Γενική περιγραφή

Στο οικοσύστημα του Internet of Thinks (IoT) υπάρχει μια ευρεία γκάμα από δικτυακές συσκευές. Πολλές από αυτές έχουν ελάχιστο μέγεθος και λειτουργούν με μπαταρίες. Για αυτόν τον λόγο, οι υπολογιστικοί τους πόροι είναι αρκετά περιορισμένοι. Στις περισσότερες περιπτώσεις, η υλοποίηση ολόκληρου του TCP πρωτοκόλλου είναι αδύνατη και η μεταφορά της πληροφορίας γίνεται με UDP. Υπάρχουν όμως εφαρμογές που απαιτούν αξιόπιστη επικοινωνία και μεταφορά δεδομένων.

Ο σκοπός του project είναι η υλοποίηση μιας βιβλιοθήκης η οποία θα παρέχει μια απλή έκδοση του TCP. Παρόλα αυτά, θα είναι ικανή να παρέχει αξιόπιστη επικοινωνία μεταξύ συσκευών με περιορισμένους υπολογιστικούς πόρους χρησιμοποιώντας το διαθέσιμο UDP πρωτόκολλο μεταφοράς.

1. Δομή του project - Προθεσμίες

Το project αποτελείται από **2 φάσεις**.

Στην πρώτη φάση ζητείται να υλοποιήσετε την δομή των microTCP πακέτων, το 3-way handshake (χειραψία) και το finalize (τερματισμό) της microTCP σύνδεσης. Επίσης, θα υλοποιήσετε ενα εργαλείο μεταφοράς αρχείων με TCP το οποίο θα χρησιμοποιήσετε στο τέλος για την σύγκριση της απόδοσης της υλοποίησή σας, τόσο σε σχέση με την πλήρη υλοποίηση του TCP, όσο και με υλοποιήσεις άλλων ομάδων.

Στην δεύτερη φάση θα πρέπει να υλοποιήσετε την βασική λειτουργία του TCP η οποία περιλαμβάνει τους μηχανισμούς για acknowledgements, retransmissions, error checking, TCP windowing, Congestion control και Slow start.

Απαγορεύεται να αλλάξετε τα συμβόλαια των συναρτήσεων που σας έχουν δοθεί και το structure των headers που σας δίνονται. Η αλλαγή τους θα σημαίνει και τον μηδενισμό του project. Παρόλα αυτά είναι επιθυμητές οι αλλαγές/προσθήκες στα υπόλοιπα structs.

2. Ζητούμενα Β' Φάσης

2.1 Συναρτήσεις μεταφοράς δεδομένων

Σε αυτή την φάση, θα πρέπει να υλοποιήσετε τις συναρτήσεις *microtcp_send()* και *microtcp_recv()* οι οποίες παρέχουν παρόμοια λειτουργικότητα με τις αντίστοιχες *send()* και *recv()* του TCP.

- ```
  ssize_t
  microtcp_send(microtcp_sock_t *socket,
                const void *buffer,
                size_t length, int flags);
  ```

  Η συνάρτηση αυτή είναι υπεύθυνη για την αξιόπιστη αποστολή δεδομένων. Οι παράμετροι είναι οι εξής:

  - socket: Ο pointer του socket που διαχειρίζεται την microTCP σύνδεση. Είναι pointer καθώς θα χρειαστεί να ανανεώσετε διάφορα πεδία που περιγράφουν την κατάσταση της σύνδεσης.
  - buffer: Τα δεδομένα που θέλει ο χρήστης να στείλει.
  - length: Ο αριθμός των bytes που θέλει να στείλει ο χρήστης.
  - flags: Flags που τυχόν επιθυμεί ο χρήστης να περάσει στο socket.

  Η συνάρτηση θα πρέπει να επιστρέφει **τον αριθμό των bytes που επιτυχημένα και επιβεβαιωμένα** έστειλε στον παραλήπτη. Σε περίπτωση οποιουδήποτε λάθους θα πρέπει να επιστρέφει -1.

- ```
  ssize_t
  microtcp_recv(microtcp_sock_t *socket,
                void *buffer,
                size_t length, int flags);
  ```

  Η συνάρτηση αυτή είναι υπεύθυνη για την αξιόπιστη λήψη δεδομένων. Οι παράμετροι είναι οι εξής:

  - socket: Ο pointer του socket που διαχειρίζεται την microTCP σύνδεση. Είναι pointer καθώς θα χρειαστεί να ανανεώσετε διάφορα πεδία που περιγράφουν την κατάσταση της σύνδεσης.
  - buffer: Θέση μνήμης στην οποία τα δεδομένα από το δίκτυο θα αποθηκευτούν. Είναι ευθύνη του χρήστη, να παρέχει αρκετό χώρο.
  - length: Ο αριθμός των bytes που θέλει να λάβει ο χρήστης.
  - flags: Flags που τυχόν επιθυμεί ο χρήστης να περάσει στο socket.

