Implementing Fifo Channel using lossy|LRD channel

Shankar

September 18, 2014
Data transfer protocol: DtpDist

- Implements $\text{FifoChannel}(a_1,a_2)$ using lossy/lrd channel

- Sliding window protocol (Swp)
  - $a_1 \rightarrow a_2$ fifo transfer
  - define programs at "algorithm-level": atomic rules
  - prove correctness with modulo-$N$ sequence numbers
  - correctness-preserving refinement to await program

- Obtain DtpDist via correctness-preserving merge of two SwpDist
Outline

Sliding Window Protocol

Analysis of Sliding Window Protocol

Await-structured Source and Sink Programs

Data transfer Protocol and Proof

Graceful-closing Data transfer Protocol

Abortable Data transfer Protocol
Sliding window protocol: SwpDist

\[
\text{SwpDist}(a_1, a_2, \ldots) \\
[x_{a_1}, x_{a_2}] \leftarrow \text{startSystem(} \text{Lossy/LrdChannel}(a_1, a_2)\text{)}; \\
y_{a_1} \leftarrow \text{startSystem(} \text{Source}(a_1, a_2, x_{a_1}, \ldots)\text{)}; \\
y_{a_2} \leftarrow \text{startSystem(} \text{Sink}(a_2, a_1, x_{a_2}, \ldots)\text{)}; \\
\text{return } [y_{a_1}, y_{a_2}]
\]
Solution using sequence numbers

- \( d_0, d_1, \ldots \): data blocks sent into source by its user

- Source
  - \( \text{tx } [d_k, k] \) repeated until acked \(/ / k: \text{seq } \#\)

- Sink
  - respond with \text{awaited seq } \# \hspace{1cm} (/ / \text{cumulative ack})
  - has 0 \ldots 4 \sqcup 6, \ldots 8: \hspace{1cm} \text{rx 7 / tx 5; } \text{rx 5 / tx 9}
  - passes data blocks to its user in order

- For good throughput
  - > 1 \text{outstanding} at source, buffer \text{out-of-sequence} at sink

- Above requires \text{unbounded seq } \#s \hspace{1cm} (\text{usn})
  - not good for hw implementation

- Instead use \text{cyclic seq } \#s \hspace{1cm} (\text{csn})
Using modulo-\(N\) cyclic sequence numbers

- Use mod-\(N\) csn instead of usn
- Send csn mod(\(k, N\)) instead of usn \(k\)
- Receiver of \(csn\) has to infer the corresponding \(usn\)
  - maintains window of “possible” usn
    - say \(L \cdots U\)
  - maps rcvd \(csn\) to \(usn\) with same cyclic value
    - \(usn \leftarrow L + \text{mod}(csn - L, N)\);
      if \(usn > U\) ignore rcvd \(csn\)
- Seq #s in transit must remain close to window
Sliding window protocol with mod-$N$ csn

Source
- ng: 　# blks from user
- ns: 　# blks sent at least once
- na: 　# blks acked
- map sbuff:  for blks na ··· ng−1
- send window:  na ··· ns+SW−1  // SW < N
- outstanding window:  na ··· ns  // ok if low end is na+1

Sink
- nd: 　# blks to user
- nr: 　# contiguous blks rcvd
- map rbuff:  for blks nd ··· nd+RW−1  // RW < N
- receive window:  nr ··· nd+RW−1

Over time, windows slide to increasing seq #s
Sliding windows

vars
- ng: # blks from user
- ns: # sent
- na: # acked
- sbuff

Source

Sink

- acknowledged
- delivered
- rbuff
- received
- possibly received
- nd: # blks to user
- nr: blk next awaited
- nd+RW-1

not yet sent
outstanding
na+SW-1
send window
receive window

nr

nr (not received)
Swp mod-N: algorithm-level

Source

na, ns, ng ← 0; sbuff ← []
- db from user
  sbuffng ← db; ng++
- ns < min(na+SW, ng)
  send [sbuffns, ns]; ns++
- k in na..ns−1
  send [sbuffk, k]
- rcv [cn]
  j ← na + cn−na;
  if (na < j ≤ ns)
    sbuff.remove(na..j−1);
  na ← j;

