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 → 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

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

**SwpDist**

- **Swp**($a_1, a_2$)
  
  $x_{a1}, x_{a2} \leftarrow \text{startSystem}([\text{Lossy} | \text{LrdChannel}(a_1, a_2)])$
  
  $y_{a1} \leftarrow \text{startSystem}([\text{Source}(a_1, a_2, x_{a1})])$
  
  $y_{a2} \leftarrow \text{startSystem}([\text{Sink}(a_2, a_1, x_{a2})])$
  
  return $y_{a1}, y_{a2}$
Solution using sequence numbers

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

**Source**
- send $[d_k, k]$ repeatedly until acked \hspace{1cm} // $k$: seq #

**Sink**
- respond with **awaited** seq # \hspace{1cm} // cumulative ack
  - has $0 \ldots 4 \sqsubseteq 6, \ldots 8$: 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 $\text{mod}(k, N)$ 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
    - $\text{usn} \leftarrow L + \text{mod}(\text{csn} - L, N)$;
    - if $\text{usn} > U$ 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

Source
- 0
- 1
- 0

acked

Sink
- 0
- 1
delivered

nd

rbuff

nd

nr: blk next awaited

nr (not received)

not yet sent

na

send window

receive window

rsbuff

received

possibly received

outstanding

ns

na

na+SW-1

nd+RW-1

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

# blks

# sent

# acked

0 from user

vars

# blks

# sent

# acked

0 from user
Swp mod-N: algorithm-level

Source
\[\text{na, ns, ng} \leftarrow 0; \text{sbuff} \leftarrow []\]
- \text{db from user:}
  \text{sbuff}\text{ng} \leftarrow \text{db}; \text{ng}++
- \text{ns} < \min (\text{na+SW}, \text{ng}):
  \text{send} [\text{sbuff}\text{ns}, \overline{\text{ns}}]; \text{ns}++
- \text{k in na..ns-1:}
  \text{send} [\text{sbuff}\text{k}, \overline{\text{k}}]
- \text{recv [cn]:}
  \text{j} \leftarrow \text{na + cn-na}
  \text{if na} < \text{j} \leq \text{ns:}
    \text{sbuff.remove(na..j-1)}
  \text{na} \leftarrow \text{j}

Sink
\[\text{nd, nr} \leftarrow 0; \text{rbuff} \leftarrow []\]
- \text{nd} < \text{nr:}
  \text{rbuff}\text{nd} \text{to user; nd}++
- \text{recv [db, cn]:}
  \text{j} \leftarrow \text{nr + cn-nr}
  \text{if nr} \leq \text{j} < \text{nd+RW:}
    \text{rbuff}\text{j} \leftarrow \text{db}
    \text{while nr in rbuff.keys:}
    \text{nr} \leftarrow \text{nr}+1
    \text{send} [\overline{\text{nr}}]
\overline{\text{k}} \leftarrow \text{mod}(\text{k}, \text{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

\[ X_1: Inv \ (nd < nr \Rightarrow rbuff_{nd} = dbh_{nd}) \]

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

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

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

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

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

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

- Add auxiliary *usn* field to msgs
  - source sends \([sbuff_j, j, j]\)
  - sink sends \([\overline{nr}, nr]\)

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

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

\[ A_0 : \text{nd} \cdots \text{nr-1 in rbuff.keys} \]
\[ A_1 : (k \text{ in rbuff.keys}) \Rightarrow rbuff_k = \text{dbh}_k \]
\[ A_2 : \text{na} \leq \text{nr} \leq \text{ns} \leq \text{na+SW} \]

Correct interpretation

\[ A_3 : \text{data j rcvable} \Rightarrow j \text{ in } \text{nr-N+RW} \cdots \text{nr+N-1} \]
\[ A_4 : \text{ack j rcvable} \Rightarrow j \text{ in } \text{ns-N+1} \cdots \text{na+N} \]

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

- 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 $\text{Inv } A_3 - A_4$ holds if $N \geq SW + RW$
Preserving $A_3$ wrt send data

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

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

- **Send data k preserves $A_3$.rhs lower bound** if $N \geq \text{SW} + \text{RW}$
  - $k \geq \text{na}$  
    \[ \geq \text{ns - SW} \]  
    \[ \geq \text{nr - SW} \]  
    \[ \geq \text{nr - N + RW} \]  
  
- **Send data k preserves $A_3$.rhs upper bound**  
  - $k \leq \text{na + SW - 1}$  
    \[ \leq \text{nr + SW - 1} \]  
    \[ \leq \text{nr + N - 1} \]  

// guards, $A_2$  
// $A_2$  
// $A_2$  
// $N \geq \text{SW} + \text{RW}$  
// guards, $A_2$  
// $A_2$  
// $N \geq \text{SW}$
Preserving $A_3$ wrt $nr$ increase

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

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

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

- Above is an operational proof; see text for an assertional proof
Preserving $A_4$

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

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

- 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 $j$ receivable $\Rightarrow$ 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
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 LRD channel
  - obviously $\text{Inv } A_3\text{--}A_4$ does not hold for any $N$

But $\text{Inv } A_3\text{--}A_4$ holds if
  - LRD channel has max msg lifetime $L$
  - min time $\delta$ between $ns$ increments
  - $N \geq SW + RW + \frac{L}{\delta}$
Preserving $A_3$