  Η συνάρτηση θα πρέπει να επιστρέφει **τον αριθμό των bytes που επιτυχημένα λήφθηκαν**. Σε περίπτωση οποιουδήποτε λάθους θα πρέπει να επιστρέφει -1. Επιπλέον, καθώς η *microtcp_recv()* είναι μια κλήση που μπορεί να μπλοκάρει, υπάρχει η πιθανότητα να λάβει ένα μήνυμα τερματισμού της σύνδεσης (FIN, ACK). Σε αυτή την περίπτωση, θα πρέπει να επιστρέφει -1 και να θέτει την κατάσταση της microTCP σύνδεσης ως *CLOSING_BY_PEER*. Την πληροφορία αυτή την εκμεταλλεύεται η microtcp_shutdown() για να συνεχίσει κατάλληλα τον τερματισμό της σύνδεσης.

3. TCP Λειτουργίες

Το microTCP θα πρέπει να παρέχει την λειτουργικότητα του TCP χρησιμοποιώντας κάποιους από τους βασικούς του μηχανισμούς.

3.1 Error checking

Αξιοποιώντας το πεδίο **checksum** του *microtcp_header_t* header θα πρέπει να γίνεται ο έλεγχος αν το πακέτο λήφθηκε σωστά. Σε περίπτωση που το checksum δεν είναι σωστό, το πακέτο πρέπει να θεωρείται κατεστραμμένο και μπορείτε να θεωρήσετε ότι δεν λήφθηκε ποτέ, ενεργοποιώντας τους κατάλληλους μηχανισμούς.

3.2 Λήψη πακέτων με την σωστή σειρά

Στόχος του microTCP είναι η σωστή και ασφαλής μεταφορά δεδομένων. Για αυτό τον λόγο θα πρέπει να εξασφαλίζεται η λήψη των πακέτων με σωστή σειρά. Κάθε πακέτο περιέχει ένα sequence number. Χρησιμοποιώντας αυτό τον αριθμό κατάλληλα, μπορείτε να αποφασίσετε αν ένα πακέτο λήφθηκε ή όχι με την σωστή σειρά.

Η σωστή σειρά πακέτων θα πρέπει να ελέγχεται καθ' όλη την διάρκεια της σύνδεσης, δηλαδή από το 3-way handshake έως και τον τερματισμό της σύνδεσης και για όλα τα πακέτα που ανταλλάσσονται.

3.3 Retransmissions

Για κάθε πακέτο που αποστέλλεται θα πρέπει να ληφθεί το αντίστοιχο ACK. Σε περίπτωση που το ACK δεν ληφθεί σε *MICROTCP_ACK_TIMEOUT_US* microsecons (us) (ορίζεται στο microtcp.h), ο αποστολέας πρέπει να ξανα-στείλει το πακέτο.

Το πρόβλημα όμως είναι πως η *recvfrom()* που χρησιμοποιείται εσωτερικά είναι μια συνάρτηση που μπλοκάρει μέχρι να λάβει κάποιο πακέτο. Μπορείτε παρόλα αυτά να της θέσετε ένα timeout χρησιμοποιώντας την *setsockopt()* και το όρισμα *SO_RCVTIMEO*. Με αυτό τον τρόπο, αν η *recvfrom()* δεν λάβει κάποιο πακέτο μέσα στον χρόνο που ορίσατε, επιστρέφει έναν αρνητικό αριθμό. Τον timeout χρόνο μπορείτε να τον ορίσετε με τον παρακάτω τρόπο:

```
struct timeval timeout;
timeout.tv_sec = 0;
timeout.tv_usec = MICROTCP_ACK_TIMEOUT_US;
if (setsockopt(receive_socket, SOL_SOCKET,
               SO_RCVTIMEO, &timeout,
```

```
                sizeof(struct timeval)) < 0) {
    perror("setsockopt");
}
```

Retransmission θα πρέπει να γίνει επίσης στην περίπτωση που ο αποστολέας λάβει 3 συνεχόμενα duplicate ACK. Τι ακριβώς είναι το duplicate ACK περιγράφεται παρακάτω.

3.4 Duplicate Acknowledgements και Fast Retransmit

Τα duplicate ACKs είναι ένας ιδιαίτερα σημαντικός μηχανισμός του TCP με την βοήθεια του οποίου ο αποστολέας μαθαίνει άμεσα την κατάσταση του πακέτου που έστειλε.

