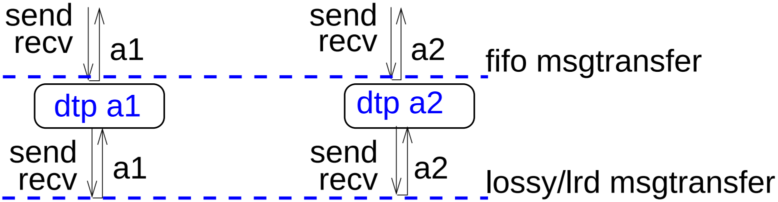
Implementing fifo msgtransfer2 using lossy or LRD msgtransfer2

Shankar

March 12, 2019

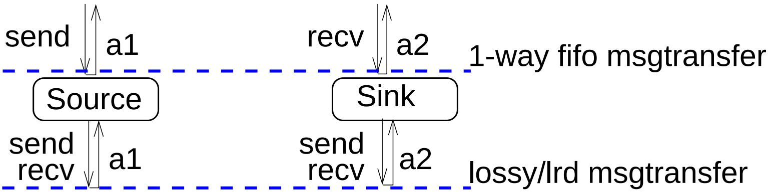
Data transfer protocol: DtpDist



- Implements fifo msgtransfer(a1,a2) using lossy|lrd msgtransfer
- Sliding window protocol (Swp)
  - a1 $\rightarrow$ a2 fifo transfer
  - define programs at "algorithm-level": atomic rules
  - prove correctness with modulo-N sequence numbers
  - correctness-preserving refinement to await program
- Obtain dtp via correctness-preserving merge of two Swps

Outline

Sliding window protocol: SwpDist



- Swp(a1, a2)

```
  x_a1, x_a2  ←  startSystem(Lossy|LrdChannel(a1, a2))
  y_a1 ← startSystem(Source(a1, a2, x_a1))
  y_a2 ← startSystem(Sink(a2, a1, x_a2))
  return y_a1, y_a2
```

Solution using sequence numbers

- $d_0, d_1, \cdots$: data blocks sent into source by its user
- Source
  - send $[d_k, k]$ repeatedly until acked // k: seq #
- Sink
  - respond with awaited seq # // cumulative ack
    - has $0 \ldots 4 \sqcup 6, \ldots 8$: recv 7 / send 5; recv 5 / send 9
  - passes data blocks to its user in order
- For good throughput
  - > 1 outstanding at source, buffer out-of-sequence at sink
- Above requires unbounded seq #s (usn)
  - not good for hw implementation
- Instead use cyclic seq #s (csn)

Using modulo-N cyclic sequence numbers

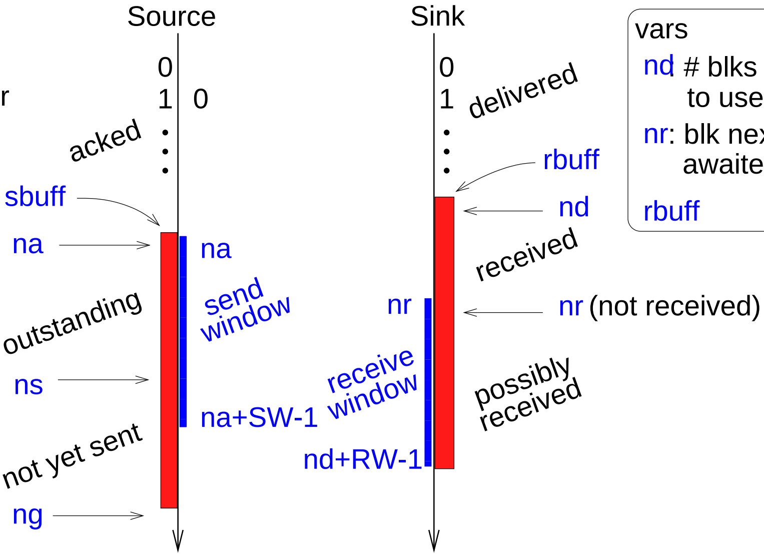
- Use mod-N csn instead of usn
- Send csn $\mathrm{mod}(k, N)$ instead of usn k
- Receiver of *csn* has to infer the corresponding *usn*
  - maintains window of "possible" usn
    - say $L \cdots U$
  - maps rcvd *csn* to *usn* with same cyclic value
    - $usn \leftarrow L + \mathrm{mod}(csn - L,\ N)$;
      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 $\cdots$ ng-1
  - send window: na $\cdots$ ns$+$SW-1 // SW $<$ N
  - outstanding window: na $\cdots$ ns // ok if low end is na+1
- Sink
  - nd: # blks to user
  - nr: # contiguous blks rcvd
  - map rbuff: for blks nd $\cdots$ nd$+$RW-1 // RW $<$ N
  - receive window: nr $\cdots$ nd$+$RW-1
- Over time, windows slide to increasing seq #s