Sink

nd, nr ← 0; rbuff ← []
- nd < nr
  rbuffnd to user; nd++
- rcv [db, cn]
  j ← nr + cn−nr;
  if (nr ≤ j < nd+RW)
    rbuffj ← db;
    while (nr in rbuff.keys)
      nr ← nr+1;
    send [nr]

\( k \text{: mod}(k,N) \)
Outline

Sliding Window Protocol
Analysis of Sliding Window Protocol
Await-structured Source and Sink Programs
Data transfer Protocol and Proof
Graceful-closing Data transfer Protocol
Abortable Data transfer Protocol
Desired properties of SwpDist

- **dbh**: auxiliary var indicating seq of blks sent by user

Desired properties

\[ X_1 : Inv \ (nd < nr \ \Rightarrow \ \text{rbuff}_{nd} = \text{dbh}_{nd}) \]

\[ X_2 : (nd = k < ng \ \text{leads-to} \ nd > k) \]

assuming wfair send \( \text{sbuff}_{ns} \), resend \( \text{sbuff}_{na} \),
deliver \( \text{rbuff}_{nd} \) to user,
source/sink rcv msg,
channel progress

Intermediate desired \( Inv \ A_1 \rightarrow A_3 \)

\[ A_0 : nd \cdots nr-1 \ \text{in} \ \text{rbuff.keys} \]

\[ A_1 : (k \ \text{in} \ \text{rbuff.keys}) \ \Rightarrow \ \text{rbuff}_k = \text{dbh}_k \quad \text{//} \ A_{0,1} \Rightarrow X_1.p \]

\[ A_2 : na \leq nr \leq ns \leq na+SW \quad \text{and} \quad ns \leq ng \]
Correct interpretation conditions

- Add auxiliary usn field to msgs // not read
  - source sends \([sbuff_j, j, j]\)
  - sink sends \([\bar{nr}, nr]\)

- Sink maps rcvd \([\bar{j}, j]\) wrt rcv window \(nr \ldots nr+\text{RW}-1\)
  - \(j\) in window: mapped correctly
  - \(j < \text{window}: nr-1 \checkmark, nr-2 \checkmark, \ldots \checkmark, nr-N+\text{RW}-1 \times\)
  - \(j > \text{window}: nr+\text{RW} \checkmark, nr+\text{RW}+1 \checkmark, \ldots \checkmark, nr+N \times\)
  - desired \(\text{Inv } A_3\)
    \[
    A_3 : \text{data } j \text{ rcvable} \Rightarrow j \text{ in } nr-N+\text{RW} \ldots nr+N-1
    \]

- Source maps rcvd \([\bar{j}, j]\) wrt outstanding window \(na \ldots ns\)
  - desired \(\text{Inv } A_3\)
    \[
    A_4 : \text{ack } j \text{ rcvable} \Rightarrow j \text{ in } ns-N+1 \ldots na+N
    \]
Goal

\( A_0: \text{nd} \cdots \text{nr}-1 \in \text{rbuff.keys} \)

\( A_1: (k \in \text{rbuff.keys}) \Rightarrow \text{rbuff}_k = \text{dbh}_k \)

\( A_2: \text{na} \leq \text{nr} \leq \text{ns} \leq \text{na} + \text{SW} \)

Correct interpretation

\( A_3: \text{data j rcvable} \Rightarrow j \in \text{nr}-\text{N}+\text{RW} \cdots \text{nr}+\text{N}-1 \)

\( A_4: \text{ack j rcvable} \Rightarrow j \in \text{ns}-\text{N}+1 \cdots \text{na}+\text{N} \)

For every step \( e: \{A_0 - A_4\} e \{A_0, A_1, A_2\} \) holds

Suffices to establish

\( \{A_2, A_3, A_4\} e \{A_3\} \) for \( e: \) send data \( k; \) increase \( \text{nr} \)

\( \{A_2, A_3, A_4\} e \{A_3\} \) for \( e: \) send ack; increase \( \text{ns} \)
Outline

**Sliding Window Protocol**
Analysis of Sliding Window Protocol
  - Inv $A_3 - A_4$ for lossy channel
  - Inv $A_3 - A_4$ for LRD channel
  - Progress for lossy/LRD channel