Αν το πακέτο λήφθηκε με λάθη, ο παραλήπτης στέλνει πίσω ένα ACK που είναι το ίδιο με το ACK του τελευταίου πακέτου που λήφθηκε σωστά. Ο αποστολέας είναι σε θέση εύκολα να καταλάβει ότι το συγκεκριμένο ACK αφορά προηγούμενο πακέτο, οπότε αναγνωρίζει πως το πακέτο δεν έφτασε σωστά και ενεργοποιεί τον retransmission μηχανισμό. Επίσης duplicate ACK στέλνεται στην περίπτωση που ο παραλήπτης λάβει ένα πακέτο με λάθος sequence number.

Χωρίς το duplicate ACK, ο αποστολέας θα έπρεπε να περιμένει μέχρι να κάνει timeout προτού κάνει retransmission. Κάτι τέτοιο όμως θα μείωνε δραματικά την απόδοση, επειδή το timeout συμβαίνει σχετικά μετά από ένα μεγάλο διάστημα. Λαμβάνοντας λοιπόν ο receiver 3 duplicate ACKs, ξεκινάει αμέσως το retransmission. ο μηχανισμός αυτός είναι γνωστός ως **Fast Retransmit**, καθώς αντιδράει άμεσα σε μια πιθανή απώλεια δεδομένων.

Ένα ερώτημα που προκύπτει είναι ποιο πακέτο θα πρέπει να ξαναστείλει ο αποστολέας. Το duplicate ACK περιέχει όμως τον αριθμό των bytes που έχουν ληφθεί σωστά από τον παραλήπτη. Επομένως είναι εύκολο να βρεθεί ποια δεδομένα θα πρέπει να ξανασταλθούν.

3.5 Flow Control

Το TCP εφαρμόζει end-to-end έλεγχο ροής για να αποφύγει απώλεια πακέτων. Για παράδειγμα αν ένας host στέλνει σε ένα κινητό πακέτα πολύ γρήγορα, το κινητό με τις περιορισμένες υπολογιστικές δυνατότητες τα επεξεργάζεται πιο αργά. Τελικά οι buffers του κινητού θα γεμίσουν και πακέτα μπορεί να χαθούν. Σκοπός του flow control είναι να περιοριστεί η ταχύτητα αποστολής ανάλογα με τις δυνατότητες του παραλήπτη.

Το TCP το επιτυγχάνει χρησιμοποιώντας τον αλγόριθμο του sliding window. Αρχικά κατά το 3-way handshake ανταλλάσσεται το αρχικό window size. Για την

περίπτωση του microTCP, αυτό ορίζεται με την σταθερά *MICROTCP_WIN_SIZE*. Μόλις εγκαθιδρυθεί η σύνδεση και οι δύο πλευρές δεσμεύουν μνήμη για τον receive buffer τους. Το μέγεθος του buffer θα πρέπει να είναι μεγαλύτερο ή ίσο του window. Για την υλοποίησή μας το μέγεθος είναι ίσο με *MICROTCP_RECVBUF_LEN*.

Το window αναφέρεται στον αριθμό των bytes που είναι σε θέση να δεχθεί ο παραλήπτης. Όσο λαμβάνει πακέτα ο διαθέσιμος χώρος μικραίνει, άρα και το window. Όταν τα δεδομένα, προωθηθούν προς τον χρήστη ο διαθέσιμος χώρος αυξάνεται ξανά, άρα το window μεγαλώνει. Η κατάσταση του window αποθηκεύεται στο *microtcp_sock_t*. Το *curr_win_size* αναφέρεται στην τρέχουσα τιμή του window, ενώ το *init_win_size* στην αρχική τιμή που συμφωνήθηκε κατά το handshake.

Ο αλγόριθμος δουλεύει ως εξής:

