwidth aggregation could be nullified.

When channel losses are considered in addition to congestion losses, PET-BMP performs better than WRR-BMP, and can still bring in considerable benefits over using just the highest bandwidth interface (HBI). However, it falls behind MTCP by a larger margin (78-174 kbps) than with just congestion losses. This is mainly due to the aggressive behavior of MTCP during losses (multiple congestion windows) and also due to the inability of PET to achieve loss isolation.

Though PET-BMP performs sub-optimally in the presence of channel errors, we argue that this is not much of a problem due to the following reason. The scheduling algorithm PET can achieve loss isolation (similar throughputs as MTCP) as long as loss rates are low and losses are more spread out. In the above experiments, while congestion losses were under 0.7%, additional channel losses increased this percentage to 1.4%. This degradation in throughput of PET-BMP compared to MTCP is mainly due to this increase in loss rate. It does not matter to PET (or to TCP) whether losses are due to congestion or channel errors as long as these rates are not too high. Ideally TCP should react to only congestion losses not channel errors. Considerable research [1] has gone into undoing the effect channel errors have on the congestion window of the TCP sender. With some of these mechanisms in place (e.g. through use of ELN/ECN bit), we can expect PET-TCP to perform similar to MTCP, as was seen in the no-channel-loss case in Fig. 7. As far as congestion losses are concerned, the loss rates should not reach high values because of TCP default behavior of cutting down its sending rate in response to congestion.

Fig. 9 shows the performance of the different algorithms for the case of 3 interfaces with just congestion losses. As the number of interfaces increase, so does the scope for reordering. As can be seen, the difference between PET-BMP and MTCP is wider than in the case of 2 interfaces. The difference ranges between 30-60kbps now.

D. *Summary of Results*

The above experimental results indicate that PET-BMP can bring in significant benefits through bandwidth aggregation over using just a single interface. It performs as well as the application level MTCP solution and outperforms by a large margin approaches based on using Weighted-Round-Robin in most scenarios. It achieves this through meeting the design criteria in Sec. IV – all except “isolation of losses”. While, it can still perform close to MTCP, when the loss rates are low, higher loss rates degrade its performance due to its inability to perform “loss isolation” and due to the inherent aggressiveness of MTCP. However, when wireless losses are minimized using other mechanisms (e.g. [1]), the performance of PET-BMP can become comparable to MTCP.

For the range of scenarios we have considered, the estimation techniques used by PET are effective, and we have not found the need for any parameter tuning – PET is simple enough to be robust in this regard. Given the ease of deployment of PET-BMP and the performance gains for effective bandwidth aggregation, we believe that it has wide applicability.

VII. Discussion

In this section, we elaborate on the validity of some of our assumptions, and other issues with our network layer approach. In parallel, we also point out some design alternatives worth further study.

A. *Validity of Assumptions*

Two assumptions that feature in the above work are (a) WFQ implementation at the BSs, and (b) proper estimation of delay on the paths from proxy to BSs.

Unlike First-Come-First-Serve implementation, WFQ implementation or other variants of it, divide the link capacity equally among all the flows and thereby help bandwidth estimation techniques in getting a reliable estimate. Though the scheduling policy employed at a BS is not in our control, we believe that WFQ is a good design choice for a variety of reasons and should be adopted at the edge Radio Access Networks (RANs). For one, it ensures fair allocation of the already scarce wireless capacity to the different flows. It reduces the complexity of bandwidth monitoring tools employed by end users, or by the network operators to monitor link utilization. Different protocols can benefit from good bandwidth estimation to improve their performance. For example, bandwidth estimates can help (regular, single interface) TCP tune its optimal window size. Since the number of flows at the edge is anyway small, the scalability of WFQ is not much of an issue.

We now turn to the issue of delay estimates. Obtaining delay estimates for the path between proxy and the BSs during the course of the connection without support from the BSs, is in general a difficult task. This is because, it is difficult to figure out the contribution of queuing delay to the overall end to end delay observed on the path. As mentioned earlier, we don't view this as a serious limitation because of the following reasons. For one, delay estimates during connection setup (where there is no queuing) or estimates from the recent past (few tens of seconds to a few minutes) will most likely be sufficient for the duration of the connection. This is because Internet path delays are known to vary only slowly, over several tens of minutes [24]. Further, any errors in estimation which usually are small (as average delays on the backbone are themselves small) will likely be masked by the transmission and queuing delay at the bottleneck bandwidth. In Equation 1 of section III, A_l dominates $a_i + D_l$ for most packets. We observed this in our experiments as well.