**Await-structured Source and Sink Programs**

**Data transfer Protocol and Proof**

**Graceful-closing Data transfer Protocol**

**Abortable Data transfer Protocol**
For SwpDist with lossy channel

now show that \textit{Inv }$A_3$–$A_4$ holds if $N \geq SW + RW$
Preserving $A_3$ wrt send data

$A_2: \text{na} \leq \text{nr} \leq \text{ns} \leq \text{na}+\text{SW}$

$A_3: \text{data j rcvable} \Rightarrow j \text{ in } \text{nr}−N+\text{RW} \ldots \text{nr}+N−1$

- **Send data k preserves $A_3$.rhs lower bound** if $N \geq \text{SW}+\text{RW}$
  - $k \geq \text{na}$
    - $\geq \text{ns}−\text{SW}$
    - $\geq \text{nr}−\text{SW}$
    - $\geq \text{nr}−N+\text{RW}$

- **Send data k preserves $A_3$.rhs upper bound**
  - $k \leq \text{na}+\text{SW}−1$
    - $\leq \text{nr}+\text{SW}−1$
    - $\leq \text{nr}+N−1$
Preserving $A_3$ wrt $nr$ increase

$A_2 : na \leq nr \leq ns \leq na+SW$

$A_3 : \text{data j rcvable } \Rightarrow j \in nr-N+RW \ldots nr+N-1$

- $nr$ increase preserves $A_3$ if $N \geq SW+RW$
  - let $nr$ become $k$ at time $t_0$.  
  - so $k-1$ rcvd at $t_0$ or prior, so sent at some $t_1$  
    // $t_1 < t_0$
    - $ns(t_1) \geq k+1$  
      // guards
    - $na(t_1) \geq k+1-SW$  
      // $A_2$
  - let $j$ be rcvable after $t_0$, so sent at some $t_2$ after $t_1$
    - $j \geq na(t_2) \geq na(t_1)$  
      // na non-decreasing
    - $\geq k+1-SW$  
      // $\ast\ast$
    - $\geq k+1-N+RW$  
      // $N \geq SW+RW$
Preserving $A_4$

- $A_2: \text{na} \leq \text{nr} \leq \text{ns} \leq \text{na}+\text{SW}$
- $A_4: \text{ack } j \text{ rcvable } \Rightarrow j \text{ in ns-}N+1 \cdots \text{na}+N$

**Proof of Inv $A_4$**

1. acks sent have non-decreasing usn // nr non-decreasing
2. acks in transit have non-decreasing usn // no reordering
3. ack usn lower bounded by na // na is an ack usn
4. Inv (ack $j$ rcvable $\Rightarrow$ na $\leq$ j $\leq$ nr) // 1, 2, 3
5. Inv $A_4$ // 4, $A_2$
Outline

**LRD channel**

- Sliding Window Protocol
  - Analysis of Sliding Window Protocol
    - Inv $A_3 - A_4$ for lossy channel
    - Inv $A_3 - A_4$ for LRD channel
    - Progress for lossy/LRD channel

**swp analysis**

- Await-structured Source and Sink Programs
- Data transfer Protocol and Proof
- Graceful-closing Data transfer Protocol
- Abortable Data transfer Protocol
For SwpDist with LRD channel

- Obviously $\text{Inv } A_3 - A_4$ does not hold for any $N$

But $\text{Inv } A_3 - A_4$ holds if

- LRD channel has max msg lifetime $L$
- Min time $\delta$ between $ns$ increments
- $N \geq SW + RW + \frac{L}{\delta}$
Preserving $A_3$

$A_2$ : $na \leq nr \leq ns \leq na+SW$

$A_3$ : data $j$ rcvable $\Rightarrow$ $j$ in $nr-N+RW \cdots nr+N-1$

- $A_3$.rhs upper bound holds exactly as in lossy channel case

- $A_3$.rhs lower bound holds as follows

  1. let data $j$ be rcvable at $t_0$
  2. data $j$ was sent at some $t_1 > t_0 - L$ // 1
  3. $j \geq na(t_1) \geq ns(t_1) - SW$ // guards, $A_2$
  4. during $[t_1, t_0]$ : ns increases by at most $L/\delta$ // guards
  5. $j \geq ns(t_0) - L/\delta - SW$ // 3, 4
  6. $j \geq ns(t_0) - N + RW$ // 5, $N \geq SW + RW + L/\delta$
  7. $j \geq nr(t_0) - N + RW$ // 6, $A_2$
Preserving $A_4$