Sliding windows

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



vars
nd: # blks to user
nr: blk next awaited
rbuff

Swp mod-N: algorithm-level

Source

na,ns,ng ← 0; sbuff ← []

- db from user:
  sbuff$_{ng}$ ← db; ng++
- ns < min (na+SW, ng):
  send [sbuff$_{ns}$, $\overline{\text{ns}}$]; ns++
- k in na..ns-1:
  send [sbuff$_k$, $\overline{\text{k}}$]
- recv [cn]:
  j ← na + $\overline{\text{cn-na}}$
  if na < j ≤ ns:
    sbuff.remove(na..j-1)
    na ← j

Sink

nd,nr ← 0; rbuff ← []

- nd < nr:
  rbuff$_{nd}$ to user; nd++
- recv [db,cn]:
  j ← nr + $\overline{\text{cn-nr}}$
  if nr ≤ j < nd+RW:
    rbuff$_j$ ← db
    while nr in rbuff.keys:
      nr ← nr+1
  send [$\overline{\text{nr}}$]

$\boxed{\overline{k}\text{: mod}(k,N)}$

Outline

Desired properties of SwpDist

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

- Desired properties

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

  X_2 : (nd = k < ng *leads-to* nd > k)

  assuming wfair send sbuff_{ns}, resend sbuff_{na},
  deliver rbuff_{nd} to user,
  source/sink rcv msg,
  channel progress

- Intermediate desired Inv A_1–A_3

  A_0 : nd $\cdots$ nr-1 in rbuff.keys

  A_1 : (k in rbuff.keys) $\Rightarrow$ $\text{rbuff}_{\text{k}} = \text{dbh}_{\text{k}}$ // $A_{0,1} \Rightarrow X_1$.pred

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

Correct interpretation conditions

- Add *auxiliary usn* field to msgs // not read
  - source sends [$\mathtt{sbuff}_j$, $\bar{j}$, j]
  - sink sends [$\overline{nr}$, nr]
- Sink maps rcvd [$\bar{j}$, j] wrt rcv window nr $\cdots$ nr+RW-1
  - j in window: mapped correctly
  - j < window: nr-1 ✓, nr-2 ✓, $\cdots$ ✓, nr-N+RW-1 ✗
  - j > window: nr+RW ✓, nr+RW+1 ✓, $\cdots$ ✓, nr+N ✗
  - desired $Inv\ A_3$

    $\boxed{A_3 : \text{ data j rcvable } \Rightarrow \text{ j in nr-N+RW } \cdots \text{ nr+N-1}}$
- Source maps rcvd [$\bar{j}$, j] wrt outstanding window na $\cdots$ ns
  - desired $Inv\ A_3$

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

Recap

- Goal

  A_0 : nd $\cdots$ nr-1 in rbuff.keys

  A_1 : (k in rbuff.keys) $\Rightarrow$ rbuff$_k$ $=$ dbh$_k$

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

- Correct interpretation

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

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

- For every step e: $\{A_0 - A_4\}\ e\ \{A_0, A_1, A_2\}$ holds
- Suffices to establish
  - $\{A_{2,3,4}\}\ e\ \{A_3\}$ for e: send data; rcv data affecting nr
  - $\{A_{2,3,4}\}\ e\ \{A_4\}$ for e: send ack; rcv ack affecting ns

Outline

Analysis for Lossy Channel

- For `SwpDist` with lossy channel
  - now show that $\mathit{Inv}\ A_3$–A_4 holds if $\mathtt{N} \geq \mathtt{SW} + \mathtt{RW}$

