Peer-to-Peer File Transfer in Wireless Mesh Networks

Sherif M. ElRakabawy and Christoph Lindemann
University of Leipzig
Department of Computer Science
Augustusplatz 10-11
04109 Leipzig, Germany
http://rvs.informatik.uni-leipzig.de/

***Abstract*— In this paper, we consider the peer-to-peer transfer of popular files between devices in a wireless mesh network. We address the problem that occurs when multiple nodes try to access the same file simultaneously, resulting in increased contention on the shared wireless channel. To counteract this problem, we propose a cooperative file transfer protocol which splits a file into fixed-sized pieces and allows simultaneous downloads of such pieces from multiple peers. Opposed to previous approaches, the proposed protocol selects the potential download peers such that the corresponding download paths possess minimum interference among each other. In a performance study where we compare our approach with other download schemes proposed in the literature, we show that our cooperative protocol roughly halves the time required for downloading a file.**

***Index Terms*— Data dissemination, Peer-to-Peer systems, Network applications, Performance evaluation.**

I. Introduction

Support for simultaneous downloads of a file from multiple sources is included in recent releases of many popular peer-to-peer (P2P) file sharing systems for the Internet, e.g., KaZaA [16] and many implementations of the Gnutella protocol [15]. Typically, such systems split a file into pieces of equal size, commonly denoted as *chunks*. For exploiting a peer's entire downlink bandwidth, multiple chunks are downloaded in parallel from different peers, where the source for each chunk is chosen more or less greedily. Recently, it has been observed that a sophisticated scheme for exchanging chunks among peers can increase the total service capacity of the system by creating diversity. That is, each peer does not only consume upload capacity from the system, but can also provide upload capacity to some other peers, making the system self-scaling. Such approaches are typically denoted as *swarming* approaches.

The potential of swarming protocols encouraged the development of second generation P2P systems like cooperative content delivery systems [6], [21]. Due to the self scaling property, a cooperative content delivery system is highly efficient and even resilient to *flash crowds* - sudden request burst for popular web content that make the origin server collapse. Furthermore, cooperative content delivery systems are self-organizing and can be deployed without supporting infrastructure, challenging classical content delivery networks that deploy thousands of servers around the world. Application scenarios for cooperative content delivery systems include providing an software update to a known set of computers or simultaneous access to popular content by many users [12].

The ongoing success of wireless communication technology permanently extends the coverage to the Internet. A recent trend for enabling cost efficient Internet access are wireless mesh networks [1]. Figure 1 depicts the two-tier urban mesh access network as considered in [7], which comprises a backhaul tier of fixed mesh routers and an access tier of mobile clients. Such networks permit sharing of digital content among wireless devices or accessing content through the Internet by using content delivery systems. These systems support a wide scope of utilities, spanning applications that need to access large multimedia files for infotainment or programs that maintain the wireless network by providing critical software updates. Wireless mesh networks are particularly sensitive to flash crowd-like requests for popular content by many devices, since all devices share a common wireless medium. In particular, in the spatial proximity of a gateway that connects many devices to the origin server of popular content, severe congestion of the wireless medium may occur. In such scenarios, cooperative file transfer among wireless devices using a swarming approach may prove useful for avoiding network congestion. Consistent with [7], we assume that the wireless network runs a MANET routing protocol, e.g., *Ad hoc On demand Distance Vector* (*AODV*, [18]).

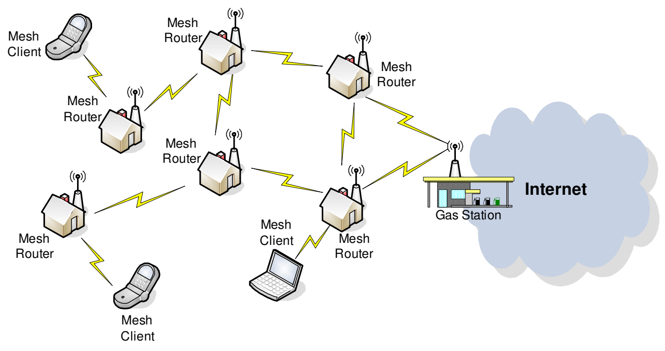

Figure 1: Two-tier urban wireless mesh network

In this paper, we present a cooperative P2P file transfer protocol for wireless mesh networks. As major contribution, our protocol applies a swarming approach to balance the load in the network. Beyond recent proposals for cooperative content delivery in wireless multihop networks, our protocol downloads multiple chunks from several peers in parallel, exploiting diversity and achieving a higher utilization of the wireless channel in the proximity of the downloading peer. Download peers are selected based on the current load on the download paths. In fact, our protocol comprises an iterative probing algorithm that determines a set of download peers, which maximize aggregate goodput. The order of peers to probe is determined by a procedure denoted as *interference dependency estimation*, that ensures the selection of peers with a minimum interference on the download paths.

In a simulation study we compare our protocol to other approaches for cooperative content delivery. These include an approach that do not employ parallel downloads and select a single download peer randomly (inspired by [12]) or based on the network proximity in hops (inspired by [17]), respectively. The results of the simulation study show that our protocol decreases the required time for downloading a file by more than 50%.

The remainder of this paper is organized as follows. Section II summarizes related work on cooperative file transfer. Our cooperative protocol is introduced in Section III, whereas Section IV compares the performance of our protocol with two other approaches for cooperative file transfer. Finally, concluding remarks are given in Section V.