$A_2 : \text{na} \leq \text{nr} \leq \text{ns} \leq \text{na} + \text{SW}$

$A_4 : \text{ack } j \text{ rcvable } \Rightarrow j \text{ in } \text{ns} - N + 1 \cdots \text{na} + N$

- $A_4$.rhs upper bound holds exactly as in lossy channel case
- $A_4$.rhs lower bound holds as follows

1. let $\text{ack } j$ be rcvable at $t_0$
2. $\text{ack } j$ was sent at some $t_1 > t_0 - L$ // 1
3. $j = \text{nr}(t_1) \geq \text{na}(t_1) \geq \text{ns}(t_1) - \text{SW}$ // 2, guard, $A_2$
4. during $[t_1, t_0]$: $\text{ns}$ increases by at most $L/\delta$
5. $j \geq \text{ns}(t_0) - L/\delta - \text{SW}$ // 3, 4
6. $j \geq \text{ns}(t_0) - N + 1$ // 5, $N \geq \text{SW} + \text{RW} + L/\delta$, $\text{RW} > 0$
Outline

Sliding Window Protocol
Analysis of Sliding Window Protocol
  Inv $A_3 - A_4$ for lossy channel
  Inv $A_3 - A_4$ for LRD channel
  Progress for lossy/LRD channel

Await-structured Source and Sink Programs
Data transfer Protocol and Proof
Graceful-closing Data transfer Protocol
Abortable Data transfer Protocol
$X_2 : \text{(nd} = k \text{ < ng leads-to nd} > k\text{)}$

assuming
- wfair send (of sbuff$_{ns}$)
- wfair resend (of sbuff$_{na}$)
- wfair sink user rx (of rbuff$_{nd}$)
- wfair source rx (of msg from lossy/lrd channel)
- wfair sink rx (of msg from lossy/lrd channel)
- lossy/lrd channel progress
\( P_1 : \) \( na = k < nr \leq ns \) leads-to \( na > k \)

// src resend, snk rx, src rx, channel progress

\( P_2 : \) (\( na = k = nr < ns \) and \( nr < nd+RW \)) leads-to \( nr > k \)

// src resend, snk rx, channel progress

\( P_3 : \) (\( na = k = nr < ns \) and \( nr = nd+RW \)) leads-to \( nr > k \)

// sink user rx, \( P_2 \)

\( P_4 : \) \( na = k = nr < ns \) leads-to \( nr > k \)

// \( P_2, P_3 \)

\( P_5 : \) \( na = k < ns \) leads-to \( na > k \)

// \( P_4, P_1 \)

\( P_6 : \) \( nr = k < ns \) leads-to \( nr > k \)

// \( P_5, \text{Inv } nr \geq na \)
Progress: user-level ↔ windows

\[ P_7 : (ns = k < ng \text{ and } ns < na+SW) \ leads-to \ ns > k \]

// src send

\[ P_8 : (ns = k < ng \text{ and } ns = na+SW) \ leads-to \ (ns = k < ng \text{ and } ns < na+SW) \]

// \( P_5 \)

\[ P_9 : ns = k < ng \ leads-to \ ns > k \]

// \( P_7, P_8 \)

\[ P_{10} : nr = k < ng \ leads-to \ nr > k \]

// \( P_9, P_6 \)

\[ P_{11} : nd = k < ng \ leads-to \ nd > k \]

// \( P_{10}, \text{sink user rx} \)
Outline

Sliding Window Protocol
Analysis of Sliding Window Protocol
Await-structured Source and Sink Programs
Data transfer Protocol and Proof
Graceful-closing Data transfer Protocol
Abortable Data transfer Protocol
Given: algorithm-level program $A$
- init, vars, atomic rules, fairness for rules

Goal: program $B$ that implements $A$
- init, vars, threads, await statements, fairness for threads