Another choice we made when running the experiments is to disable the use of delayed ACKs. This ensured that during slow start packets always come in pairs at the proxy. If this option is enabled, we still get back to back packets but with less frequency and that number can be greater than 2. Our scheme can be extended to work in this case too but the number of samples we collect for bandwidth estimation can go down. This design possibility is worth further study.

B. Deployment Complexity and Overheads

Our network layer architecture has been designed with the goal of introducing minimum changes to the infrastructure. The only changes needed are software changes at the MH and deployment of proxies, no changes are needed in the radio network or server software. The deployment complexity of our architecture is thus minimal. To increase reliability and scalability of the architecture, we envision multiple proxies, each providing service to a subset of MHs.

As far as the complexity of algorithms go, we note that the implementation of BMP at the MH or PET at proxy is unlikely to be a source of major overhead in terms of memory[4] or CPU requirements. Further, although we have presented BMP as a network layer approach, there is no reason why it cannot be integrated into the TCP receiver. This does not need many changes to the infrastructure, only MHs who want to take advantage of bandwidth aggregation only need apply this patch. This can further reduce some of the state that needs to be maintained at the network layer, which TCP receiver already does.

[4]Maximum buffer size in BMP is at most the size of TCP congestion window, usually under 128 kbytes

There is a source of network overhead in PET-BMP – the need to send a SIG-ACK to proxy for every packet received at the MH. Though this doubles the load in the reverse direction, the additional bandwidth needed is very small as the size of these packets is very small. Even if the wireless links are asymmetric in nature (uplink has much lower bandwidth than downlink), given that we have a choice of more than one interface, there will normally be enough left-over bandwidth to send these packets. In the event this were not the case because of heavy uplink traffic, it is possible to minimize the overhead by performing bandwidth estimation at the client and passing information to the proxy only in the event of a considerable increase or decrease in bandwidth. This possibility needs further evaluation.

C. Miscellaneous issues

Mobility, blackouts, and losses are an integral part of a wireless environment and should be addressed in the design of the network architecture. Since in our architecture, we have mobility support integrated with scheduling, it is easy to handle stalls during handoff by not sending packets on the interface which is performing handoff related processing. Blackouts on an interface can be similarly handled – they can be detected if no SIG-ACKS arrive from MH in response to packets sent on it in a long while.

Another important aspect when performing bandwidth aggregation is to ensure how friendly a TCP flow that uses bandwidth aggregation is to others that don't. Since, we did not make any changes of TCP, it reacts to losses the same way as the other flows and hence bandwidth aggregation does not interfere with other flows. On the contrary, approaches based on opening multiple TCP sockets as in [11], [13] may be too aggressive in face of losses.

An important observation to make is that our network layer solution preserves the semantics of the IP service model, and does not violate the end-to-end design principle. Our solution delays or drops TCP packets, both of which IP is allowed to do in its service model.

VIII. Related Work

Bandwidth aggregation across different logical channels has its origins as a link layer approach in the context of analog dial-ups, ISDN, and ATM [8]–[10]. These approaches require identical, stable link characteristics, special hardware and/or access to the endpoints of the links or specialized headers. This makes it difficult or infeasible to use them in the present scenario, where the Radio Access Networks in question belong to different domains controlled by different providers.

The Stripe protocol [28] is a generic load-sharing protocol that can be used over any logical First-In-First-Out (FIFO) channels, it was implemented in some routers in the context of Multilink PPP. It is based on Surplus Round Robin (SRR) and provides FIFO delivery of these packets to higher layers with minimum overhead in the form of packet processing (looking up the packet sequence number). The design goals of stripe are different from those considered in this paper. SRR boils down to WRR when the packets are of the same size which is likely the case when using TCP for FTP flows. As was shown in earlier sections, WRR introduces a lot of reordering and hence does not perform well in this setting.