II. Related Work

Several studies as e.g. [2], [4], [5], [12], [19], and [22] has extensively investigated swarming as a technique for optimizing content delivery performance in the Internet. Proposals of swarming-based systems include BitTorrent [6] and Slurpie [21]. BitTorrent [6] uses either a centralized server or a decentralized distributed hash table for discovering peers that store chunks of a desired file. Parallel downloads are performed from several peers selected by a tit-for-tat incentive mechanism. Slurpie [21] features decentralized peer discovery and an algorithm to estimate the bandwidth of the local peer's Internet connection. Building upon the assumption that such connection is the major bottleneck, Slurpie adjusts the number of connections to other peers as well as the number of simultaneous downloads according to the acquired bandwidth estimates. Similar to BitTorrent and Slurpie, our protocol performs parallel downloads from multiple peers. Opposed to BitTorrent and Slurpie, our protocol selects peers with non-interfering download paths. That is, other than the case in the Internet, nodes in a wireless mesh network can interfere with each other, making the need for a peer selection strategy based on non-interfering download paths inevitable for parallel downloads.

As a first approach for swarming in ad hoc wireless networks, Nandan et. al presented SPAWN [17]. SPAWN is designed for hybrid wired / wireless vehicular ad-hoc networks with Internet connectivity, performs peer selection based on network proximity, and downloads one chunk at a time. In contrast, our protocol is designed for wireless mesh networks and performs peer selection based on the current network load on paths to potential download peers. Furthermore, our protocol simultaneously downloads from multiple peers, given that non-interfering paths to such peers exist. Recently, Rajagopalan et. al proposed and adoption of BitTorrent for wireless multihop networks denoted as BTM [20]. BTM uses a cross-layer approach that performs peer discovery assisted by the MANET routing protocol, while peer and content selection are implemented according to the BitTorrent specification. Opposed to BTM, we focus on developing a peer selection algorithm for wireless mesh networks which accounts for the special characteristics of such networks.

Load-based path metrics have recently been studied in the context of routing protocols for wireless mesh networks [8], [10]. Similar to [8], [10], our protocol makes decisions based on the current load and interference on a path. However, while path metrics enable discovery of an optimal path for a given source-destination pair, our protocol aims at determining an optimal source for a fixed destination in order to maximize the aggregate goodput..

III. Cooperative File Transfer

A. Overview

Our cooperative file transfer protocol incorporates a peer discovery as well as a peer selection mechanisms, which are described in this section. Subsequently, we denote a mobile node which wants to retrieve a file as the *downloader*. We assume that files have (pseudo) unique identifiers, e.g., given by SHA-1 hash sums over the file contents. For transfer, a file is split into chunks of fixed size. Chunks are identified by numbering them in ascending order from the beginning to the end of the file.

First, the downloader initiates a peer discovery procedure to find potential *download peers* sharing chunks of the file. Subsequently, the downloader runs the peer selection algorithm to determine a set of *active download peers*. Consistent with [6], [12], and [17], chunks are downloaded following rarest-chunk-first content selection strategy. Chunks are downloaded using the transmission control protocol, TCP, and are explicitly requested by the downloader using the chunk number. Our protocol monitors the performance of a download using end-to-end goodput measurements. Since our protocol does not require any support from the transport layer, any TCP variant can be employed for downloading. However, we emphasize that the TCP variant should provide good fairness between concurrent flows in order to achieve a performance improvement when performing simultaneous downloads. Therefore, we employ TCP-AP [11] which has proven to provide considerably better fairness between competing flows in multihop wireless networks than TCP NewReno. Nevertheless, in Section IV, we also conduct a simulation where we compare the performance of our protocol running TCP NewReno as transport layer protocol.

For each file that is currently downloaded, our protocol maintains a basic data structure which is updated during the peer discovery and selection procedures. The data structure is denoted as *file distribution table* (FDT). Each entry in FDT contains the IP address of a download peer as well as the numbers of the chunks stored by this peer.

B. *Peer Discovery*

The focus of our protocol lies on the peer selection procedure. Thus, we just include a simple peer discovery mechanism to provide a complete system for cooperative content delivery. To initiate the download of a given file, a node issues a QUERY message which contains the file ID as well as the chunk numbers of the requested chunks. For a file consisting of C chunks, chunk numbers can be encoded by a bit array with C bits, where bit c is set to 1, if the downloader is interested in chunk c, and to 0 otherwise, $1 \leq c \leq C$. The QUERY message is transmitted to all reachable nodes by flooding. When a download peer receives the QUERY message, it checks whether it possess requested chunks. If it does, it sends a RESPONSE message containing the file ID and the chunk numbers of available chunks to the downloader via unicast. Upon receipt of the RESPONSE message, the downloader updates its FDT by assigning the chunks in the RESPONSE message to the corresponding peer. The information in FDT is used for peer selection as described below.

Similar to [20], we use a cross-layer approach to embed protocol-specific messages into control packets of the routing protocol. That is, QUERY messages of our protocol are piggybacked on route request messages (RREQ) of the routing protocol. Similar to the QUERY/RREQ procedure, RESPONSE messages are piggybacked on the route reply (RREP) messages of the routing protocol and returned to the downloader during the route reply procedure.

Opposed to [20], we do not use expanding ring search for limiting the scope of flooding, since it is crucial for our peer selection algorithm that potential download peers have large distances, as we show in Section V. Using this piggyback method instead of application layer flooding during peer discovery has two advantages: First, a node need not run an instance of our protocol in order to forward QUERY and RESPONSE messages. Second, the piggyback method significantly reduces control traffic, since both peer discovery and route discovery can be performed in a single step.

C. *Peer Selection*

Previous approaches for cooperative content delivery like SPAWN [17] and BTM [20] assume that the closest-peer provides the highest TCP goodput and, thus, employ network proximity as metric for peer selection. These previous approaches are based on the observation that the TCP goodput degrades as the number of hops increases [11], [14]. However, such a proximity-based metric does not account for the current link layer contention between the path to the downloader and other flows in the network.