Construct $B$ as follows
- include $A$-vars
- each $A$-rule $\rightarrow$ await statement $//$ preserves
- additional $B$-steps do not affect $A$-vars $//$ safety
- allocate local/guest threads to awaits $//$ preserves
- fairness of $A$-rules $\rightarrow$ fairness of threads $//$ progress
program Source(aL, aR, xL, N, SW, RW)

parameters
  aL: local addr, aR: remote addr, xL: channel access sid

constants:  DAT, ACK, RTO // data msg, ack msg, timeout

functions
  input mysid.tx: get db from user
  doTxDat: send ns, resend k // calls xL.tx
  doRxAck: rcv ack msg // calls xL.rx

main
  ng, ns, na ← 0, sbuff ← []
  timer ← OFF // OFF, 0, 1, ...
  tSrcTx ← startThread(doTxDat())
  tSrcRx ← startThread(doRxAck())
  return mysid
```javascript
input mysid.tx(aR, db)
    await (true)
    sbuffng ← db;
    ng++;
    return;

function doTxDat()
    while (true)
        • await ((timer = OFF and ns < min(ng, na+SW)) or
                    (timer > RTO and na < ns))
        ns ← min(ng, na+SW);
        for (j in na..ns−1)
            xL.tx(aR, [DAT, sbuffj, mod(j,N)]);
        timer ← 0;                     // (re)start timer
```
function void doRxAck()  
  while (true)  
    • Seq msg ← xL.rx();  
      ia {msg in Tuple<ACK, 0..N−1>}  
      await (true)  
        int j ← na + mod(msg[1]−na, N);  
        if (na < j ≤ ns)  
          for (k in na..j−1) sbuff.remove(k);  
          na ← j;  
        if (na = ns)  
          timer ← OFF; // stop timer

■ atomicity assumption: awaits

■ progress assumption: weak fairness of threads
- Send new data blocks asap, instead of upon timeout
- Reduce extent of blocking at awaits
  - concurrent access to sbuff
  - duplicate counters
  - ...
- Adapt RT0 to measured roundtrip time // congestion control
program Sink(aL, aR, xL, N, SW, RW)

parameters
- aL: local addr, aR: remote addr, xL: channel access sid

constants: DAT, ACK // data msg, ack msg

functions
- input mysid.rx: deliver db to user
- doRxDatTxAck: rcv data msg, send ack // calls xL.rx, xL.tx

main
- nd, nr ← 0, rbuff ← []
- tSnkRx ← startThread(doRxDatTxAck())
- return mysid
input mysid.rx()
  • await (nd < nr)
    Seq db ← rbuff\textsubscript{nd};
    rbuff.remove(nd);
    nd++;
    return db;
function doRxDatTxAck()
  while (true)
    • Seq msg ← xL.rx();
      ia {msg in Tuple<DAT, 0..N−1, Seq>}
      await (true)
      int j ← nr + mod(msg[2]−nr, N);
      if ((nr ≤ j < nd+RW)
          and (not j in rbuff.keys))
        rbuffj ← msg[1];
        while (nr in rbuff.keys)
          nr++;
        xL.tx(aR,[ACK, mod(nr,N)])
Outline

Sliding Window Protocol
Analysis of Sliding Window Protocol
Await-structured Source and Sink Programs
Data transfer Protocol and Proof
Graceful-closing Data transfer Protocol
Abortable Data transfer Protocol
Program Dtp overview

- **DtpDist**: merge two SwpDist, but one lossy/lrd channel

- **Dtp**: Source + Sink, but share channel access
  - params, ia, constants: // same as src, snk

- **Source**
  - ng, ns, na, sbuff, timer
  - mysid.tx()
  - doTxDat(): thread tSrcTx // await{..xL.tx..}
  - doRxAck(): thread tSrcRx // xL.rx

- **Sink**
  - nd, nr, rbuff
  - mysid.rx
  - doRxDatTxAck: thread tSnkRx // xL.rx, await{..xL.tx..}
  - doRxDatAck(): thread tRx // xL.rx, await{..xL.tx..}
  - rcv data/ack msg, do doRxAck / doRxDatTxAck
Program Dtp – 1

- `params`: aL, aR, xL, N, SW, RW

- `main`
  - ng, ns, na, sbuff, timer
  - nd, nr, rbuff
  - `tSrcTx ← startThread(doTxDat())`
  - `tSnkRx ← startThread(doRxDatAck())`