- Κατά την αποστολή του πρώτου πακέτου ο αποστολέας στέλνει ένα πακέτο μεγέθους X bytes. Το X δεν μπορεί να ξεπεράσει σε μέγεθος το Maximum Segment Size (MSS). Αν και υπάρχουν μηχανισμοί για τον αυτόματο εντοπισμό του MSS, το microTCP χρησιμοποιεί σταθερό MSS του οποίου το μέγεθος ορίζεται από την σταθερά *MICROTCP_MSS*.
- Ο παραλήπτης δέχεται το πακέτο και το αποθηκεύει στον receive buffer. Καθώς στέλνει πίσω το ACK, ενημερώνει κατάλληλα τον αποστολέα για το πόσα bytes είναι πρόθυμος να δεχθεί με το επόμενο πακέτο, τοποθετώντας την τιμή $window - X$ στο αντίστοιχο πεδίο του header.
- Ο αποστολέας μπορεί να στείλει το πολύ όσα bytes αναφέρονται στο πεδίο window του ACK header.
- Προωθώντας bytes στον χρήστη, ο παραλήπτης αυξάνει το window του κατά Y.
- Υπάρχει πιθανότητα το window κάποια στιγμή να γίνει 0. Σε μια τέτοια περίπτωση ο αποστολέας θα πρέπει να στέλνει επανειλημμένα ένα πακέτο χωρίς payload έως ότου πάρει ACK με μη-μηδενικό window. Πριν την αποστολή αυτού του ειδικού σκοπού πακέτου, περιμένει random χρόνο μεταξύ 0 και *MICROTCP_ACK_TIMEOUT_US* microsecons. Μόλις λάβει ACK με μη-μηδενικό window size συνεχίζει την αποστολή.

3.6 Congestion Control

Στο flow control μπορεί ο αποστολέας να συμμετέχει ενεργά αυξομειώνοντας τον όγκο των δεδομένων που στέλνει, όμως ο παραλήπτης είναι αυτός που καθορίζει τελικά τον αριθμό των bytes που θα στείλει ο αποστολέας βάση την πληρότητα των receive buffer του. Για αυτό συνήθως θεωρείται πως το flow control υλοποιείται στον παραλήπτη.

Αντίθετα το congestion control υλοποιείται στον αποστολέα, αντλώντας πληροφορία για την κατάσταση του δικτύου κυρίως από τα ACKs. Στόχος του είναι να αποφύγει την συμφόρηση στο δίκτυο εμποδίζοντας δυναμικά την ανεξέλεγκτη αποστολή δεδομένων, κρατώντας όμως την απόδοση σε υψηλά επίπεδα. Το congestion control περιλαμβάνει τους αλγόριθμους slow start, congestion avoidance, fast retransmit και fast recovery. Στο microTCP θα υλοποιήσετε τους μηχανισμούς slow start, congestion avoidance και τον προαναφερθέν fast retransmit.

Παρόλο που το congestion avoidance και το slow start έχουν εντελώς διαφορετικό στόχο, εντούτοις όταν μια TCP σύνδεση αντιμετωπίσει συμφόρηση στο δίκτυο, θα πρέπει να περιοριστεί η ταχύτητα αποστολής δεδομένων και αργότερα να αποκατασταθεί. Για αυτό τον λόγο συνήθως οι αλγόριθμοι congestion avoidance και slow start υλοποιούνται μαζί.

Για την υλοποίησή τους χρειάζονται δύο μεταβλητές για κάθε σύνδεση:

- **cwnd:** Congestion window. Ο αριθμός των bytes που μπορεί να στείλει ο αποστολέας, χωρίς να περιμένει για τα αντίστοιχα ACKs.
- **ssthresh:** Slow start threshold. Το όριο αυτό ορίζει αν θα χρησιμοποιηθεί ο αλγόριθμος slow start ή congestion avoidance. Αν $cwnd \leq ssthresh$ τότε χρησιμοποιείται ο slow start αλγόριθμος. Διαφορετικά ο congestion avoidance.

Αρχικά το cwnd τίθεται ίσο με $3 \times MSS$ χρησιμοποιώντας την σταθερά *MICROTCP_INIT_CWND*. Αντίστοιχα το αρχικό ssthresh είναι ίσο με το window του flow control και ορίζεται με τη σταθερά *MICROTCP_INIT_SSTHRESH*.

3.6.1 Slow start

Κατά την διάρκεια του slow start, για κάθε σωστό ACK που λαμβάνεται, το congestion window αυξάνεται κατά MSS bytes. Αυτό σημαίνει ότι για κάθε RTT (x πακέτα στάλθηκαν - x ACKs λήφθηκαν) το congestion window διπλασιάζεται.

3.6.2 Congestion avoidance

Κατά την διάρκεια του congestion avoidance, για κάθε RTT (x πακέτα στάλθηκαν - x ACKs λήφθηκαν) το congestion window αυξάνεται κατά MSS bytes. Το congestion window θα αυξάνεται μέχρι να αρχίσουν πακέτα να χάνονται. Αν ληφθούν 3 duplicate ACKs τότε θα πρέπει να γίνουν οι παρακάτω αλλαγές:

```
ssthresh = cwnd/2;
cwnd = cwnd/2 + 1;
```

Στην περίπτωση που περιμένοντας για ACK συμβεί timeout, τότε:

```
ssthresh = cwnd/2;
cwnd = min(MICROTCP_MSS, ssthresh);
```

και ενεργοποιείται ο slow start αλγόριθμος.