To demonstrate the impact of background traffic on TCP goodput and to show that the shortest path does not always

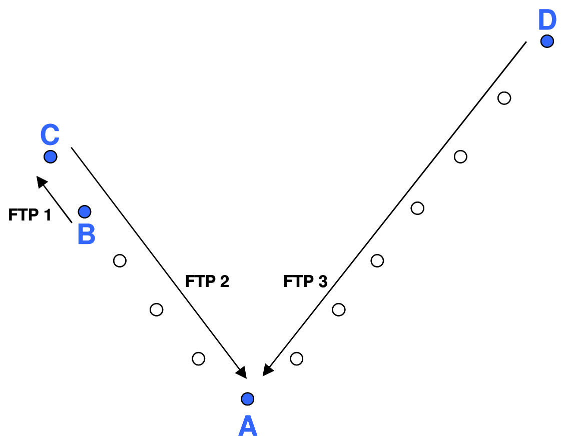

Figure 2: Simple scenario with two chains of different lengths

provide the best goodput, we conduct a simple simulation using the scenario depicted in Figure 2. The distance between adjacent nodes is 200m and the interference/carrier sensing ranges are 550m. We define three FTP flows: FTP 1 runs from node D to node A, FTP 2 runs from node C to node A and FTP 3 runs from node B to node C. This scenario simulates the case where node A constitutes a downloader which wishes to download a chunk either from node C or node D. According to the adopted quality selection metric, node A would either choose the nearest peer like in [17], [20] (C in this case) or the peer which provides the best end-to-end goodput, in case the traffic load on the paths is also considered.

To show the impact of the metric used for peer selection on the achieved end-to-end goodput, we consider four cases:

(1) FTP 2 is the only flow active, simulating the case where node A decides to download the chunk from node C.
(2) FTP 3 is the only flow active, simulating the case where node A decides to download the chunk from node D.
(3) FTP 2 is active while the background flow FTP 1 is also active, simulating the case where node A decides to download the chunk from node C with background traffic present.
(4) FTP 3 is active while the background flow FTP 1 is also active, simulating the case where node A decides to download the chunk from node D with background traffic present.

Figure 3 shows the results for cases (1) and (2) whereas Figure 4 shows the results for cases (3) and (4). In Figure 3, we observe that in case there is no background traffic produced by FTP 1, the goodput on the short path to node C outperforms the goodput on the long path to node D by 24%. However, as soon as there is background traffic present, the interference on the short path increases due to the traffic of FTP 1, which decreases the achieved goodput significantly as shown in Figure 4. Since FTP 3, which runs on the longer path is out of the interference range of FTP 1, it achieves around 66% more goodput than FTP 2 which runs on the short path. This shows that the shortest path does not always provide the best goodput. To that end, our protocol is designed to select download peers with the least-loaded paths.

Our protocol incorporates two basic peer selection metrics

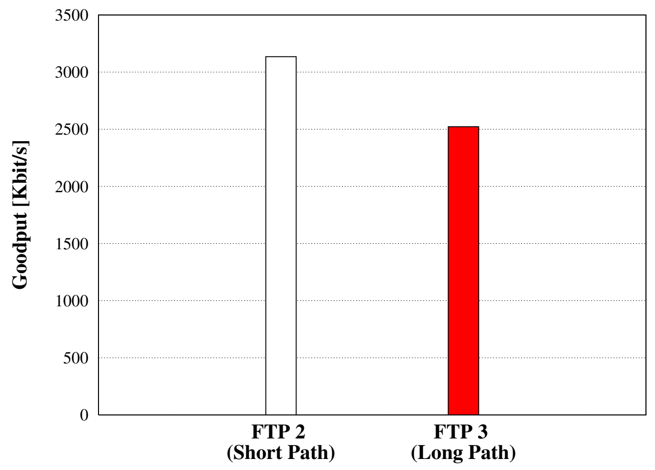

Figure 3: Goodput of FTP 2 and FTP 3 without background traffic of FTP 1

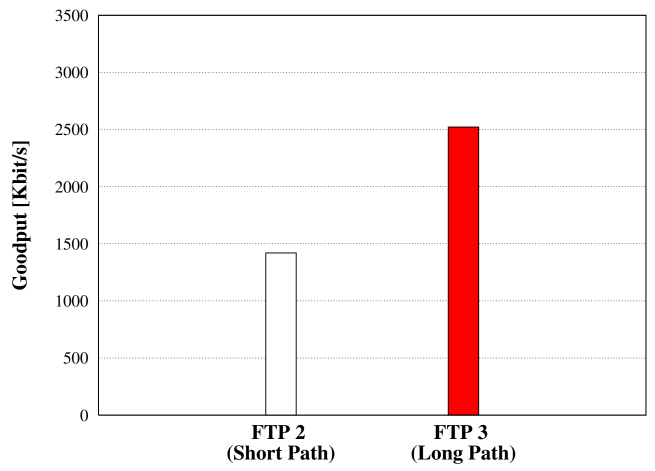

Figure 4: Goodput of FTP 2 and FTP 3 with background traffic of FTP 1

which account for the end-to-end load on the paths to a potential download peer. The first metric, denoted as response delay, is defined by the time elapsed between issuing a QUERY and receiving a corresponding RESPONSE message for each potential download peer. Obviously, the response delay depends on the current link layer contention on the path to a potential download peer due to the fact that high load increases the number of link layer retransmissions required for delivering the RESPONSE message.