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

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

- $A_3.rhs$ 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 $t_1 > t_0 - L$ \hspace{1cm} // 1
  3. $j \geq \text{na}(t_1) \geq \text{ns}(t_1) - \text{SW}$ \hspace{1cm} // guards, $A_2$
  4. during $[t_1, t_0]$: ns increases by at most $L/\delta$ \hspace{1cm} // guards
  5. $j \geq \text{ns}(t_0) - L/\delta - \text{SW}$ \hspace{1cm} // 3, 4
  6. $j \geq \text{ns}(t_0) - N + \text{RW}$ \hspace{1cm} // 5, $N \geq \text{SW} + \text{RW} + L/\delta$
  7. $j \geq \text{nr}(t_0) - N + \text{RW}$ \hspace{1cm} // 6, $A_2$
Preserving $A_4$

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

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

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

- $A_4.rhs$ lower bound holds as follows
  1. let $\text{ack } j$ be rcvable at $t_0$
  2. $\text{ack } j$ was sent at some $t_1 > t_0 - L$  \hspace{1cm} // 1
  3. $j = nr(t_1) \geq na(t_1) \geq ns(t_1) - SW$ \hspace{1cm} // 2, guard, $A_2$
  4. during $[t_1, t_0]$: $\text{ns}$ increases by at most $L/\delta$  
  5. $j \geq ns(t_0) - L/\delta - SW$ \hspace{1cm} // 3, 4
  6. $j \geq ns(t_0) - N + 1$ \hspace{1cm} // 5, $N \geq SW + RW + L/\delta$, $RW > 0$
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
$X_2 : (\text{nd} = k < \text{ng} \quad \textit{leads-to} \quad \text{nd} > k)$

assuming

- wfair send (of sbuff$_{ns}$)
- wfair resend (of sbuff$_{na}$)
- wfair sink user recv (of rbuff$_{nd}$)
- 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

\[ P_1 : \text{na} = k < \text{nr} \leq \text{ns} \quad \text{leads-to} \quad \text{na} > k \]

// src resend, snk recv, src recv, channel progress

\[ P_2 : (\text{na} = k = \text{nr} < \text{ns} \quad \text{and} \quad \text{nr} < \text{nd+RW}) \quad \text{leads-to} \quad \text{nr} > k \]

// src resend, snk recv, channel progress

\[ P_3 : (\text{na} = k = \text{nr} < \text{ns} \quad \text{and} \quad \text{nr} = \text{nd+RW}) \]

\[ \text{leads-to} \quad \text{nr} > k \quad \text{// sink user recv, } P_2 \]

\[ P_4 : \text{na} = k = \text{nr} < \text{ns} \quad \text{leads-to} \quad \text{nr} > k \quad \text{// } P_2, P_3 \]

\[ P_5 : \text{na} = k < \text{ns} \quad \text{leads-to} \quad \text{na} > k \quad \text{// } P_4, P_1 \]

\[ P_6 : \text{nr} = k < \text{ns} \quad \text{leads-to} \quad \text{nr} > k \quad \text{// } P_5, \text{Inv } \text{nr} \geq \text{na} \]
\[ P_7 : (ns = k < ng \text{ and } ns < na + SW) \text{ leads-to } ns > k \quad \text{ // src send} \]

\[ P_8 : (ns = k < ng \text{ and } ns = na + SW) \text{ leads-to} \]
\[ (ns = k < ng \text{ and } ns < na + SW) \quad \text{ // } P_5 \]

\[ P_9 : ns = k < ng \text{ leads-to } ns > k \quad \text{ // } P_7, P_8 \]

\[ P_{10} : nr = k < ng \text{ leads-to } nr > k \quad \text{ // } P_9, P_6 \]

\[ P_{11} : nd = k < ng \text{ leads-to } nd > k \quad \text{ // } P_{10}, \text{ sink user recv} \]
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
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
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, ...
- tSrcSend ← startThread(doSendDat())
- tSrcRecv ← startThread(doRecvAck())
- return mysid
input mysid.send(aR, db):
    await True:
        sbuffng ← 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, sbuffj, 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
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.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
input mysid.recv():
   • await nd < nr:
     Seq db ← rbuff
     rbuff.remove(nd)
     nd++
     return db
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):
                    rbuffj ← 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

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.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()>
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
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 $txh_{a1,a2} = aux \ var \ Dtp(a1).dbh$
  - see text for details
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
Adding graceful closing to Dtp

- Allow user to close the data transfer
  - open → closing → 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
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 doRecvDatAck() to handle FIN and FINACK
Change part after “msg ← 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 \(j.\text{recv}\) returns \([0,\text{data}]\): \(j.\text{drxh} \circ [\text{data}]\) prefix-of \(k.\text{dtxh}\)
  - if \(j.\text{recv}\) returns \([-1]\) or \(j\) is closed:
    \(j.\text{drxh} = k.\text{dtxh}\) and \(k\) is closing or closed.

- **Progress properties**
  - \(j.\text{send}\) ongoing \(\text{leads-to}\) \(j.\text{send}\) returns
  - \(j.\text{recv}\) ongoing, \(j.\text{drxh} \neq k.\text{dtxh}\) \(\text{leads-to}\) \(j.\text{recv}\) returns
  - \(j.\text{recv}\) ongoing, \(j.\text{drxh} = k.\text{dtxh}\), \(k\) is closing or closed
    \(\text{leads-to}\) \(j.\text{recv}\) returns
  - \(j\) closing, \(k\) is closing or closed
    \(\text{leads-to}\) \(j\) becomes closed or \(j.\text{recv}\) 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