4. Λεπτομέρειες υλοποίησης

Αρχικά θα πρέπει να κατεβάσετε τον ανανεωμένο κώδικα από το Github repository https://github.com/surligas/microTCP. Συστήνεται ιδιαίτερα να δείτε τις αλλαγές στο *microtcp_sock_t* καθώς και σε άλλα σημεία του κώδικα.

Παρακάτω ακολουθεί ένα roadmap για την πιο εύκολη και σταδιακή υλοποίηση της φάσης αυτής.

Roadmap:

1. Κάντε τις απαραίτητες αλλαγές στο 3-way handshake. Οι δύο host θα πρέπει να ανταλλάσσουν το αρχικό flow control window μέσω του πεδίου *window* του header.
2. Τροποποιείστε την *microtcp_shutdown()* κατάλληλα, ώστε να παίρνει υπόψιν αν το FIN λήφθηκε ήδη από μια προηγούμενη κλήση της *microtcp_recv()*.
3. Ξεκινήστε να υλοποιείτε τις *microtcp_recv()* και *microtcp_send()*. Σε αυτό το σημείο οι συναρτήσεις αυτές απλά θα στέλνουν και θα λαμβάνουν δεδομένα, πραγματοποιώντας μόνο τυπικούς ελέγχους.
4. Προσθέστε στις *microtcp_recv()* και *microtcp_send()* το error checking και τον έλεγχο των sequence και ack numbers για την διασφάλιση της λήψης πακέτων με σωστή σειρά. Σε οποιαδήποτε περίπτωση λάθους θα πρέπει να στέλνεται ένα duplicate ACK.
5. Προσθέστε στην *microtcp_send()* των έλεγχο για το duplicate ACK. Αν το ACK που λήφθηκε ήταν duplicate ACK, η *microtcp_send()* θα πρέπει να στέλνει ξανά το τελευταίο πακέτο. Αν καθώς περιμένει για ACK γίνει κάποιο timeout, ξαναστέλνει το τελευταίο πακέτο.
6. Υλοποιήστε στις *microtcp_recv()* και *microtcp_send()* τον flow control μηχανισμό.
7. Υλοποιήστε στην *microtcp_send()* τον congestion control μηχανισμό.
8. Προσαρμόστε το *bandwidth_test* εργαλείο στη ανάγκες του microTCP. Πλέον ο κώδικας για την TCP υλοποίηση σας δίνεται έτοιμος και μεταφέρει το αρχείο σωστά στην άλλη πλευρά. **ΠΡΟΣΟΧΗ:** Τα μεγέθη των buffers που επιθυμεί να στείλει ο client θα πρέπει να παραμείνουν ίδια. Επίσης θα πρέπει να εκτυπώνετε το αποτέλεσμα με ακριβώς τον ίδιο τρόπο στην οθόνη.

9. **ΚΑΛΕΣ ΔΙΑΚΟΠΕΣ!**

4.1 Διαδικασία αποστολής πακέτων

Κάθε UDP πακέτο που αποστέλλεται με την *microtcp_send()* θα πρέπει να έχει μέγεθος μικρότερο ή ίσο με το MSS. Αυτό γίνεται για να αποφευχθεί το IP fragmentation. Επομένως ο buffer του χρήστη, θα πρέπει να κατακερματιστεί σε πολλαπλά chunks των MSS bytes. Επιπρόσθετα, πριν την αποστολή των πακέτων θα πρέπει να ελέγχεται και ο επιτρεπτός αριθμός bytes που μπορούν να σταλθούν βάσει των flow και congestion control μηχανισμών. Γενικά ο αποστολέας μπορεί να στείλει το πολύ *min(window, cwnd)* bytes κάθε φορά.

```
ssize_t
microtcp_send(microtcp_sock_t *socket,
              const void *buffer,
              size_t length, int flags)
{
  .
  .
  .
  remaining = length;
  while(data_sent < length){
    bytes_to_send = min(flow_ctrl_win, cwnd, remaining);
    chunks = bytes_to_send / MICROTCP_MSS;
    for(i = 0; i < chunks; i++){
      sendto(....);
    }
    /* Check if there is a semi-filled chunk */
    if(bytes_to_send % MICROTCP_MSS){
      chunks++;
      sendto(....);
    }

    /* Get the ACKs */
    for(i = 0; i < chunks; i++){
      recvfrom(...);
    }

    /* Retransmissions */
    /* Update window */
    /* Update congestion control */

    remaining -= bytes_to_send;
```

```
    data_sent += bytes_to_send;
    .
    .
    .
}
```