Unfortunately, starting the first download will highly influence the path load in the network. Thus, response delay measurements can only be used to identify the first active download peer. Therefore, we use end-to-end goodput measurements as additional metric. The goodput of a download peer is defined by the number of unique bytes received from the peer divided by the time required for receiving these bytes, whereas the aggregate goodput constitutes the sum of the goodput of all active download peers. Since goodput can only be determined while transferring chunks from some download peers, our protocol uses an algorithm that optimistically probes some combinations of download peers in order to maximize the aggregate goodput.

As a major difference to cooperative content delivery systems for the Internet, a content delivery system for wireless mesh networks must consider that potential download peers use a shared channel. Therefore, simultaneous downloads from multiple peers only increase the aggregate goodput if the download paths do not significantly interfere with each other. For discovering non-interfering paths, the downloader uses a mechanism denoted as *Interference Dependency Estimation* (*IDE*) to measure the end-to-end delay to potential download peers while downloading from active download peers. Specifically, *IDE Echo Request* (*IDEREQ*) packets are transmitted back-to-back to all potential download peers. Peers receiving such IDEREQ immediately reply with an *IDE Echo Reply* (*IDEREP*) packet. Upon receiving the IDEREP message, the downloader computes the delay between issuing the IDEREQ message and receiving the IDEREP message and assigns the delay to the corresponding peer. Recall that network routes to potential peers are established during peer discovery so that an IDEREQ does not trigger a new route discovery procedure. Similar to the response delay, the IDE delay reflects the level of interference on the download path to a potential peer. In particular, it indicates how much starting a download from a new download peer will interfere with downloads from currently active download peers.

In order to improve the reliability of such end-to-end measures, the downloader computes the end-to-end delay as the average delay by sending five IDEREQ messages to each potential peer. Note that the IDEREQ/IDEREP functionality is very similar to ICMP Echo functionality. However, we choose not to use default ICMP Ping messages, since we use the IDEREQ/IDEREP messages to piggyback further protocol-specific information, e.g., about newly available chunks. The IDEREQ/IDEREP messages are always padded to the size of 1400 Bytes.

Peer Selection Algorithm

The peer selection operates iteratively in multiple rounds. In each round, $r = 0, 1, ...$, it optimistically adds a new peer to the current active download peers in order to increase the aggregate goodput. Which peer to add is determined using the IDE procedure described above, since an optimal peer combination comprises peers with little mutual interference on the corresponding download paths. The algorithm continues probing a new peer in each round until the aggregate goodput stops improving. Then, the algorithm terminates and returns the set of peers which provided the highest aggregate goodput.

A set $\mathbb{P}$ includes all potential download peers identified during peer discovery. $\mathbb{P}_r$ determines the active download peers after round r. The priority of peer $p \in \mathbb{P}$ in round r is denoted as $\delta_r(p)$, whereas p_r is the peer selected for probing in round r. A parameter α_r denotes the aggregate goodput achieved by simultaneous downloads from the active download peers $\mathbb{P}_r$ in round r. *RespDel*(p) denotes the response delay value for peer $p \in \mathbb{P}$ during peer discovery, *IDE*(p) represents the IDE delay of peer p, and *Goodput*($\mathbb{Q}$)

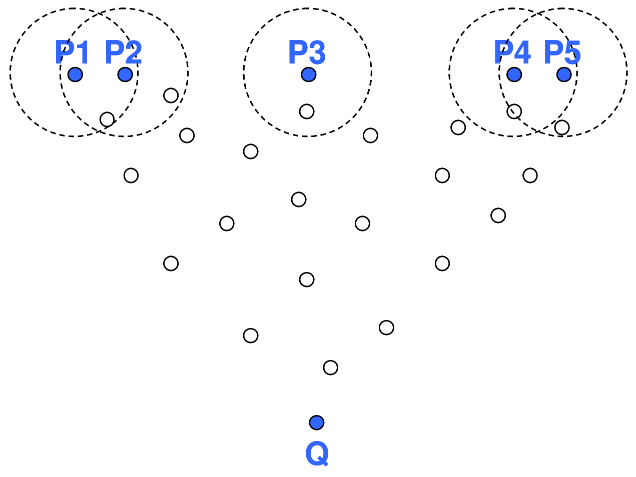

Figure 5: Sample scenario

TABLE 1: OPERATION OF OUR PROTOCOL IN THE SAMPLE SCENARIO OF FIG. 5

Round r	p_r	$\delta_r(p)$	α_r	$\mathbb{P}_r$
0	-	$\delta_0(P3) < \delta_0(P2) < \delta_0(P4) < \delta_0(P5) < \delta_0(P1)$	0	$\varnothing$
1	P3	$\delta_1(P2) < \delta_1(P4) < \delta_1(P5) < \delta_0(P1)$	$\alpha_1 > 0$	{P3}
2	P2	$\delta_2(P4) < \delta_2(P5) < \delta_2(P1)$	$\alpha_2 > \alpha_1$	{P2, P3}
3	P4	$\delta_3(P1) < \delta_3(P5)$	$\alpha_3 > \alpha_2$	{P2, P3, P4}
4	P1	$\delta_4(P5)$	$\alpha_4 < \alpha_3$	{P2, P3, P4}

denotes the aggregate goodput of simultaneous downloads from all peers $p \in \mathbb{Q}$, $\mathbb{Q} \subset \mathbb{P}$.