Some application layer approaches to bandwidth aggregation using multiple TCP connections have been proposed in [11], [12] albeit in a different context (all the TCP connections are over the same path). Contemporary to our initial work [22], where we introduced the concept of network layer bandwidth aggregation and looked at performance issues faced by real-time applications, some transport and network layer solutions have also been proposed to achieve bandwidth aggregation. The Reliable Multiplexing Transport Protocol (RMTP) [14] is a rate-based transport protocol that multiplexes application data onto different channels. Parallel TCP (pTCP) [13] is another transport layer approach that opens multiple TCP connections one for each interface in use. Another reliable transport protocol proposed for use in message-based signaling in IP networks is the Stream Control Transmission Protocol (SCTP) [29]. SCTP supports multi-streaming and multi-homing. However, it does not ensure in-order delivery across data streams which then has to be handled at the application layer. The focus of this paper is on an architecture that introduces minimum changes to the infrastructure while enabling many diverse services. The application/transport layer solutions cannot achieve the same as they require changes to server software. Further these approaches can be TCP unfriendly when the multiple TCP connections they open share a common bottleneck. Additionally, unless mobility support is integrated within, these approaches may have to rely on a solution similar to ours for mobility support.

A network layer solution based on tunneling similar in spirit to our initial work [22], was proposed in [30] for bandwidth aggregation. This work does not address in depth the architectural components that enable diverse services. Further, the scheduling algorithm proposed for use with TCP was WRR (no buffer management policy was considered). As was demonstrated in this paper, WRR performs poorly if the scheduling does not track varying available bandwidths. Some of the other suggestions that were made to tune TCP parameters to overcome reordering i.e permit large window sizes, set high values for retransmission timeouts and avoid fast retransmissions on DUP-ACKs when implemented, will perform very poorly in presence of losses, even nullifying any benefits that can be had through bandwidth aggregation.

Some modifications to the TCP sender to make the connection more robust to packet reordering have been proposed in [31], [32]. The TCP sender is extended to detect unnecessary retransmissions due to packet reordering through the use of duplicate selective acknowledgment (DSACK). DSACK reports to the sender any duplicate packets received permitting the sender to undo any effects (reduction in congestion window) the spurious retransmission had on congestion control state. This work does not address the problem of bandwidth aggregation, only how to cope with reordering that can be caused by one of many reasons - high-speed switches, satellite links, differentiated services etc including multipath. Thus, PET can also be used in conjunction with this TCP modification, eliminating the need for BMP. Though this warrants further study, we believe BMP is a better approach as it a pro-active approach in that it prevents the need for retransmissions (hence wasting scarce bandwidth) in the first place as opposed to taking a corrective measure. In fact BMP when integrated into the TCP receiver can be viewed as a receiver side modification to TCP to make it robust against reordering.

IX. Conclusions

In this paper, we motivate the advantages of simultaneous use of multiple interfaces and consider a network layer architecture that enables such use. Our network layer architecture provides many different services and is transparent to applications/transport protocols and is easy to deploy. One of the services provided by the architecture is that of BAG - aggregating bandwidth of the multiple interfaces to increase application throughput. We look at this service in the context of TCP applications. The use of multiple paths with varying characteristics introduces challenges in the form of reordering. TCP reacts to the DUP-ACKs generated by the reordered packets by unnecessary fast retransmissions and by invoking congestion control which can lower throughput significantly. To improve the performance of TCP in this setting, we propose a scheduling algorithm PET that minimizes reordering by taking into consideration individual path characteristics. To overcome any residual reordering after PET, we propose a buffer management policy BMP that tries to hide the effects of reordering from TCP by sending packets in order as far as possible. We show through simulation that PET in combination with BMP achieves good bandwidth aggregation under a variety of network conditions. Our solution is thus both performance-effective as well as easy to deploy.

References

[1] H. Balakrishnan, V.Padmanabhan, S.Sheshan, and R.Katz, "A comparision of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, no. 6, pp. 756–769, Dec 1997.

[2] H. Balakrishnan, V.Padmanabhan, and R.Katz, "The effects of asymmetry on TCP performance," *Mobile Networks and Applications*, vol. 4, no. 3, pp. 219–241, Oct 1999.

[3] S. Lu, V. Bharghavan, and R. Srikant, "Fair scheduling in wireless packet networks," *IEEE/ACM Trans. Networking*, vol. 7, no. 4, pp. 473–489, Aug 1999.

[4] A. Campbell, J. Gomez, S. Kim, Z. Turanyi, C.-Y. Wan, and A. Valko, "Comparison of IP micromobility protocols," *IEEE Wireless Communications*, vol. 9, no. 1, pp. 72–82, Feb 2002.

