CHAPTER 3 PROPOSED WORK

In on-path caching, a cache request from a given node goes through a single path toward the gateway. However, a wireless mesh network often has several paths from clients to the gateway. Using only a single path to transfer data would underutilize the network capacity. In this project, we propose the cache request techniques for on-path caching that allow a cache request to be sent to other upstream nodes in addition to the one specified in the routing table. This technique, which can potentially improve the delay and throughput performance, exploits alternate paths toward the gateway which link bandwidth is limited.

Sending a cache request to a randomly selected neighboring upstream proxy would be the simplest choice. An alternative is to broadcast the cache request. In this case, less time is wasted in case of cache miss but the sender has to deal with redundant replies if many of the neighboring proxies turn out to have more cache hits. Also, some neighboring proxies may also be serving many clients, i.e., carrying more loads than the others at that time. Links to those proxies can also have different loss ratios and bandwidths. It is therefore better to avoid proxies having high loads (already connected to many active transfer sessions), and those with lower link loss ratios are preferred.

In this project, two main cache request techniques are proposed that are Random-path cache request and Session-count cache request which will be described next.

3.1 Random-Path Cache Request

The detail of random-path cache request is shown in Figure 3.1. In random-path cache request, when a mesh router receives a file request from one of its attached clients, it checks if the file exists in its cache. If the cache is hit, the mesh router transfers the file back to the client (via TCP). If the cache is missed, the cache request is forwarded to a randomly selected neighboring proxy from some intermediate upstream mesh nodes toward the gateway. The underlying routing protocol can be extended to keep a list of candidate next-hop mesh routers with small additional routing message overheads and storage space. Since a wireless mesh network is generally static, such candidate neighbor list would be constructed only once when routing starts.

When receiving the cache request, the selected intermediate mesh node does exactly the same as the mesh router by checking for the cache hit or miss. If the cache is hit, the intermediate mesh node transfers the file chunk-by-chunk to the requesting node, which replied the file back to the client simultaneously. If the cache is missed, this mesh node randomly selects one of its upstream nodes toward the gateway. The same procedure is repeated until the file requests get to the gateway, in which case the file is retrieved directly from the Internet and transferred to downstream mesh nodes that have previously forwarded the request.

Each intermediate node, when receiving a file in response to the cache request, also caches the whole file if the storage space permits. Otherwise, it attempts to delete files whose status is inactive with the lowest number of past request files, i.e., not being transferred to or from other nodes. If no inactive file exists, the file will not be cached and its packets are just forwarded through it.

Algorithm 1 Random-Path Cache Request

```
1:  BEGIN
2:  if request for file f is received then
3:      if file f in the cache then
4:          ▷ Mark file f as active if it is inactive.
5:          ▷ Transfer file f to the requesting client,
6:            MR/AMR, chunk-by-chunk.
7:          ▷ Wait for an ack for the file receipt.
8:          ▷ Mark file f as inactive.
9:      else
10:         if there exists pending request for file f then
11:             ▷ Wait for the response of the pending
12:               request.
13:         else
14:             ▷ Randomly request file f from one of
15:               the upstream MRs toward the gateway.
16:
17: if response for file f is received then
18:     ▷ Mark file f as active
19:     ▷ Receive and forward file f to the downstream MR
20:     if available storage space ≥ size(f) then
21:         ▷ Store file f in the cache
22:     else
23:         if size(inactive files) ≥ size(f) then
24:             ▷ Purge inactive files (with lowest number of
25:               past requests first)
26:             ▷ Store file f in the cache
27:     ▷ Acknowledge the receipt of file f to the sender
28:       after receiving the whole file completely.
29: END
```

Figure 3.1 Algorithm for Random-path cache request

3.2 Session-Count Cache Request

In the session-count cache request, a node that encounters a cache miss will broadcast a query message to other upstream mesh nodes to ask for a current number of active sessions. Once, all the replies are collected, the node sends/forwards a cache request to an upstream proxy with lowest concurrent active sessions. In case of a tie, it randomly chooses one of them. The detail of Session-count cache request is shown in Figure 3.2

Algorithm 2 Session-Count Cache Request

```
1: BEGIN
2: if status query message is received then
3:     Return the current number of active sessions.
4:
5: if request for file f is received then
6:     if file f in the cache then
7:         ▷ Mark file f as active if it is inactive.
8:         ▷ Transfer file f to the requesting client,
9:           MR/AP, chunk-by-chunk.
10:        ▷ Wait for an ack for the file receipt.
11:        ▷ Mark file f as inactive.
12:    else
13:        if there exists pending request for file f then
14:            ▷ Wait for the response of the pending
15:              request.
16:        else
17:            ▷ Broadcast query message to
18:              the upstream MRs toward the gateway.
19:            ▷ Wait for all the replies
20:            ▷ Select an MR with lowest number of active
21:              sessions to forward a request for file f.
22:
23: if response for file f is received then
24:    ▷ Mark file f as active
25:    ▷ Receive and forward file f to the downstream MR
26:    if available storage space ≥ size(f) then
27:        ▷ Store file f in the cache
28:    else
29:        if size(inactive files) ≥ size(f) then
30:            ▷ Purge inactive files (with lowest number of
31:              past requests first)
32:            ▷ Store file f in the cache
33:    ▷ Acknowledge the receipt of file f to the sender
34:      after receiving the whole file completely.
35: END
```

Figure 3.2 Algorithm for Session-count cache request