After issuing a QUERY and receiving RESPONSE messages from the download peers, for all responding peers $p \in \mathbb{P}$, the initial priorities $\delta_0(p)$ are initialized by *RespDel*(p) for the first round. During each round r, chunks are continuously downloaded from the active peers $\mathbb{P}_{r-1}$ identified in round r - 1. That is, as soon as an active peer p is idle, the downloader sends a request for a chunk to p. The chunk is chosen according to the rarest-chunk-first content selection strategy, i.e., the missing chunk with the fewest copies available at the potential download peers. Such a chunk can be easily identified in FDT. While downloading simultaneously from active peers, the algorithm chooses the peer p_r with the highest priority (i.e. the lowest delay) $\delta_{r-1}(p)$ from all non-active download peers $p \in \mathbb{P} \setminus \mathbb{P}_{r-1}$. To probe if selecting p_r as additional active download peer increases the aggregate goodput, a chunk c is downloaded from p_r. Again, c is selected according to the rarest-chunk-first strategy.

In order to determine $\delta_r(p)$ for a potential next round, the IDE procedure is performed for all other download peers $p \in \mathbb{P} \setminus (\mathbb{P}_{r-1} \cup \{p_r\})$. Since IDE is performed simultaneously while downloading from all peers $p \in \mathbb{P}_{r-1} \cup \{p_r\}$, IDE delays quantify interference with all current downloads. Recall that information about chunk availability may be piggybacked on IDEREQ/IDEREP messages. Thus, FDT gets updated during the IDE procedure.

After downloading the complete chunk c from p_r, the aggregate goodput α_r of all simultaneous downloads since the beginning of round r is computed and compared to the aggregate goodput α_{r-1} of the previous round. If the aggregate goodput α_r exceeds the aggregate goodput α_{r-1}, the peer p_r improves the overall aggregate goodput and thus is added to the set of active download peers after round r, $\mathbb{P}_r$. The algorithm continues until the aggregate goodput stops improving, i.e., $\alpha_{r'} < \alpha_{r'-1}$ in a round r'. After terminating, the algorithm returns the set of active peers with the highest goodput, i.e., $\mathbb{P}_{r'-1}$.

D. Sample Scenario

We illustrate the operation of our algorithm using a scenario with multiple download peers shown in Figure 5. The downloader Q issues a QUERY for a file f of size 100 MBytes. Download peers P1 to P5 possess file f. The dotted circles around the download peers indicate the interference range of a download peer, i.e., peers P1 and P2 both interfere with each other as well as peers P4 and P5. Relay nodes, which may also interfere with each other, are placed randomly between downloader Q and peers P1 to P5 such that they ensure full connectivity.

After peers P1 to P4 receive the QUERY message for file f, they send RESPONSE messages. Subsequently, the downloader Q receives the RESPONSE messages in the order P3, P2, P4, P5, and P1. For each round r, Table 2 shows the key values of the peer selection algorithm. the potential download peer p_r, the priorities $\delta_r(p)$, the aggregate goodput α_r, and the set with the active download peers after the round $\mathbb{P}_r$. We observe that the aggregate goodput improves up to the third round by including a new peer in the set $\mathbb{P}_r$ in each round. Since adding an additional peer at round four does not improve the aggregate goodput, the algorithm terminates with {P2, P3, P4} as set of active download peers.

IV. PERFORMANCE EVALUATION

We compare the performance of our protocol with two other approaches for cooperative content delivery in multihop wireless networks with related peer selection strategies. The first approach is the *random-peer* approach, which chooses a download peer randomly. Such a random selection strategy is adopted in some Gnutella implementations and delivers good performance in the Internet compared to other peer selection strategies [12]. The second approach is the *closest-peer* approach implemented in the SPAWN protocol [17], which chooses the closest download peer in terms of number of hops.

The simulation experiments are conducted using the network simulator ns-2 [13] with AODV [18] as MANET routing protocol. We deploy the IEEE 802.11g link layer protocol which is configured to provide a transmission range of 250m and a carrier sensing range as well as an interference range of 550m. Consistent with [17], the Request-To-Send/Clear-To-Send (RTS/CTS) handshake is enabled. We consider a wireless channel bandwidth of 54 Mbit/s and set the size of TCP data packets to 1460 Bytes. As already mentioned, we employ TCP-AP [11] as a transport layer protocol for reliable data delivery, since TCP-AP provides considerably better fairness between competing flows than TCP NewReno [11]. However, in order to study the performance gained when running TCP-AP, we also conduct a simulation experiment where we compare the performance of our protocol running TCP NewReno. We assume that files are available in the wireless domain, e.g., through Internet downloads, storage media or self-generated pictures and videos. Thus, we do not consider Internet gateways in our scenarios, but rather focus on the content delivery within the wireless mesh network.

A. *Sample Scenario*

Figure 6 plots the aggregate goodput achieved at the downloader for the sample scenario shown in Figure 5. We observe that both the random-peer and closest-peer approaches achieve almost the same aggregate goodput, whereas our protocol achieves up to 100% more aggregate goodput than both approaches.

Figure 7 shows the time required for downloading the requested file for each considered variant. Consistent with the results of Figure 6, we see that our protocol achieves around 50% less download time than the random-peer and closest-peer variants.