[5] M. Stemm and R. Katz, "Vertical handoffs in wireless overlay networks," *Mobile Networks and Applications*, vol. 3, no. 4, pp. 335–350, 1998.

[6] B. Girod, M. Kalman, Y. Liang, and R. Zhang, "Advances in channel-adaptive video streaming," *Wireless Communications and Mobile Computing*, vol. 2, no. 6, pp. 549–552, Sep 2002.

[7] K. Chebrolu and R. R. Rao, "Bandwidth aggregation for real-time applications in heterogeneous wireless networks," submitted for Publication.

[8] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The PPP multilink protocol (MP)," RFC 1990, Aug 1996.

[9] J. Duncanson, "Inverse multiplexing," *IEEE Commun. Mag.*, vol. 32, no. 4, pp. 34–41, Apr 1994.

[10] *Inverse Multiplexing for ATM (IMA) specification, Version 1.1, ATM Forum Doc. AF-PHY-0086.001*, The ATM Forum Technical Committee Std., 1999.

[11] H. Sivakumar, S.Bailey, and R.L.Grossman, "Psockets: The case for application-level network striping for data intensive applications using high speed wide area networks," in *Proc. IEEE Supercomputing'00*, Dallas, Nov. 2000.

[12] S.Ostermann, M.Allman, and H.Kruse, "An application-level solution to TCP's satellite inefficiencies," in *Proc. WOSBIS'96*, Rye, Nov. 1996.

[13] H. Hsieh and R. Sivakumar, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," in *Proc. ACM MOBICOM'02*, Atlanta, Sept. 2002.

[14] L. Magalhaes and R. Kravets, "Transport level mechanisms for bandwidth aggregation on mobile hosts," in *Proc. IEEE ICNP'01*, Riverside, Nov. 2001.

[15] C. E. Perkins, "Mobile IP," *IEEE Commun. Mag.*, vol. 35, no. 5, pp. 84–99, May 1997.

[16] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *Computer Communication Review*, vol. 26, no. 3, pp. 5–21, Jul 1996.

[17] K. Chebrolu, R. Mishra, P. Johansson, and R. R. Rao, "A network layer architecure to enable multi-access services," unpublished.

[18] D. Brudnicki. Third generation wireless technology. [Online]. Available: http://www.seasim.org/archive/sim102001.pdf

[19] Network simulator. ns2. [Online]. Available: http://www.isi.edu/nsnam/ns

[20] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," in *Proc. ACM SIGCOMM'89*, Austin, Texas, Sept. 1989, pp. 1–12.

[21] Web traffic generator. [Online]. Available: http://www.isi.edu/nsnam/ns/ns-contributed.html

[22] K. Chebrolu and R. R. Rao, "Communication using multiple wireless interfaces," in *Proc. IEEE WCNC'02*, Orlando, Mar. 2002.

[23] L. Kleinrock, Ed., *Queueing Theory*. New York: Wiley, 1975.

[24] A. Acharya and J. Saltz, "A study of internet round-trip delay," Univ. of Maryland, College Park, Tech. Rep. CS-TR 3736, 1996.

[25] S. Keshav, "A control-theoretic approach to flow control," *Computer Communication Review*, vol. 21, no. 4, pp. 3–15, 1991.

[26] V. Jacobson. (1990, Apr.) Modified TCP congestion avoidance algorithm. end2end-interest mailing list. [Online]. Available: ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt

[27] J. Aweya, M. Ouellette, and D. Y. Montuno, "A self-regulating TCP acknowledgment (ACK) pacing scheme," *International Journal of Network Management*, vol. 12, no. 3, pp. 145–163, May/June 2002.

[28] H. Adiseshu, G. Parulkar, and G. Varghese, "A reliable and scalable striping protocol," *ACM Computer Communication Review*, vol. 26, no. 4, pp. 131–141, Oct 1996.

[29] R. Stewart *et al.*, "Stream control transmission protocol," RFC 2960, Oct. 2000.

[30] D. S. Phatak and T. Goff, "A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments," in *Proc. IEEE INFOCOM'02*, New York, June 2002, pp. 773–781.

[31] E. Blanton and M. Allman, "On making tcp more robust to packet reordering," *Computer Communication Review*, vol. 32, no. 1, pp. 20–30, Jan 2002.

[32] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "Improving TCP's performance under reordering with DSACK," International Computer Science Institute, Berkeley, Tech. Rep. TR-02-006, Jul 2002.