Preserving A_3 wrt send data

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

- Send data k preserves A_3.rhs lower bound if N $\geq$ SW + RW
  - k $\geq$ na // guards, A_2
    $\geq$ ns - SW // A_2
    $\geq$ nr - SW // A_2
    $\geq$ nr - N + RW // N $\geq$ SW + RW
- Send data k preserves A_3.rhs upper bound
  - k $\leq$ na + SW - 1 // guards, A_2
    $\leq$ nr + SW - 1 // A_2
    $\leq$ nr + N - 1 // N $\geq$ SW

Preserving A_3 wrt *nr* increase

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

- 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 $\star\star$ // 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 // $\star\star$
      $\geq$ k+1−N+RW // N $\geq$ SW + RW
- Above is an operational proof; see text for an assertional proof

Preserving A_4

A_2 : na $\leq$ nr $\leq$ ns $\leq$ na+SW
A_4 : ack j rcvable $\Rightarrow$ j in ns−N+1 $\cdots$ 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 reodering
  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
- Above is an operational proof; see text for an assertional proof

Analysis for LRD Channel

- For `SwpDist` with LRD channel
  - obviously $Inv\ A_3$–A_4 does not hold for any N
- But $Inv\ A_3$–A_4 holds if
  - LRD channel has max msg lifetime L
  - min time δ between *ns* increments
  - $\mathtt{N} \geq \mathtt{SW} + \mathtt{RW} + \frac{L}{\delta}$

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/δ // guards

5 j $\geq$ ns(t_0) - L/δ - SW // 3, 4

6 j $\geq$ ns(t_0) - N + RW // 5, N $\geq$ SW + RW + L/δ

7 j $\geq$ nr(t_0) - N + RW // 6, A_2

Preserving A_4

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

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

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

1 let ack j be rcvable at t_0

2 ack j was sent at some $t_1 > t_0 - L$ // 1

3 j $=$ nr(t_1) $\geq$ na(t_1) $\geq$ ns(t_1) - SW // 2, guard, A_2

4 during $[t_1, t_0]$: ns increases by at most L/δ

5 j $\geq$ ns(t_0) - L/δ - SW // 3, 4

6 j $\geq$ ns(t_0) - N + 1 // 5, N $\geq$ SW + RW + L/δ, RW > 0

Outline

Desired progress of Swp

X_2 : (nd $=$ k $<$ ng   *leads-to*   nd $>$ k)

assuming

- wfair send (of $\mathtt{sbuff}_{\mathsf{ns}}$)
- wfair resend (of $\mathtt{sbuff}_{\mathsf{na}}$)
- wfair sink user recv (of $\mathtt{rbuff}_{\mathsf{nd}}$)
- wfair source recv (of msg from lossy/lrd channel)
- wfair sink recv (of msg from lossy/lrd channel)
- lossy/lrd channel progress

Progress of sliding windows

P_1 : na $=$ k $<$ nr $\leq$ ns *leads-to* na $>$ k
// src resend, snk recv, src recv, channel progress

P_2 : (na $=$ k $=$ nr $<$ ns and nr $<$ nd+RW) *leads-to* nr $>$ k
// src resend, snk recv, channel progress

P_3 : (na $=$ k $=$ nr $<$ ns and nr $=$ nd+RW)
leads-to nr $>$ k // sink user recv, 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, *Inv* nr $\geq$ na

P_7 : (ns $=$ k $<$ ng and ns $<$ na+SW) *leads-to* ns $>$ k
// src send

P_8 : (ns $=$ k $<$ ng and ns $=$ na+SW) *leads-to*
(ns $=$ k $<$ ng 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}, sink user recv

Outline

Correctness-preserving transformation

- 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

Source program – 1

- 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.send: get db from user
    - doSendDat: send ns, resend k // calls xL.send
    - doRecvAck: rcv ack msg // calls xL.recv
  - main
    - ng, ns, na ← 0, sbuff ← []
    - timer ← OFF // OFF, 0, 1, $\cdots$
    - tSrcSend ← startThread(doSendDat())
    - tSrcRecv ← startThread(doRecvAck())
    - return mysid

- input mysid.send (aR, db):
    await True:
      $sbuff_{ng}$ ← db
      ng++
      return

- function doSendDat():
    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.send (aR, [DAT, $sbuff_j$, mod(j,N)])
        timer ← 0 // (re)start timer