- `input mysid.tx(aR, db)`
  
  <as in Source()>

- `input mysid.rx()`
  
  <as in Sink()>}

- `function doTxDat()`
  
  <as in Source()>
function doRxDatAck()

    while (true)
        Seq msg ← xL.rx();
        ia {msg in union(Tuple<ACK, 0..N-1>,
            Tuple<DAT, 0..N-1, Seq> }

        if (msg[0] = ACK)
            <Source.doRxAck() update>
        else if (msg[0] = ACK)
            <Sink.doRxDatTxAck() update>

atomicity assumption: awaits

progress assumption: weak fairness of threads
Proving that DtpDist implements Fifo Channel

- Obtain fifo channel inverse
  - only two addresses, so no internal nondeterminism

- Define program Z of DtpDist and fifo channel inverse

- Define assertions that Z must satisfy

- Proof that Z satisfies them follows from Swp properties
  - fifo-channel inverse \( \text{txh}_{a_1,a_2} = \text{aux var} \ Dtp(a_1).dbh \)
  - see text for details
Sliding Window Protocol
Analysis of Sliding Window Protocol
Await-structured Source and Sink Programs
Data transfer Protocol and Proof
Graceful-closing Data transfer Protocol
Abortable Data transfer Protocol
Adding graceful closing to Dtp

- Allow user to close the data transfer
  - open $\rightarrow$ closing $\rightarrow$ closed
  - after call: user cannot tx, but can rx
  - returns when
    - remote user also has requested closing
    - all data in both directions have been delivered to users

- New messages
  - [FIN]: indicates closing
  - [FINACK]: ack to [FIN]

- New flags, all initially false:
  - finRcvd
  - finAckRcvd
  - closed
input mysid.close()

- wait for na = ng;  // all outgoing data acked
- repeatedly send [FIN] until finAckRcvd is true;
- wait for finRcvd to be true;
- wait for nd = nr;  // no more incoming data
- set closed to true;
- return;
Augment doRxDatAck() to handle FIN and FINACK

Change part after “msg ← xL.rx()” to:

- if (not closed)
  - if msg is DAT or ACK: handle as before
  - if msg is FIN: set finRcvd, send [FINACK]
  - if msg is FINACK: set finAckRcvd
- else if (closed and msg is [FIN])
  - send [FINACK]

Modify mysid.rx

- return [0, data] if nr > nd  // incoming data
- return [-1] if finRcvd and nr = nd  // no more incoming data
Graceful-closing $\text{DtpDist}(j, k)$

- **Safety properties**
  - if $j.rx$ returns $[0,\text{data}]$: $j.dr\times h \circ [\text{data}]$ prefix-of $k.dt\times h$
  - if $j.rx$ returns $[-1]$ or $J$ is closed:
    $j.dr\times h = k.dt\times h$ and $k$ is closing or closed.

- **Progress properties**
  - $j.tx$ ongoing $\implies$ $\text{leads-to}$ $j.tx$ returns
  - $j.rx$ ongoing, $j.dr\times h \neq k.dt\times h$ $\implies$ $\text{leads-to}$ $j.rx$ returns
  - $j.rx$ ongoing, $j.dr\times h = k.dt\times h$, $k$ is closing or closed
    $\implies$ $\text{leads-to}$ $j.rx$ returns
  - $j$ closing, $k$ is closing or closed
    $\implies$ $\text{leads-to}$ $j$ becomes closed or $j.rx$ not ongoing
Outline

Sliding Window Protocol
Analysis of Sliding Window Protocol
Await-structured Source and Sink Programs
Data transfer Protocol and Proof
Graceful-closing Data transfer Protocol
Abortable Data transfer Protocol
Abortable Dtp

- Graceful-closing terminates connection
  - but not dtp systems or lossy/lrd channel
  - closed dtp system still needs to respond to FIN

- To close dtp systems and channel, need to modify Dtp
  - **abort** if FINACK not rcvd within some $K$ sends of FIN
  - can do the same for data transfer
  - abort: return guest threads, retrieve local threads, end system

- Abortable DtpDist provides a weaker service
  - data delivered is prefix of data sent
    // still holds
  - all data sent is delivered when close returns
  - all data sent is eventually delivered