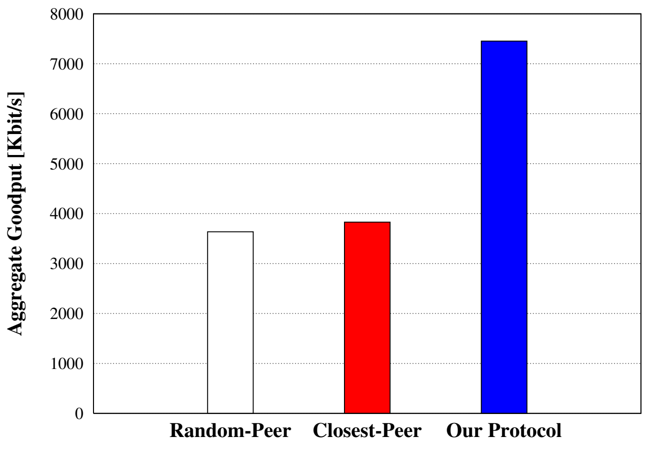

Figure 6: Sample scenario: Aggregate goodput at downloader (running our protocol over TCP-AP)

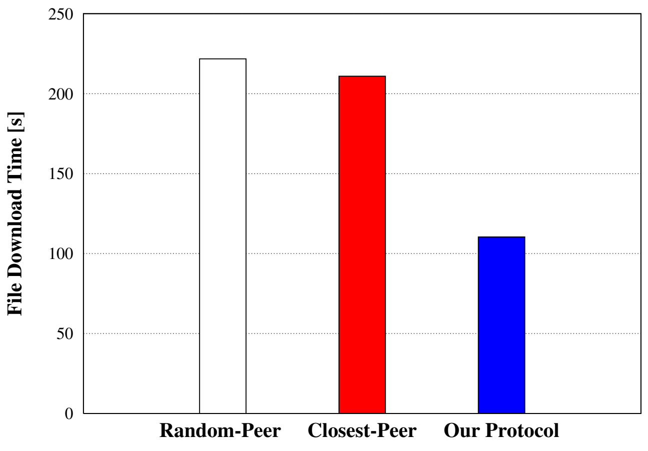

Figure 7: Sample scenario: Time required to download the requested file (running our protocol over TCP-AP)

B. *Static Scenario with Random Placement*

To evaluate the performance of the considered peer selection approaches in absence of disturbing effects caused by mobility (e.g route failures), we consider a scenario where all nodes in the network are stationary. The network constitutes 120 nodes which are distributed randomly on a flat area of 2500m x 1000m. According to [3], all nodes in the network can communicate with each other over one or more hops with probability 0.999. For this set of simulations, we perform 40 independent replicates for each experiment, each replicate with a different random node placement. The considered performance measures are then derived from the values of the 40 replicates with 95% confidence intervals.

In order to determine the best chunk size for which our protocol achieves the best performance, we conduct an experiment where we vary the chunk size from 10 KBytes up to 1000 KBytes and plot the average download time of the file accordingly. In this experiment, a node requests a file of 50 MBytes while there exist ten download peers which possess this file. We plot both the original curve as well as its Bezier-smoothed version to illustrate the smoothed gradient of the aggregate goodput. Figure 8 shows that the average download time is relatively high for a chunk size of 10 KBytes and drops sharply as the chunk size increases, where it becomes stable for a chunk size of 200 KBytes and above. The reason for such a behavior is that for small chunk sizes below 200 KBytes, the TCP goodput measurements are inaccurate. This is because TCP cannot fully utilize the available bandwidth fast enough since the growth of the TCP window as well as the adaptive pacing rate of TCP-AP are dependent on the number of received TCP acknowledgments. That is, small chunks lead to a small number of TCP packets which results in a small number of TCP acknowledgments, yielding inaccurate goodput measurements and inappropriate selection of download peers. For larger chunks of 200 KBytes and above, goodput measurements get more accurate. We use a chunk size of 200 KBytes for our protocol, since this size is large enough to provide accurate goodput measurements, yet small enough to grant flexibility in case the peer selection algorithm decides to choose different download peers.

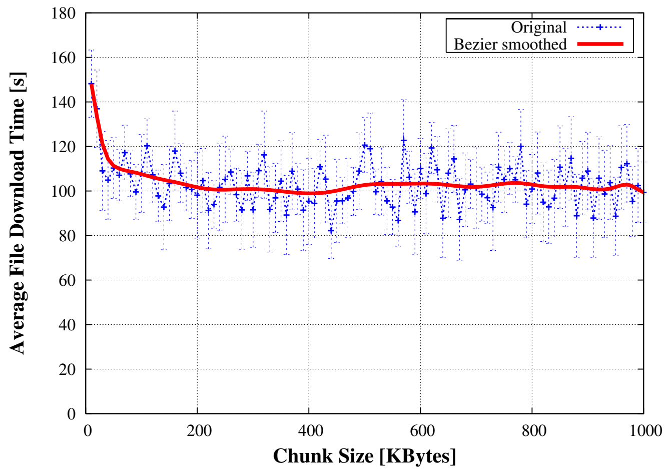

Figure 8: Static random scenario: Avg. file download time vs. chunk size (running our protocol over TCP-AP)

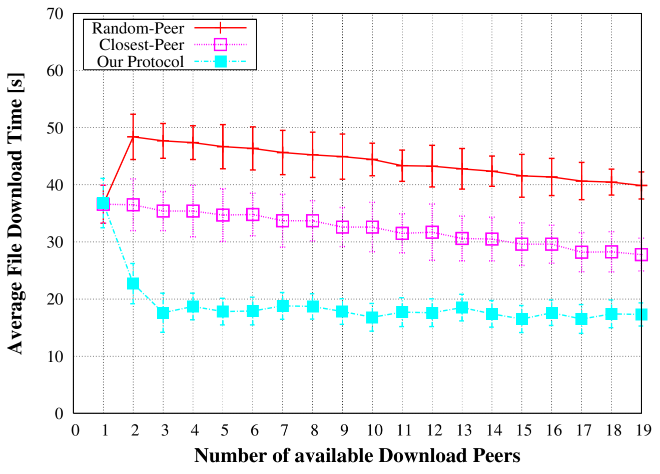

Figure 9: Static random scenario: Avg. file download time vs. number of initially available download peers for 1 downloader (running our protocol over TCP-AP)