- function void doRecvAck():

```
while True:
  • Seq msg ← xL.recv();
    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

Source program: improving performance

- Send new data blocks asap, instead of upon timeout
- Reduce extent of blocking at awaits
  - concurrent access to sbuff
  - duplicate counters
  - $\ldots$
- Adapt RTO to measured roundtrip time // congestion control

Sink program – 1

- 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.recv: deliver db to user
    - doRecvDatSendAck: rcv data, send ack // calls xL.recv, xL.send
  - main
    - nd, nr ← 0, rbuff ← []
    - tSnkRecv ← startThread(doRecvDatSendAck())
    - return mysid

Sink program – 2

- `input mysid.recv():`
  - `await nd < nr:`

    Seq db $\leftarrow$ $\mathtt{rbuff}_{\mathtt{nd}}$

    `rbuff.remove(nd)`

    `nd++`

    `return db`

Sink program – 3

- function doRecvDatSendAck():

```
    while True:
      • Seq msg ← xL.recv();
        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):
                rbuff_j ← msg[1]
                while nr in rbuff.keys:
                    nr++
            xL.send(aR, [ACK, mod(nr,N)])
```

- atomicity assumption: awaits
- progress assumption: weak fairness of threads

Outline

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.send()`
    - `doSendDat()`: thread `tSrcSend` // `await{.. xL.send ..}`
    - ~~`doRecvAck()`~~: thread ~~`tSrcRecv`~~ // `xL.recv`
  - Sink
    - nd, nr, `rbuff`
    - `mysid.recv`
    - ~~`doRecvDatSendAck`~~: thread ~~`tSnkRecv`~~ // `xL.recv`, `await{.. xL.send ..}`
  - `doRecvDatAck()`: thread `tRecv` // `xL.recv`, `await{.. xL.send ..}`
    - rcv data/ack msg, do doRecvAck / doRecvDatSendAck

Program Dtp – 1

- params: aL, aR, xL, N, SW, RW
- main
  - ng, ns, na, sbuff, timer
  - nd, nr, rbuff
  - tSrcSend ← startThread (doSendDat())
  - tSnkRecv ← startThread (doRecvDatAck())
- input mysid.send (aR, db)
  <as in Source()>
- input mysid.recv ()
  <as in Sink()>
- function doSendDat()
  <as in Source()>

Program Dtp – 2

- function doRecvDatAck():

```
    while True:
      • Seq msg ← xL.recv();
        ia {msg in union (Tuple<ACK, 0..N-1>,
                          Tuple<DAT, 0..N-1, Seq>}
      if msg[0] == ACK:
         <Source.doRecvAck() update>
      elif msg[0] == ACK:
         <Sink.doRecvDatSendAck() 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 $\mathtt{txh}_{\mathtt{a1},\mathtt{a2}}$ = aux var `Dtp(a1).dbh`
  - see text for details

Outline

Adding graceful closing to Dtp

- Allow user to close the data transfer
  - open $\rightarrow$ closing $\rightarrow$ closed
  - after call: user cannot send, but can recv
  - 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

Graceful-closing Dtp – 1

- 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;

Graceful-closing Dtp – 2

- Augment `doRecvDatAck()` to handle `FIN` and `FINACK`
  Change part after "`msg` $\leftarrow$ `xL.recv()`" 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.recv`
  - return `[0,data]` if nr > nd // incoming data
  - return `[-1]` if `finRcvd` and nr = nd // no more incoming data

Graceful-closing DtpDist(j, k)

- Safety properties
  - if `j.recv` returns `[0,data]`: `j.drxh` $\circ$ `[data]` prefix-of `k.dtxh`
  - if `j.recv` returns `[-1]` or `j` is closed:
    `j.drxh` $=$ `k.dtxh` and `k` is closing or closed.
- Progress properties
  - `j.send` ongoing *leads-to* `j.send` returns
  - `j.recv` ongoing, `j.drxh` $\neq$ `k.dtxh` *leads-to* `j.recv` returns
  - `j.recv` ongoing, `j.drxh` $=$ `k.dtxh`, `k` is closing or closed
    leads-to `j.recv` returns
  - `j` closing, `k` is closing or closed
    leads-to `j` becomes closed or `j.recv` not ongoing

Outline

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~~