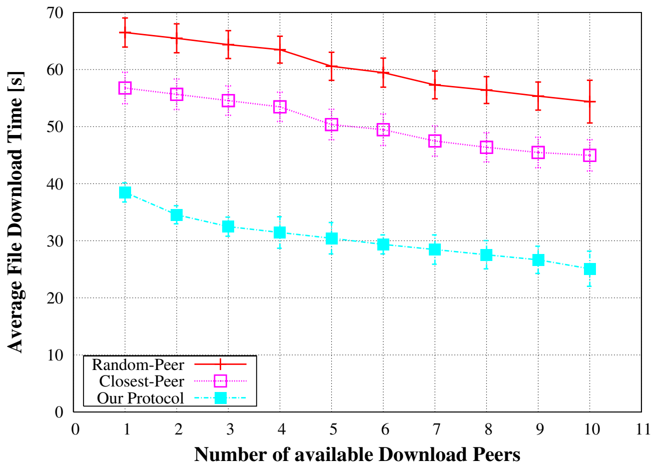

Figure 10: Static random scenario: Avg. file download time vs. number of initially available download peers for 10 downloaders (running our protocol over TCP-AP)

In a further experiment, we define a fixed number of downloaders which request a 10 MBytes file, while varying the number of initially available download peers. Figure 9 shows the average file download time for one downloader, whereas Figure 10 plots the average file download time over 10 downloaders. In Figure 9 we observe that our protocol achieves up to 60% less average download time than all other approaches, i.e., the file can be downloaded in less than half of the time. We make a number of interesting observation from the curves in Figure 9: First, we see that the download time of all considered download strategies is similar for one download peer since there is no peer selection, i.e., only one download peer is available. Second, we see that the download time of our protocol decreases with increasing number of available download peers up to three peers and then it stabilizes. From this behavior we conclude that due to node interference and due to the nature of the shared wireless channel, simultaneous downloads from more than three peers does not yield any improvement. Third, we observe that the download time of the random-peer and closest-peer approaches decreases as the number of available download peers increases. The reason for such a behavior is that the probability for selecting a close peer increases with increasing number of available download peers, which yields more goodput and thus less download time.. The findings in Figure 10 are consistent with our observations in Figure 9. Note that in Figure 10, the average download time of our protocol increases beyond three available download peers since we consider the average download time over ten downloaders and not only one downloader like in Figure 9.

To study the performance of our protocol over a standard TCP variant, we conduct a further simulation where we study the performance of our protocol using TCP NewReno and compare it to TCP-AP [11]. Figure 11 shows the file download time for the considered variants for one downloader and ten initially available download peers whereas Figure 12 shows the file download time for 10 downloaders and 10 download peers. Note that, as consistent throughout this paper, we denote our protocol running TCP-AP simply by "Our Protocol". In Figure 11 we see that our protocol running TCP NewReno yields almost the same file download time like the closest-peer download strategy, i.e, it requires almost double the download time compared to our protocol running TCP-AP. The reason for such performance difference is the bad fairness of TCP NewReno which does not allow the full utilization of the simultaneous downloads from multiple peers. That is, due to the aggressive window strategy of TCP NewReno, only a small fraction of the active TCP flows acquire the available bandwidth at cost of other flows in the vicinity [11]. On the contrary, TCP-AP gains better fairness results, preventing simultaneous active flows from starving due to the aggressiveness of other flows in the vicinity.

In Figure 12 we notice that our protocol running TCP NewReno yields slightly better performance than in Figure 11. This is because the traffic load in the network is distributed over more nodes since there are ten downloaders present rather than one like the case in Figure 11. This results in less interference between the active flows and thus better fairness results. Nevertheless, our protocol running TCP NewReno still requires around 40% more download time than the case when running TCP-AP.

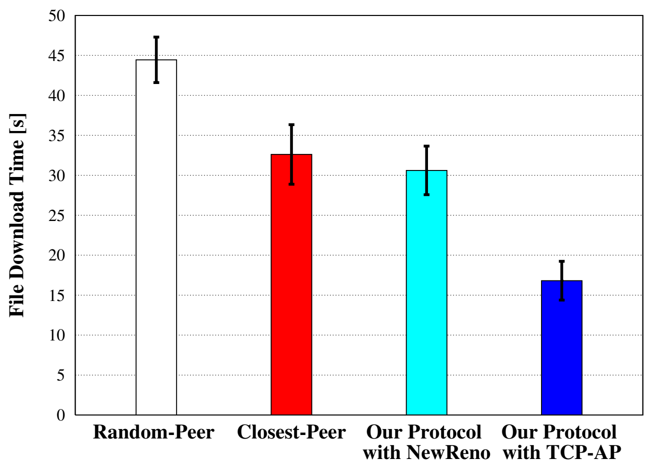

Figure 11: Static random scenario: Avg. file download time for 1 downloader and 10 initially available download peers

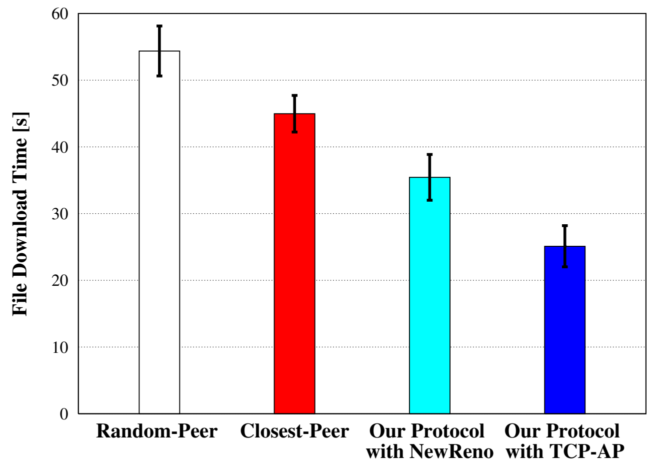

Figure 12: Static random scenario: Avg. file download time for 10 downloaders and 10 initially available download peers

V. Conclusion

In this paper, we introduced a cooperative file transfer protocol for wireless mesh networks. The proposed protocol is designed for bulk delivery of popular digital content in wireless mesh networks, and implements a chunk-based swarming approach. Beyond previous proposals for cooperate file transfer in wireless mesh networks, our protocol uses an advanced end-to-end peer selection algorithm that selects potential download peers based on the current load and interference on the download paths. Furthermore, our protocol exploits diversity by downloading multiple chunks simultaneously such that the chosen download paths possess minimum interference among each other.

Compared to other approaches for cooperate file transfer in wireless mesh networks, our protocol roughly halves the time required for downloading a file and is easily deployable since it operates in a pure end-to-end fashion and requires no support from lower layer protocols.

In future work, we are investigating the performance of our protocol in presence of node mobility in order to identify mobility-induced problems and extend our protocol accordingly.

Acknowledgements

The authors would like to thank Oliver Waldhorst for his suggestions which significantly improved the clarity and presentation of this paper.

References

[1] I. F. Akyildiz, X. Wang, and W. Wang, Wireless Mesh Networks: a survey, Computer Networks, 47, 2005.

[2] F. Bin, D. Chiu, J. Lui, Stochastic Analysis and File Availability Enhancement for BT-like File Sharing Systems, *Proc. IEEE IWQoS*, New Haven, CT, 2006.

[3] C. Bettstetter, On the Minimum Node Degree and Connectivity of a Wireless Multihop Network, *Proc. ACM MobiHoc*, Lausanne, Switzerland, 80-91, 2002.

[4] E. Biersack, P. Rodriguez, and P. Felber, Performance Analysis of Peer-to-Peer Networks for File Distribution, *Proc. QofIS*, Barcelona, Spain, 1-10, 2004.

[5] Y. Chiu and D. Eun, Minimizing File Download Time over Stochastic Channels in Peer-to-Peer Networks, *Proc. CISS*, Princeton, NJ, 2006.

[6] B. Cohen, Incentives Build Robustness in BitTorrent, *Proc. P2P ECON*, Berkeley, CA, 251-260, 2003.

[7] J. Camp, J. Robinson, C. Steger, and E. Knightly, Measurement Driven Deployment of a Two-Tier Urban Mesh Access Network, *Proc. ACM MobiSys*, Uppsala, Sweden, 96-109, 2006.

[8] D. De Couto, D. Aguayo, J. Bicket, and R. Morris, A High-Throughput Path Metric for Multihop Wireless Routing, *Proc. ACM MobiCom*, San Diego, CA, 134-146, 2003.

[9] R. Draves, J. Padhye, and B. Zill, Comparison of Routing Metrics for Static Multi-Hop Wireless Networks, *Proc. ACM SIGCOMM*, Portland, OR, 367-378, 2004.

[10] R. Draves, J. Padhye, and B. Zill, Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks, *ACM MobiCom*, Philadelphia, PA, 114-128, 2004.

[11] S. ElRakabawy, A. Klemm, and C. Lindemann, TCP with Adaptive Pacing for Multihop Wireless Networks, *Proc. ACM MobiHoc*, Urbana-Champaign, IL, 2005.

[12] P. Felber and E. Biersack, Cooperative Content Distribution: Scalability Through Self-Organization, *Self-star Properties in Complex Information Systems*, Ö. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. van Moorsel, and M. van Steen (Eds.), LNCS 3460, Springer, 2005.

[13] K. Fall and K. Varadhan (Ed.), The ns-2 Manual, Technical Report, The VINT Project, UC Berkeley, LBL, USC/ISI and Xerox PARC, 2005.

[14] M. Gerla, K. Tang, and R. Bagrodia, TCP Performance in Wireless Multi-Hop Networks, *Proc. IEEE WMCSA*, New Orleans, LA, 1999.

[15] Gnutella Protocol Development. *http://rfc-gnutella.sourceforge.net/*.

[16] KaZaA homepage. *http://www.kazaa.com/*.

[17] A. Nandan, S. Das, G. Pau, M. Gerla, and M.Y. Sanadidi, Cooperative Downloading in Vehicular Ad-hoc Wireless Networks, *Proc. WONS*, St. Moritz, Switzerland, 2005.

[18] C. Perkins, E. Belding-Royer, and S. Das, *Ad hoc On-Demand Distance Vector (AODV) Routing*, IETF RFC 3561, 2003.

[19] D. Qiu and R. Srikant, Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks, *Proc. ACM SIGCOMM*, Portland, OR, 367-378, 2004.

[20] S. Rajagopalan, and C. Shen, A Cross-layer, Decentralized BitTorrent for Mobile Ad hoc Networks, *Proc. MOBIQUITOUS*, San Jose, CA, 2006.

[21] R. Sherwood, R. Braud, and B. Bhattacharjee, Slurpie: A Cooperative Bulk Data Transfer Protocol, *Proc. IEEE INFOCOM*, Hong Kong, 941-951, 2004.

[22] X. Yang and G. de Veciana, Service Capacity of Peer to Peer Networks, *Proc. IEEE INFOCOM*, Hong Kong, 2242-2252, 2004.